



macromedia®

FLASH™

Components Language Reference

8

Trademarks

1 Step RoboPDF, ActiveEdit, ActiveTest, Authorware, Blue Sky Software, Blue Sky, Breeze, Breezo, Captivate, Central, ColdFusion, Contribute, Database Explorer, Director, Dreamweaver, Fireworks, Flash, FlashCast, FlashHelp, Flash Lite, FlashPaper, Flash Video Encoder, Flex, Flex Builder, Fontographer, FreeHand, Generator, HomeSite, JRun, MacRecorder, Macromedia, MXML, RoboEngine, RoboHelp, RoboInfo, RoboPDF, Roundtrip, Roundtrip HTML, Shockwave, SoundEdit, Studio MX, UltraDev, and WebHelp are either registered trademarks or trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words, or phrases mentioned within this publication may be trademarks, service marks, or trade names of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

Third-Party Information

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

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



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

Opera ® browser Copyright © 1995-2002 Opera Software ASA and its suppliers. All rights reserved.

Macromedia Flash 8 video is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Visual SourceSafe is a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Copyright © 2005 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without written approval from Macromedia, Inc. Notwithstanding the foregoing, the owner or authorized user of a valid copy of the software with which this manual was provided may print out one copy of this manual from an electronic version of this manual for the sole purpose of such owner or authorized user learning to use such software, provided that no part of this manual may be printed out, reproduced, distributed, resold, or transmitted for any other purposes, including, without limitation, commercial purposes, such as selling copies of this documentation or providing paid-for support services.

Acknowledgments

Project Management: Sheila McGinn

Writing: Bob Berry, Jen deHaan, Peter deHaan, David Jacowitz, Wade Pickett

Managing Editor: Rosana Francescato

Lead Editor: Lisa Stanziano

Editing: Evelyn Eldridge, Mary Ferguson, Mary Kraemer, Jessie Wood

Production Management: Patrice O'Neill, Kristin Conradi, Yuko Yagi

Media Design and Production: Adam Barnett, Aaron Begley, Paul Benkman, John Francis, Geeta Karmarkar, Masayo Noda, Paul Rangel, Arena Reed, Mario Reynoso

Special thanks to Jody Bleyle, Mary Burger, Lisa Friendly, Stephanie Gowin, Bonnie Loo, Nivesh Rajbhandari, Mary Ann Walsh, Erick Vera, the beta testers, and the entire Flash and Flash Player engineering and QA teams.

First Edition: September 2005

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

Contents

Chapter 1: Components Dictionary	29
Types of components	30
Other listings in this chapter	33
Chapter 2: Accordion component (Flash Professional only)	35
Using the Accordion component (Flash Professional only)	36
Customizing the Accordion component (Flash Professional only)	40
Accordion class (Flash Professional only)	47
Accordion.change	51
Accordion.createChild()	53
Accordion.createSegment()	55
Accordion.destroyChildAt()	57
Accordion.getChildAt()	58
Accordion.getHeaderAt()	59
Accordion.numChildren	60
Accordion.selectedChild	61
Accordion.selectedIndex	62
Chapter 3: Alert component (Flash Professional only)	65
Using the Alert component (Flash Professional only)	66
Customizing the Alert component (Flash Professional only)	67
Alert class (Flash Professional only)	71
Alert.buttonHeight	76
Alert.buttonWidth	76
Alert.CANCEL	77
Alert.cancelLabel	78
Alert.click	79
Alert.NO	80
Alert.noLabel	81
Alert.NONMODAL	82
Alert.OK	83
Alert.okLabel	84
Alert.show()	84
Alert.YES	86
Alert.yesLabel	87

Chapter 4: Button component	89
Using the Button component	90
Customizing the Button component	94
Button class	101
Button.icon	106
Button.label	107
Button.labelPlacement	108
Chapter 5: CellRenderer API	109
Understanding the List class	109
Using the CellRenderer API	111
CellRenderer.getCellIndex()	118
CellRenderer.getDataLabel()	119
CellRenderer.getPreferredHeight()	120
CellRenderer.getPreferredWidth()	121
CellRenderer.listOwner	122
CellRenderer.owner	123
CellRenderer.setSize()	123
CellRenderer.setValue()	124
Chapter 6: CheckBox component	129
Using the CheckBox component	130
Customizing the CheckBox component	132
CheckBox class	135
CheckBox.click	140
CheckBox.label	142
CheckBox.labelPlacement	143
CheckBox.selected	145
Chapter 7: Collection interface (Flash Professional only)	147
Collection class (Flash Professional only)	147
Collection.addItem()	148
Collection.contains()	149
Collection.clear()	150
Collection.getItemAt()	151
Collection.getIterator()	152
Collection.getLength()	153
Collection.isEmpty()	153
Collection.removeItem()	154

Chapter 8: ComboBox component	157
Using the ComboBox component	159
Customizing the ComboBox component	162
ComboBox class	165
ComboBox.addItem()	171
ComboBox.addItemAt()	172
ComboBox.change	173
ComboBox.close()	174
ComboBox.close	175
ComboBox.dataProvider	176
ComboBox.dropdown	178
ComboBox.dropdownWidth	179
ComboBox.editable	179
ComboBox.enter	181
ComboBox.getItemAt()	182
ComboBox.itemRollOut	183
ComboBox.itemRollOver	185
ComboBox.labelField	186
ComboBox.labelFunction	187
ComboBox.length	188
ComboBox.open()	188
ComboBox.open	189
ComboBox.removeAll()	191
ComboBox.removeItemAt()	192
ComboBox.replaceItemAt()	193
ComboBox.restrict	194
ComboBox.rowCount	196
ComboBox.scroll	197
ComboBox.selectedIndex	199
ComboBox.selectedItem	200
ComboBox.sortItems()	201
ComboBox.sortItemsBy()	202
ComboBox.text	205
ComboBox.textField	205
ComboBox.value	206

Chapter 9: Data binding classes (Flash Professional only) . . . 207

Making data binding classes available at runtime (Flash Professional only)	207
Classes in the mx.data.binding package (Flash Professional only)	208
Binding class (Flash Professional only)	208
Constructor for the Binding class	209
Binding.execute()	211
CustomFormatter class (Flash Professional only)	212
CustomFormatter.format()	214
CustomFormatter.unformat()	215
CustomValidator class (Flash Professional only)	216
CustomValidator.validate()	217
CustomValidator.validationError()	219
EndPoint class (Flash Professional only)	220
Constructor for the EndPoint class	222
EndPoint.component	222
EndPoint.constant	223
EndPoint.event	223
EndPoint.location	224
EndPoint.property	226
ComponentMixins class (Flash Professional only)	226
ComponentMixins.getField()	227
ComponentMixins.initComponent()	228
ComponentMixins.refreshDestinations()	229
ComponentMixins.refreshFromSources()	230
ComponentMixins.validateProperty()	230
Data Type class (Flash Professional only)	233
Data Type.encoder	234
Data Type.formatter	235
Data Type.getAnyTypedValue()	236
Data Type.getAsBoolean()	237
Data Type.getAsNumber()	238
Data Type.getAsString()	238
Data Type.getTypedValue()	239
Data Type.kind	240
Data Type.setAnyTypedValue()	240
Data Type.setAsBoolean()	241
Data Type.setAsNumber()	242
Data Type.setAsString()	243
Data Type.setTypedValue()	244
TypedValue class (Flash Professional only)	245
Constructor for the TypedValue class	246

TypedValue.type.....	246
TypedValue.typeName	247
TypedValue.value.....	247

Chapter 10: DataGrid component (Flash Professional only) . . 249

Interacting with the DataGrid component (Flash Professional only)	250
Using the DataGrid component (Flash Professional only)	251
DataGrid performance strategies	256
Customizing the DataGrid component (Flash Professional only) ..	258
DataGrid class (Flash Professional only)	262
DataGrid.addColumn()	268
DataGrid.addColumnAt()	269
DataGrid.addItem()	270
DataGrid.addItemAt()	271
DataGrid.cellEdit	272
DataGrid.cellFocusIn.....	274
DataGrid.cellFocusOut	275
DataGrid.cellPress	277
DataGrid.change	278
DataGrid.columnCount	280
DataGrid.columnNames	280
DataGrid.columnStretch	281
DataGrid.dataProvider	282
DataGrid.editable	283
DataGrid.editField()	285
DataGrid.focusedCell	286
DataGrid.getColumnAt()	287
DataGrid.getColumnIndex().....	288
DataGrid.headerHeight	289
DataGrid.headerRelease	289
DataGrid.hScrollPolicy	291
DataGrid.removeAllColumns()	292
DataGrid.removeColumnAt().....	293
DataGrid.replaceItemAt()	294
DataGrid.resizableColumns	295
DataGrid.selectable	296
DataGrid.showHeaders	297
DataGrid.sortableColumns.....	298
DataGrid.spaceColumnsEqually()	299
DataGridColumn class (Flash Professional only)	300
Constructor for the DataGridColumn class.....	302
DataGridColumn.cellRenderer	303

DataGridColumn.columnName	303
DataGridColumn.editable	304
DataGridColumn.headerRenderer	306
DataGridColumn.headerText	306
DataGridColumn.labelFunction	307
DataGridColumn.resizable	308
DataGridColumn.sortable	309
DataGridColumn.sortOnHeaderRelease	310
DataGridColumn.width	311

Chapter 11: DataHolder component (Flash Professional only) 313

Creating an application with the DataHolder component (Flash Professional only)	314
DataHolder class	315
DataHolder.data	316

Chapter 12: DataProvider API 317

DataProvider class	317
DataProvider.addItem()	319
DataProvider.addItemAt()	319
DataProvider.editField()	320
DataProvider.getEditingData()	321
DataProvider.getItemAt()	321
DataProvider.getItemID()	322
DataProvider.length	323
DataProvider.modelChanged	323
DataProvider.removeAll()	325
DataProvider.removeItemAt()	325
DataProvider.replaceItemAt()	326
DataProvider.sortItems()	327
DataProvider.sortItemsBy()	328

Chapter 13: DataSet component (Flash Professional only) 331

Using the DataSet component	331
DataSet class (Flash Professional only)	335
DataSet.addItem	338
DataSet.addItem()	340
DataSet.addItemAt()	342
DataSet.addSort()	343
DataSet.afterLoaded	346
DataSet.applyUpdates()	347

DataSet.calcFields.....	348
DataSet.changesPending().....	349
DataSet.clear().....	350
DataSet.createItem().....	351
DataSet.currentItem.....	352
DataSet.dataProvider.....	353
DataSet.deltaPacket.....	354
DataSet.deltaPacketChanged.....	355
DataSet.disableEvents().....	356
DataSet.enableEvents().....	357
DataSet.filtered.....	359
DataSet.filterFunc.....	361
DataSet.find().....	364
DataSet.findFirst().....	366
DataSet.findLast().....	367
DataSet.first().....	369
DataSet.getItemId().....	370
DataSet.getIterator().....	371
DataSet.getLength().....	373
DataSet.hasNext().....	373
DataSet.hasPrevious().....	374
DataSet.hasSort().....	375
DataSet.isEmpty().....	376
DataSet.items.....	377
DataSet.itemClassName.....	378
DataSet.iteratorScrolled.....	378
DataSet.last().....	380
DataSet.length.....	381
DataSet.loadFromSharedObj().....	382
DataSet.locateById().....	384
DataSet.logChanges.....	385
DataSet.modelChanged.....	386
DataSet.newItem.....	389
DataSet.next().....	390
DataSet.previous().....	391
DataSet.properties.....	392
DataSet.readOnly.....	392
DataSet.removeAll().....	393
DataSet.removeItem.....	394
DataSet.removeItem().....	396
DataSet.removeItemAt().....	397
DataSet.removeRange().....	398

DataSet.removeSort()	399
DataSet.resolveDelta	401
DataSet.saveToSharedObj()	402
DataSet.schema	404
DataSet.selectedIndex	405
DataSet.setIterator()	405
DataSet.setRange()	406
DataSet.skip()	407
DataSet.useSort()	408

**Chapter 14: DateChooser component
(Flash Professional only) 411**

Using the DateChooser component (Flash Professional only)	411
Customizing the DateChooser component (Flash Professional only)	413
DateChooser class (Flash Professional only)	417
DateChooser.change	421
DateChooser.dayNames	423
DateChooser.disabledDays	423
DateChooser.disabledRanges	424
DateChooser.displayedMonth	425
DateChooser.displayedYear	425
DateChooser.firstDayOfWeek	426
DateChooser.monthNames	427
DateChooser.scroll	427
DateChooser.selectableRange	429
DateChooser.selectedDate	430
DateChooser.showToday	431

**Chapter 15: DateField component
(Flash Professional only) 433**

Using the DateField component (Flash Professional only)	433
Customizing the DateField component (Flash Professional only)	435
DateField class (Flash Professional only)	439
DateField.change	444
DateField.close()	445
DateField.close	446
DateField.dateFormatter	448
DateField.dayNames	448
DateField.disabledDays	449
DateField.disabledRanges	449
DateField.displayedMonth	450
DateField.displayedYear	451

DateField.firstDayOfWeek.....	452
DateField.monthNames	452
DateField.open().....	453
DateField.open	454
DateField.pullDown	455
DateField.scroll.....	456
DateField.selectableRange	458
DateField.selectedDate	459
DateField.showToday.....	460
Chapter 16: Delegate class	461
Delegate.create().....	462
Chapter 17: DeltaItem class (Flash Professional only)	463
DeltaItem.argList	464
DeltaItem.curValue.....	464
DeltaItem.delta	465
DeltaItem.kind	465
DeltaItem.message	466
DeltaItem.name.....	466
DeltaItem.newValue.....	467
DeltaItem.oldValue.....	467
Chapter 18: Delta interface (Flash Professional only)	469
Delta.addDeltaItem().....	470
Delta.getChangeList().....	470
Delta.getDeltaPacket().....	471
Delta.getId().....	472
Delta.getItemByName()	473
Delta.getMessage()	474
Delta.getOperation()	475
Delta.getSource().....	476
Chapter 19: DeltaPacket interface (Flash Professional only) .	479
DeltaPacket.getConfigInfo()	480
DeltaPacket.getIterator()	481
DeltaPacket.getSource()	482
DeltaPacket.getTimestamp().....	483
DeltaPacket.getTransactionId()	484
DeltaPacket.logChanges().....	485

Chapter 20: DepthManager class	487
DepthManager.createChildAtDepth()	489
DepthManager.createClassChildAtDepth()	490
DepthManager.createClassObjectAtDepth()	491
DepthManager.createObjectAtDepth()	492
DepthManager.kBottom	493
DepthManager.kCursor	493
DepthManager.kNotopmost	493
DepthManager.kTooltip	494
DepthManager.kTop	494
DepthManager.kTopmost	495
DepthManager.setDepthAbove()	495
DepthManager.setDepthBelow()	496
DepthManager.setDepthTo()	496
Chapter 21: EventDispatcher class	499
Event objects	499
EventDispatcher class (API)	500
EventDispatcher.addEventListener()	501
EventDispatcher.dispatchEvent()	503
EventDispatcher.removeEventListener()	504
Chapter 22: FLVPlayback Component (Flash Professional Only)	505
Using the FLVPlayback component	507
Using cue points	513
Playing multiple FLV files	521
Streaming FLV files from a FCS	524
Customizing the FLVPlayback component	524
FLVPlayback class	539
VideoError class	698
VideoPlayer class	706
Using a SMIL file	712
Chapter 23: FocusManager class	721
Using Focus Manager	722
Customizing Focus Manager	725
FocusManager class (API)	725
FocusManager.defaultPushButton	729
FocusManager.defaultPushButtonEnabled	730
FocusManager.enabled	730

FocusManager.setFocus()	731
FocusManager.nextTabIndex	732
FocusManager.sendDefaultPushButtonEvent()	732
FocusManager.setFocus()	734
Chapter 24: Form class (Flash Professional only)	735
Using the Form class (Flash Professional only)	735
Form class (Flash Professional only)	736
Form.currentFocusedForm	742
Form.getChildForm()	743
Form.indexInParentForm	744
Form.numChildForms	744
Form.parentIsForm	745
Form.parentForm	746
Form.rootForm	746
Form.visible	747
Chapter 25: Iterator interface (Flash Professional only)	749
Iterator.hasNext()	749
Iterator.next()	750
Chapter 26: Label component	751
Using the Label component	751
Customizing the Label component	753
Label class	755
Label.autoSize	758
Label.html	758
Label.text	759
Chapter 27: List component	761
Using the List component	762
Customizing the List component	766
List class	770
List.addItem()	776
List.addItemAt()	777
List.cellRenderer	778
List.change	778
List.dataProvider	780
List.getItemAt()	781
List.hPosition	782
List.hScrollPolicy	783

List.iconField	784
List.iconFunction	785
List.itemRollOut	786
List.itemRollOver	788
List.labelField	789
List.labelFunction	790
List.length	791
List.maxHPosition	791
List.multipleSelection	792
List.removeAll()	793
List.removeItemAt()	794
List.replaceItemAt()	795
List.rowCount	797
List.rowHeight	798
List.scroll	799
List.selectable	800
List.selectedIndex	801
List.selectedIndices	802
List.selectedItem	803
List.selectedItems	804
List.setPropertiesAt()	805
List.sortItems()	806
List.sortItemsBy()	807
List.vPosition	809
List.vScrollPolicy	810
Chapter 28: Loader component	813
Using the Loader component	813
Customizing the Loader component	816
Loader class	817
Loader.autoLoad	821
Loader.bytesLoaded	821
Loader.bytesTotal	822
Loader.complete	823
Loader.content	825
Loader.contentPath	826
Loader.load()	826
Loader.percentLoaded	828
Loader.progress	828
Loader.scaleContent	830

Chapter 29: Media components (Flash Professional only) 831

Interacting with media components (Flash Professional only)	832
Understanding media components (Flash Professional only)	833
Using media components (Flash Professional only)	836
Media component parameters (Flash Professional only)	843
Creating applications with media components (Flash Professional only)	846
Customizing media components (Flash Professional only)	847
Media class (Flash Professional only)	847
Media.activePlayControl	851
Media.addCuePoint()	852
Media.aspectRatio	853
Media.associateController()	853
Media.associateDisplay()	854
Media.autoPlay	855
Media.autoSize	856
Media.backgroundColor	857
Media.bytesLoaded	858
Media.bytesTotal	858
Media.change	859
Media.click	860
Media.complete	861
Media.contentPath	862
Media.controllerPolicy	863
Media.controlPlacement	864
Media.cuePoint	864
Media.cuePoints	865
Media.displayFull()	866
Media.displayNormal()	867
Media.getCuePoint()	868
Media.horizontal	868
Media.mediaType	869
Media.pause()	870
Media.play()	870
Media.playheadChange	871
Media.playheadTime	872
Media.playing	873
Media.preferredHeight	873
Media.preferredWidth	874
Media.progress	875
Media.scrubbing	876
Media.removeAllCuePoints()	877

Media.removeCuePoint()	878
Media.setMedia()	879
Media.stop()	880
Media.totalTime	880
Media.volume	881
Media.volume	882
Chapter 30: Menu component (Flash Professional only)	883
Interacting with the Menu component (Flash Professional only)	884
Using the Menu component (Flash Professional only)	885
About menu item types (Flash Professional only)	888
About initialization object properties (Flash Professional only)	891
Menu parameters (Flash Professional only)	892
Creating an application with the Menu component (Flash Professional only)	892
Customizing the Menu component (Flash Professional only)	897
Menu class (Flash Professional only)	901
Menu.addItem()	905
Menu.addItemAt()	907
Menu.change	908
Menu.createMenu()	910
Menu.dataProvider	911
Menu.getItemAt()	913
Menu.hide()	914
Menu.indexOf()	915
Menu.menuHide	917
Menu.menuShow	919
Menu.removeAll()	921
Menu.removeItem()	922
Menu.removeItemAt()	924
Menu.rollOut	925
Menu.rollOver	927
Menu.setItemEnabled()	929
Menu.setItemSelected()	930
Menu.show()	932
MenuDataProvider class	933
MenuDataProvider.addItem()	934
MenuDataProvider.addItemAt()	936
MenuDataProvider.getItemAt()	938
MenuDataProvider.indexOf()	939
MenuDataProvider.removeItem()	941
MenuDataProvider.removeItemAt()	943

Chapter 31: MenuBar component (Flash Professional only)	945
Interacting with the MenuBar component (Flash Professional only)	946
Using the MenuBar component (Flash Professional only)	946
Customizing the MenuBar component (Flash Professional only)	948
MenuBar class (Flash Professional only)	951
MenuBar.addMenu()	955
MenuBar.addMenuAt()	957
MenuBar.dataProvider	958
MenuBar.getMenuAt()	960
MenuBar.getMenuEnabledAt()	961
MenuBar.labelField	962
MenuBar.labelFunction	963
MenuBar.removeAll()	964
MenuBar.removeMenuAt()	965
MenuBar.setMenuEnabledAt()	966
Chapter 32: NumericStepper component	969
Using the NumericStepper component	970
Customizing the NumericStepper component	972
NumericStepper class	975
NumericStepper.change	980
NumericStepper.maximum	981
NumericStepper.minimum	982
NumericStepper.nextValue	983
NumericStepper.previousValue	984
NumericStepper.stepSize	985
NumericStepper.value	986
Chapter 33: PopUpManager class	987
PopUpManager.createPopUp()	988
PopUpManager.deletePopUp()	989
Chapter 34: ProgressBar component	991
Using the ProgressBar component	991
Customizing the ProgressBar component	996
ProgressBar class	999
ProgressBar.complete	1003
ProgressBar.conversion	1005
ProgressBar.direction	1006
ProgressBar.indeterminate	1007

ProgressBar.label	1008
ProgressBar.labelPlacement	1009
ProgressBar.maximum	1010
ProgressBar.minimum	1012
ProgressBar.mode	1013
ProgressBar.percentComplete	1014
ProgressBar.progress	1016
ProgressBar.setProgress()	1018
ProgressBar.source	1020
ProgressBar.value	1021
Chapter 35: RadioButton component	1023
Using the RadioButton component	1024
Customizing the RadioButton component	1025
RadioButton class	1029
RadioButton.click	1035
RadioButton.data	1037
RadioButton.groupName	1038
RadioButton.label	1039
RadioButton.labelPlacement	1040
RadioButton.selected	1041
RadioButton.selectedData	1042
RadioButton.selection	1043
Chapter 36: RadioButtonGroup component	1045
Chapter 37: RDBMSResolver component (Flash Professional only)	1047
Using the RDBMSResolver component (Flash Professional only)	1048
RDBMSResolver class (Flash Professional only)	1051
RDBMSResolver.addFieldInfo()	1053
RDBMSResolver.beforeApplyUpdates	1054
RDBMSResolver.deltaPacket	1055
RDBMSResolver.fieldInfo	1055
RDBMSResolver.nullValue	1056
RDBMSResolver.reconcileResults	1057
RDBMSResolver.reconcileUpdates	1058
RDBMSResolver.tableName	1059
RDBMSResolver.updateMode	1060
RDBMSResolver.updatePacket	1061
RDBMSResolver.updateResults	1062

Chapter 38: RectBorder class	1063
Using styles with the RectBorder class	1064
Creating a custom RectBorder implementation	1067
Chapter 39: Screen class (Flash Professional only)	1071
Loading external content into screens (Flash Professional only) .	1072
Screen class (API) (Flash Professional only)	1074
Screen.allTransitionsInDone	1080
Screen.allTransitionsOutDone	1081
Screen.currentFocusedScreen	1081
Screen.getChildScreen()	1082
Screen.indexInParent	1083
Screen.mouseDown	1084
Screen.mouseDownSomewhere	1085
Screen.mouseMove	1085
Screen.mouseOut	1086
Screen.mouseOver	1087
Screen.mouseUp	1088
Screen.mouseUpSomewhere	1088
Screen.numChildScreens	1089
Screen.parentIsScreen	1090
Screen.parentScreen	1091
Screen.rootScreen	1091
Chapter 40: ScrollPane component	1093
Using the ScrollPane component	1094
Customizing the ScrollPane component	1096
ScrollPane class	1098
ScrollPane.complete	1103
ScrollPane.content	1104
ScrollPane.contentPath	1106
ScrollPane.getBytesLoaded()	1107
ScrollPane.getBytesTotal()	1108
ScrollPane.hLineScrollSize	1109
ScrollPane.hPageScrollSize	1110
ScrollPane.hPosition	1111
ScrollPane.hScrollPolicy	1112
ScrollPane.progress	1113
ScrollPane.refreshPane()	1115
ScrollPane.scroll	1116
ScrollPane.scrollDrag	1118

ScrollPane.vLineScrollSize	1119
ScrollPane.vPageScrollSize	1120
ScrollPane.vPosition	1121
ScrollPane.vScrollPolicy	1122
Chapter 41: SimpleButton class	1125
SimpleButton.click	1129
SimpleButton.emphasized	1131
SimpleButton.emphasizedStyleDeclaration	1132
SimpleButton.selected	1132
SimpleButton.toggle	1133
Chapter 42: Slide class (Flash Professional only)	1135
Using the Slide class (Flash Professional only)	1136
Slide class (API) (Flash Professional only)	1138
Slide.autoKeyNav	1145
Slide.currentChildSlide	1146
Slide.currentFocusedSlide	1147
Slide.currentSlide	1147
Slide.defaultKeydownHandler	1148
Slide.firstSlide	1149
Slide.getChildSlide()	1150
Slide.gotoFirstSlide()	1151
Slide.gotoLastSlide()	1152
Slide.gotoNextSlide()	1154
Slide.gotoPreviousSlide()	1156
Slide.gotoSlide()	1158
Slide.hideChild	1159
Slide.indexInParentSlide	1160
Slide.lastSlide	1161
Slide.nextSlide	1163
Slide.numChildSlides	1164
Slide.overlayChildren	1165
Slide.parentIsSlide	1166
Slide.parentSlide	1166
Slide.playHidden	1167
Slide.previousSlide	1167
Slide.revealChild	1168
Slide.rootSlide	1169

Chapter 43: StyleManager class	1171
StyleManager.registerColorName()	1172
StyleManager.registerColorStyle()	1173
StyleManager.registerInheritingStyle()	1174
Chapter 44: SystemManager class	1175
SystemManager.screen	1176
Chapter 45: TextArea component	1177
Using the TextArea component	1178
Customizing the TextArea component	1180
TextArea class	1182
TextArea.change	1187
TextArea.editable	1189
TextArea.hPosition	1190
TextArea.hScrollPolicy	1191
TextArea.html	1192
TextArea.length	1193
TextArea.maxChars	1194
TextArea.maxHPosition	1195
TextArea.maxVPosition	1196
TextArea.password	1198
TextArea.restrict	1199
TextArea.scroll	1200
TextArea.styleSheet	1202
TextArea.text	1204
TextArea.vPosition	1205
TextArea.vScrollPolicy	1206
TextArea.wordWrap	1207
Chapter 46: TextInput component	1209
Using the TextInput component	1210
Customizing the TextInput component	1212
TextInput class	1214
TextInput.change	1219
TextInput.editable	1221
TextInput.enter	1222
TextInput.hPosition	1224
TextInput.length	1225
TextInput.maxChars	1226
TextInput.maxHPosition	1227

TextInput.password.....	1228
TextInput.restrict.....	1229
TextInput.text.....	1231
Chapter 47: TransferObject interface.....	1233
TransferObject.clone().....	1234
TransferObject.getPropertyData().....	1235
TransferObject.setPropertyData().....	1236
Chapter 48: TransitionManager class.....	1237
Using the TransitionManager class.....	1237
TransitionManager class summary.....	1239
TransitionManager.allTransitionsInDone.....	1240
TransitionManager.allTransitionsOutDone.....	1241
TransitionManager.content.....	1243
TransitionManager.contentAppearance.....	1243
TransitionManager.start().....	1244
TransitionManager.startTransition().....	1246
TransitionManager.toString().....	1248
Transition-based classes.....	1249
Chapter 49: TreeDataProvider interface (Flash Professional only).....	1257
TreeDataProvider.addNode().....	1258
TreeDataProvider.addNodeAt().....	1259
TreeDataProvider.attributes.data.....	1260
TreeDataProvider.attributes.label.....	1261
TreeDataProvider.getTreeNodeAt().....	1261
TreeDataProvider.removeTreeNode().....	1262
TreeDataProvider.removeTreeNodeAt().....	1263
Chapter 50: Tree component (Flash Professional only).....	1265
Using the Tree component (Flash Professional only).....	1266
Customizing the Tree component (Flash Professional only).....	1273
Tree class (Flash Professional only).....	1278
Tree.addNode().....	1285
Tree.addNodeAt().....	1286
Tree.dataProvider.....	1288
Tree.firstVisibleNode.....	1290

Tree.getDisplayIndex()	1291
Tree.getIsBranch()	1293
Tree.getIsOpen()	1294
Tree.getNodeDisplayedAt()	1295
Tree.getTreeNodeAt()	1296
Tree.nodeClose	1297
Tree.nodeOpen	1299
Tree.refresh()	1300
Tree.removeAll()	1302
Tree.removeTreeNodeAt()	1303
Tree.selectedNode	1304
Tree.selectedNodes	1305
Tree.setIcon()	1306
Tree.setIsBranch()	1308
Tree.setIsOpen()	1309
Chapter 51: Tween class	1311
Using the Tween class	1313
Applying easing methods to components	1315
Tween.continueTo()	1319
Tween.duration	1320
Tween.ffmpeg()	1320
Tween.finish	1321
Tween.FPS	1322
Tween.nextFrame()	1323
Tween.onMotionChanged	1324
Tween.onMotionFinished	1325
Tween.onMotionResumed	1326
Tween.onMotionStarted	1327
Tween.onMotionStopped	1328
Tween.position	1329
Tween.prevFrame()	1330
Tween.resume()	1331
Tween.rewind()	1333
Tween.start()	1334
Tween.stop()	1335
Tween.time	1336
Tween.toString()	1337
Tween.yoyo()	1338

Chapter 52: UIComponent class	1339
UIComponent class (API)	1339
UIComponent.enabled	1343
UIComponent.focusIn	1343
UIComponent.focusOut	1345
UIComponent.getFocus()	1346
UIComponent.keyDown	1347
UIComponent.keyUp	1348
UIComponent.setFocus()	1349
UIComponent.tabIndex	1350
Chapter 53: UIEventDispatcher class	1351
UIEventDispatcher.keyDown	1352
UIEventDispatcher.keyUp	1353
UIEventDispatcher.load	1353
UIEventDispatcher.mouseDown	1354
UIEventDispatcher.mouseOut	1354
UIEventDispatcher.mouseOver	1355
UIEventDispatcher.mouseUp	1356
UIEventDispatcher.removeEventListener()	1356
UIEventDispatcher.unload	1357
Chapter 54: UIObject class	1359
UIObject.bottom	1362
UIObject.createClassObject()	1362
UIObject.createLabel()	1363
UIObject.createObject()	1365
UIObject.destroyObject()	1365
UIObject.doLater()	1366
UIObject.draw	1368
UIObject.getStyle()	1369
UIObject.height	1370
UIObject.hide	1370
UIObject.invalidate()	1371
UIObject.left	1372
UIObject.load	1372
UIObject.move	1373
UIObject.move()	1375
UIObject.redraw()	1376
UIObject.resize	1376
UIObject.reveal	1378

UIObject.right	1379
UIObject.scaleX	1379
UIObject.scaleY	1380
UIObject.setSize()	1381
UIObject.setSkin()	1382
UIObject.setStyle()	1383
UIObject.top	1385
UIObject.unload	1385
UIObject.visible	1386
UIObject.width	1387
UIObject.x	1387
UIObject.y	1388

Chapter 55: UIScrollBar Component 1389

Using the UIScrollBar component	1389
Customizing the UIScrollBar component	1393
UIScrollBar class	1395
UIScrollBar.horizontal	1400
UIScrollBar.lineScrollSize	1401
UIScrollBar.pageScrollSize	1402
UIScrollBar.scroll	1403
UIScrollBar.scrollPosition	1406
UIScrollBar.setScrollProperties()	1408
UIScrollBar.setScrollTarget()	1409
UIScrollBar._targetInstanceName	1410

Chapter 56: Web service classes (Flash Professional only) . . 1413

Making web service classes available at runtime (Flash Professional only)	1414
Log class (Flash Professional only)	1414
Constructor for the Log class	1416
Log.getDateString()	1417
Log.logInfo()	1418
Log.logDebug()	1419
Log.level	1420
Log.name	1421
Log.onLog()	1422
PendingCall class (Flash Professional only)	1423
PendingCall.getOutputParameter()	1425
PendingCall.getOutputParameterByName()	1426
PendingCall.getOutputParameters()	1427
PendingCall.getOutputValue()	1428

PendingCall.getOutputValues()	1429
PendingCall.myCall	1430
PendingCall.onFault	1430
PendingCall.onResult	1432
PendingCall.request	1433
PendingCall.response	1433
SOAPCall class (Flash Professional only)	1434
SOAPCall.concurrency	1435
SOAPCall.doDecoding	1436
SOAPCall.doLazyDecoding	1436
WebService class (Flash Professional only)	1437
Supported types (Flash Professional only)	1438
WebService security (Flash Professional only)	1441
Constructor for the WebService class	1441
WebService.getCall()	1443
WebService.myMethodName()	1444
WebService.onFault	1445
WebService.onLoad	1447

Chapter 57: WebServiceConnector component (Flash Professional only) .1449

Using the WebServiceConnector component (Flash Professional only)	1449
WebServiceConnector class (Flash Professional only)	1451
WebServiceConnector.multiple	
SimultaneousAllowed	1453
WebServiceConnector.operation	1454
WebServiceConnector.params	1455
WebServiceConnector.result	1456
WebServiceConnector.results	1457
WebServiceConnector.send	1457
WebServiceConnector.status	1458
WebServiceConnector.suppress	
InvalidCalls	1461
WebServiceConnector.trigger()	1463
WebServiceConnector.WSDLURL	1464

Chapter 58: Window component .1465

Using the Window component	1465
Customizing the Window component	1468
Window class	1472
Window.click	1476

Window.closeButton	1478
Window.complete	1479
Window.content	1481
Window.contentPath	1482
Window.deletePopUp()	1483
Window.mouseDownOutside	1484
Window.title	1486
Window.titleStyleDeclaration	1487

Chapter 59: XMLConnector component (Flash Professional only) 1489

Using the XMLConnector component (Flash Professional only)	1489
XMLConnector class (Flash Professional only)	1491
XMLConnector.direction	1493
XMLConnector.ignoreWhite	1493
XMLConnector.multipleSimultaneousAllowed	1494
XMLConnector.params	1495
XMLConnector.result	1496
XMLConnector.results	1497
XMLConnector.send	1497
XMLConnector.status	1498
XMLConnector.suppressInvalidCalls	1500
XMLConnector.trigger()	1502
XMLConnector.URL	1503

Chapter 60: XPathAPI class 1505

Chapter 61: XUpdateResolver component (Flash Professional only) 1507

Using the XUpdateResolver component (Flash Professional only)	1508
XUpdateResolver class (Flash Professional only)	1511
XUpdateResolver.beforeApplyUpdates	1512
XUpdateResolver.deltaPacket	1513
XUpdateResolver.includeDeltaPacketInfo	1514
XUpdateResolver.reconcileResults	1514
XUpdateResolver.updateResults	1515
XUpdateResolver.xupdatePacket	1516

Index 1517

The *Components Language Reference* book describes each component and its application programming interface (API). To learn how to use, customize, and create components, see *Using Components*. In *Components Language Reference*, each component description contains information about the following:

- Keyboard interaction
- Live preview
- Accessibility
- Setting the component parameters
- Using the component in an application
- Customizing the component with styles and skins
- ActionScript methods, properties, and events

Components are presented alphabetically. You can also find components arranged by category in the tables that follow.

This chapter contains the following sections:

Types of components	30
Other listings in this chapter	33

Types of components

The following tables list the different components, arranged by category, in version 2 of the Macromedia Component Architecture.

User interface (UI) components

Component	Description
Accordion component (Flash Professional only)	A set of vertical overlapping views with buttons along the top that allow users to switch views.
Alert component (Flash Professional only)	A window that presents a message and buttons to capture the user's response.
Button component	A resizable button that can be customized with a custom icon.
CheckBox component	Allows users to make a Boolean (true or false) choice.
ComboBox component	Allows users to select one option from a scrolling list of choices. This component can have an selectable text field at the top of the list that allows users to search the list.
DataGrid component (Flash Professional only)	Allows users to display and manipulate multiple columns of data.
DateChooser component (Flash Professional only)	Allows users to select one or more dates from a calendar.
DateField component (Flash Professional only)	An nonselectable text field with a calendar icon. When a user clicks inside the component's bounding box, Macromedia Flash displays a DateChooser component.
Label component	A non-editable, single-line text field.
List component	Allows users to select one or more options from a scrolling list.
Loader component	A container that holds a loaded SWF or JPEG file.
Menu component (Flash Professional only)	A standard desktop application menu; allows users to select one command from a list.
MenuBar component (Flash Professional only)	A horizontal bar of menus.
NumericStepper component	A text box with clickable arrows that raise and lower the value of a number.
ProgressBar component	Displays the progress of a process, such as a loading operation.
RadioButton component	Allows users to select between mutually exclusive options.

Component	Description
ScrollPane component	Displays movie clips, bitmaps, and SWF files in a limited area using automatic scroll bars.
TextArea component	An optionally editable, multiline text field.
TextInput component	An optionally editable, single-line text input field.
Tree component (Flash Professional only)	Allows a user to manipulate hierarchical information.
Window component	A draggable window with a title bar, caption, border, and Close button and content-display area.
UIScrollBar Component	Allows you to add a scroll bar to a text field.

Data handling

Component	Description
Data binding classes (Flash Professional only)	Classes that implement the Flash runtime data binding functionality.
DataHolder component (Flash Professional only)	Holds data and can be used as a connector between components.
DataProvider API	The model for linear-access lists of data; it provides simple array-manipulation capabilities that broadcast data changes.
DataSet component (Flash Professional only)	A building block for creating data-driven applications.
RDBMSResolver component (Flash Professional only)	Lets you save data back to any supported data source. This component translates the XML that can be received and parsed by a web service, JavaBean, servlet, or ASP page.
Web service classes (Flash Professional only)	Classes that allow access to web services that use Simple Object Access Protocol (SOAP). These classes are in the mx.services package.
WebServiceConnector component (Flash Professional only)	Provides scriptless access to web service method calls.
XMLConnector component (Flash Professional only)	Reads and writes XML documents by using the HTTP GET and POST methods.
XUpdateResolver component (Flash Professional only)	Lets you save data back to any supported data source. This component translates the delta packet into XUpdate.

Media components

Component	Description
FLVPlayback Component (Flash Professional Only)	Lets you include a video player in your Flash application to play progressive streaming video over HTTP, from a Flash Video Streaming Service (FVSS), or from Flash Communication Server (FCS).
MediaController component (Flash Professional only)	Controls streaming media playback in an application (see “Media components (Flash Professional only)” on page 831).
MediaDisplay component (Flash Professional only)	Displays streaming media in an application (see “Media components (Flash Professional only)” on page 831).
MediaPlayback component (Flash Professional only)	A combination of the MediaDisplay and MediaController components (see “Media components (Flash Professional only)” on page 831).

Managers

Class	Description
DepthManager class	Manages the stacking depths of objects.
FocusManager class	Handles Tab key navigation between components. Also handles focus changes as users click in the application.
PopUpManager class	Lets you create and delete pop-up windows.
StyleManager class	Lets you register styles and manages inherited styles.
SystemManager class	Lets you manage which top-level window is activated.
TransitionManager class	Lets you manage animation effects to slides and movie clips.

Screens

Class	Description
Form class (Flash Professional only)	Lets you manipulate form application screens at runtime.
Screen class (Flash Professional only)	Base class for the Slide and Form classes.
Slide class (Flash Professional only)	Lets you manipulate slide presentation screens at runtime.

Other listings in this chapter

This book also describes several classes and APIs that are not included in the categories of components listed in the previous section. These classes and APIs are listed in the following table.

Item	Description
CellRenderer API	A set of properties and methods that the list-based components (List, DataGrid, Tree, Menu, and ComboBox) use to manipulate and display custom cell content for each of their rows.
Collection interface (Flash Professional only)	Lets you manage a group of related items, called collection items. Each collection item in this set has properties that are described in the metadata of the collection item class definition.
DataGridColumn class (Flash Professional only)	Lets you create objects to use as columns of a data grid.
Delegate class	Allows a function passed from one object to another to be run in the context of the first object.
Delta interface (Flash Professional only)	Provides access to the transfer object, collection, and transfer object-level changes.
Deltaltem class (Flash Professional only)	Provides information about an individual operation performed on a transfer object.
DeltaPacket interface (Flash Professional only)	Along with the Delta interface and Deltaltem class, lets you manage changes made to data.
EventDispatcher class	Let you add and remove event listeners so that your code can react to events appropriately.
Iterator interface (Flash Professional only)	Lets you step through the objects that a collection contains.
MenuDataProvider class	Lets XML instances assigned to a <code>Menu.dataProvider</code> property use methods and properties to manipulate their own data as well as the associated menu views.
RectBorder class	Describes the styles used to control component borders.

Item	Description
SimpleButton class	Lets you control some aspects of button appearance and behavior.
TransferObject interface	Defines a set of methods that items managed by the DataSet component must implement.
TreeDataProvider interface (Flash Professional only)	A set of properties and methods used to create XML for the <code>Tree.dataProvider</code> property.
Tween class	Lets you use ActionScript to move, resize, and fade movie clips easily on the Stage.
UIComponent class	Provides methods, properties, and events that allow components to share some common behavior.
UIEventDispatcher class	Allows components to emit certain events. This class is mixed in to the UIComponent class.
UIObject class	The base class for all version 2 components.

Accordion component (Flash Professional only)

The Accordion component is a navigator that contains a sequence of children that it displays one at a time. The children must be objects that inherit from the UIObject class (which includes all components and screens built with version 2 of the Macromedia Component Architecture); most often, children are a subclass of the View class. This includes movie clips assigned to the class `mx.core.View`. To maintain tabbing order in an accordion's children, the children must also be instances of the View class.

An accordion creates and manages header buttons that a user can click to navigate between the accordion's children. An accordion has a vertical layout with header buttons that span the width of the component. One header is associated with each child, and each header belongs to the accordion—not to the child. When a user clicks a header, the associated child is displayed below that header. The transition to the new child uses a transition animation.

An accordion with children accepts focus, and changes the appearance of its headers to display focus. When a user tabs into an accordion, the selected header displays the focus indicator. An accordion with no children does not accept focus. Clicking components that can take focus within the selected child gives them focus. When an Accordion instance has focus, you can use the following keys to control it:

Key	Description
Down Arrow, Right Arrow	Moves focus to the next child header. Focus cycles from last to first without changing the selected child.
Up Arrow, Left Arrow	Moves focus to the previous child header. Focus cycles from first to last without changing the selected child.
End	Selects the last child.
Enter/Space	Selects the child associated with the header that has focus.
Home	Selects the first child.
Page Down	Selects the next child. Selection cycles from the last child to the first child.

Key	Description
Page Up	Selects the previous child. Selection cycles from the first child to the last child.
Shift+Tab	Moves focus to the previous component. This component may be inside the selected child, or outside the accordion; it is never another header in the same accordion.
Tab	Moves focus to the next component. This component may be inside the selected child, or outside the accordion; it is never another header in the same accordion.

The Accordion component cannot be made accessible to screen readers.

Using the Accordion component (Flash Professional only)

You can use the Accordion component to present multipart forms. For example, a three-child accordion might present forms where the user fills out her shipping address, billing address, and payment information for an e-commerce transaction. Using an accordion instead of multiple web pages minimizes server traffic and allows the user to maintain a better sense of progress and context in an application.

Accordion parameters

You can set the following authoring parameters for each Accordion component instance in the Property inspector or the Component inspector (Window > Component Inspector menu option):

childIcons is an array that specifies the linkage identifiers of the library symbols to be used as the icons on the accordion's headers. The default value is [] (an empty array).

childLabels is an array that specifies the text labels to use on the accordion's headers. The default value is [] (an empty array).

childNames is an array that specifies the instance names of the accordion's children. The values that you enter are the instance names for the child symbols that you specify in the childSymbols parameter. The default value is [] (an empty array).

childSymbols is an array that specifies the linkage identifiers of the library symbols to be used to create the accordion's children. The default value is [] (an empty array).

You can set the following additional parameters for each Accordion component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector will have no visible effect.

You can write ActionScript to control additional options for the Accordion component by using its properties, methods, and events. For more information, see [“Accordion class \(Flash Professional only\)” on page 47](#).

Creating an application with the Accordion component

In this example, an application developer is building the checkout section of an online store. The design calls for an accordion with three forms in which a user enters a shipping address, a billing address, and payment information. The shipping address and billing address forms are identical.

To use screens to add an Accordion component to an application:

1. In Flash, select **File > New** and select **Flash Form Application**.
2. Double-click the text `Form1`, and enter the name **addressForm**.
Although it doesn't appear in the library, the `addressForm` screen is a symbol of the `Screen` class. Because the `Screen` class is a subclass of the `View` class, an accordion can use it as a child.
3. With the form selected, in the Property inspector, set the form's `visible` property to `false`.
This hides the contents of the form in the application; the form only appears in the accordion.
4. Drag components such as `Label` and `TextInput` from the Components panel onto the form to create a mock address form; arrange them, and set their properties in the Parameters tab of the Component inspector.
Position the form elements in the upper left corner of the form. This corner of the form is placed in the upper-left corner of the accordion.
5. Repeat steps 2-4 to create a screen named **checkoutForm**.

6. Create a new screen named **accordionForm**.
7. Drag an Accordion component from the Components panel to the accordionForm form, and name it **my_acc**.
8. With **my_acc** selected, in the Property inspector, do the following:
 - For the **childSymbols** property, enter **addressForm**, **addressForm**, and **checkoutForm**. These strings specify the names of the screens used to create the accordion's children.

NOTE

The first two children are instances of the same screen, because the shipping address form and the billing address form are identical.

- For the **childNames** property, enter **shippingAddress**, **billingAddress**, and **checkout**. These strings are the ActionScript names of the accordion's children.
 - For the **childLabels** property, enter **Shipping Address**, **Billing Address**, and **Checkout**. These strings are the text labels on the accordion headers.
9. Select **Control > Test Movie**.

To add an Accordion component to an application:

1. Select **File > New** and create a new Flash document.
2. Select **Insert > New Symbol** and name it **AddressForm**.
3. In the **Create New Symbol** dialog box, click the **Advanced** button and select **Export for ActionScript**. In the **AS 2.0 Class** field, enter **mx.core.View**.

To maintain tabbing order in an accordion's children, the children must also be instances of the **View** class.
4. Drag components such as **Label** and **TextInput** from the Components panel onto the Stage to create a mock address form; arrange them, and set their properties in the **Parameters** tab of the Component inspector.

Position the form elements in relation to 0,0 (the middle) on the Stage. The 0,0 coordinate of the movie clip is placed in the upper left corner of the accordion.
5. Select **Edit > Edit Document** to return to the main timeline.
6. Repeat steps 2-5 to create a movie clip named **CheckoutForm**.
7. Drag an Accordion component from the Components panel to add it to the Stage on the main timeline.

8. In the Property inspector, do the following:

- Enter the instance name **my_acc**.
- For the `childSymbols` property, enter **AddressForm**, **AddressForm**, and **CheckoutForm**.

These strings specify the names of the movie clips used to create the accordion's children.

NOTE

The first two children are instances of the same movie clip, because the shipping address form and the billing address form are identical.

- For the `childNames` property, enter **shippingAddress**, **billingAddress**, and **checkout**. These strings are the ActionScript names of the accordion's children.
- For the `childLabels` property, enter **Shipping Address**, **Billing Address**, and **Checkout**.

These strings are the text labels on the accordion headers.

- For the `childIcons` property, enter **AddressIcon**, **AddressIcon**, and **CheckoutIcon**. These strings specify the linkage identifiers of the movie clip symbols that are used as the icons on the accordion headers. You must create these movie clip symbols if you want icons in the headers.

9. Select Control > Test Movie.

To use ActionScript to add children to an Accordion component:

1. Select File > New and create a Flash document.
2. Drag an Accordion component from the Components panel to the Stage.
3. In the Property inspector, enter the instance name **my_acc**.
4. Drag a TextInput component to the library.

This adds the component to the library so that you can dynamically instantiate it in step 6.

5. In the Actions panel in Frame 1 of the timeline, enter the following (this code calls the `createChild()` method to create its child views.):

```
import mx.core.View;

// Create child panels for each form to be displayed in my_acc object.
my_acc.createChild(View, "shippingAddress", {label: "Shipping
Address"});
my_acc.createChild(View, "billingAddress", {label: "Billing Address"});
my_acc.createChild(View, "payment", {label: "Payment"});
```

6. In the Actions panel in Frame 1, below the code you entered in step 5, enter the following code (this code adds two `TextInput` component instances to the accordion's children):

```
// Create child text input for the shippingAddress panel.
var firstNameChild_obj:Object =
    my_acc.shippingAddress.createChild("TextInput", "firstName", {text:
    "First Name"});

// Set position of text input.
firstNameChild_obj.move(10, 38);
firstNameChild_obj.setSize(110, 20);

// Create another child text input.
var lastNameChild_obj:Object =
    my_acc.shippingAddress.createChild("TextInput", "lastName", {text:
    "Last Name"});

// Set position of text input.
lastNameChild_obj.move(150, 38);
lastNameChild_obj.setSize(140, 20);
```

Customizing the Accordion component (Flash Professional only)

You can transform an Accordion component horizontally and vertically during authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

The `setSize()` method and the Transform tool change only the width of the accordion's headers and the width and height of its content area. The height of the headers and the width and height of the children are not affected. Calling the `setSize()` method is the only way to change the bounding rectangle of an accordion.

If the headers are too small to contain their label text, the labels are clipped. If the content area of an accordion is smaller than a child, the child is clipped.

Using styles with the Accordion component

You can set style properties to change the appearance of the border and background of an Accordion component.

An Accordion component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. This is the only color style that doesn't inherit its value. Possible values are "haloGreen", "haloBlue", and "haloOrange".
<code>backgroundColor</code>	Both	The background color. The default color is white.
<code>borderStyle</code>	Both	The Accordion component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. For more information, see "RectBorder class" on page 1063 . The Accordion component's default border style value is "solid".
<code>headerHeight</code>	Both	The height of the header buttons, in pixels. The default value is 22.
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for the header labels. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font of the header labels. The default value is 10.
<code>fontStyle</code>	Both	The font style for the header labels; either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight for the header labels; either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".

Style	Theme	Description
<code>textDecoration</code>	Both	The text decoration; either "none" or "underline".
<code>openDuration</code>	Both	The duration, in milliseconds, of the transition animation.
<code>openEasing</code>	Both	A reference to a tweening function that controls the animation. Defaults to sine in/out. For more information, see "Customizing component animations" in <i>Using Components</i> .

So, for example, the following code sets the style appearance of the font within an accordion instance named `my_acc` to italic:

```
my_acc.setStyle("fontStyle", "italic");
```

If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties. For more information, see "Using styles to customize component color and text" in *Using Components*.

Using skins with the Accordion component

The Accordion component uses skins to represent the visual states of its header buttons. To skin the buttons and title bar while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/Accordion Assets skins states folder in the library of one of the themes' FLA files. For more information, see "About skinning components" in *Using Components*.

An Accordion component is composed of its border, background, header buttons, and children. The border and background are provided by the `RectBorder` class by default. For information on skinning the `RectBorder` class, see "[RectBorder class](#)" on page 1063. You can skin the headers with the skins listed below.

Property	Description	Default value
<code>falseUpSkin</code>	The up (normal) state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseDownSkin</code>	The pressed state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseOverSkin</code>	The rolled-over state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseDisabled</code>	The disabled state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>trueUpSkin</code>	The up (normal) state of the header above the expanded child.	<code>accordionHeaderSkin</code>

Property	Description	Default value
<code>trueDownSkin</code>	The pressed state of the header above the expanded child.	<code>accordionHeaderSkin</code>
<code>trueOverSkin</code>	The rolled-over state of the header above the expanded child.	<code>accordionHeaderSkin</code>
<code>trueDisabledSkin</code>	The disabled state of the header above the expanded child.	<code>accordionHeaderSkin</code>

Using ActionScript to draw the Accordion header

The default headers in both the Halo and Sample themes use the same skin element for all states and draw the actual graphics through ActionScript. The Halo implementation uses an extension of the `RectBorder` class and custom drawing API code to draw the states. The Sample implementation uses the same skin and the same ActionScript class as the Button skin.

To create an ActionScript class to use as the skin and provide different states, the skin can read the `borderStyle` style property of the skin to determine the state. The following table shows the border style that is set for each skin:

Property	Border style
<code>falseUpSkin</code>	<code>falseup</code>
<code>falseDownSkin</code>	<code>falsedown</code>
<code>falseOverSkin</code>	<code>falserollover</code>
<code>falseDisabled</code>	<code>falsedisabled</code>
<code>trueUpSkin</code>	<code>trueup</code>
<code>trueDownSkin</code>	<code>truedown</code>
<code>trueOverSkin</code>	<code>truerollover</code>
<code>trueDisabledSkin</code>	<code>truedisabled</code>

To create an ActionScript-customized Accordion header skin:

1. Create a new ActionScript class file.

For this example, name the file **RedGreenBlueHeader.as**.

2. Copy the following ActionScript to the file:

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueHeader extends RectBorder
{
    static var symbolName_str:String = "RedGreenBlueHeader";
    static var symbolOwner_obj:Object = RedGreenBlueHeader;

    function size():Void
    {
        var color_num:Number; // Color
        var borderStyle_str:String = getStyle("borderStyle"); // Attribute of
        Accordion

        // Define the colors of each tab in the Accordion for each tab state.
        switch (borderStyle_str) {
            case "falseup":
            case "falserollover":
            case "falsedisabled":
                color_num = 0x7777FF;
                break;
            case "falsedown":
                color_num = 0x77FF77;
                break;
            case "trueup":
            case "truedown":
            case "truerollover":
            case "truedisabled":
                color_num = 0xFF7777;
                break;
        }

        // Clear default style and draw custom style.
        clear();
       LineStyle(0, 0, 100);
       beginFill(color_num, 100);
       drawRect(0, 0, __width, __height);
       endFill();
    }

    // required for skins
    static function classConstruct():Boolean
    {
        UIObjectExtensions.Extensions();
    }
}
```

```

    _global.skinRegistry["AccordionHeaderSkin"] = true;
    return true;
}
static var classConstructed_b1:Boolean = classConstruct();
static var UIObjectExtensionsDependency_obj:Object =
    UIObjectExtensions;
}

```

This class creates a square box based on the border style: a blue box for the false up, rollover, and disabled states; a green box for the normal pressed state; and a red box for the expanded child.

3. Save the file.
4. Create a new FLA file and save it in the same folder as the AS file.
5. Create a new symbol by selecting Insert > New Symbol.
6. Set the name to `AccordionHeaderSkin`.
7. If the advanced view is not displayed, click the Advanced button.
8. Select Export for ActionScript.

The identifier is automatically filled out with `AccordionHeaderSkin`.
9. Set the AS 2.0 class to `RedGreenBlueHeader`.
10. Make sure that Export in First Frame is already selected, and click OK.
11. In Scene 1, drag an Accordion component to the Stage.
12. Set the Accordion properties so that they display several children.

For example, set the `childLabels` to an array of `[One,Two,Three]` and `childNames` to an array of `[one,two,three]`.
13. Select Control > Test Movie.

Using movie clips to customize the Accordion header skin

The example above demonstrates how to use an ActionScript class to customize the Accordion header skin, which is the method used by the skins provided in both the Halo and Sample themes. However, because the example uses simple colored boxes, it is simpler in this case to use different movie clip symbols as header skins.

To create movie clip symbols for Accordion header skins:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `RedAccordionHeaderSkin`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.
The identifier is automatically filled out with `RedAccordionHeaderSkin`.
6. Leave the AS 2.0 Class text box blank.
7. Make sure that Export in First Frame is already selected, and click OK.
8. Open the new symbol for editing.
9. Use the drawing tools to create a box with a red fill and black line.
10. Set the border style to hairline.
11. Set the box, including the border, so that it is positioned at (0,0) and has a width and height of 100.
The ActionScript code sizes the skin as needed.
12. Repeat steps 2-11 and create green and blue skins, named accordingly.
13. Click the Back button to return to the main timeline.
14. Drag an Accordion component to the Stage.
15. Set the Accordion properties so that they display several children.
For example, set `childLabels` to an array of `[One,Two,Three]` and `childNames` to an array of `[one,two,three]`.
16. Copy the following ActionScript code to the Actions panel with the Accordion instance selected:

```
onClipEvent(initialize) {
    falseUpSkin = "RedAccordionHeaderSkin";
    falseDownSkin = "GreenAccordionHeaderSkin";
    falseOverSkin = "RedAccordionHeaderSkin";
    falseDisabled = "RedAccordionHeaderSkin";
    trueUpSkin = "BlueAccordionHeaderSkin";
    trueDownSkin = "BlueAccordionHeaderSkin";
    trueOverSkin = "BlueAccordionHeaderSkin";
    trueDisabledSkin = "BlueAccordionHeaderSkin";
}
```
17. Select Control > Test Movie.

Accordion class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > Accordion

ActionScript Class Name mx.containers.Accordion

An Accordion component contains children that are displayed one at a time. Each child has a corresponding header button that is created when the child is created. A child must be an instance of UIObject.

A movie clip symbol automatically becomes an instance of the UIObject class when it becomes a child of an accordion. However, to maintain tabbing order in an accordion's children, the children must also be instances of the View class. If you use a movie clip symbol as a child, set its AS 2.0 Class field to mx.core.View so that it inherits from the View class.

Setting a property of the Accordion class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property that is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.Accordion.version);
```

NOTE

The code `trace(my_accInstance.version);` returns `undefined`.

Method summary for the Accordion class

The following table lists methods of the Accordion class.

Method	Description
Accordion.createChild()	Creates a child for an Accordion instance.
Accordion.createSegment()	Creates a child for an Accordion instance. The parameters for this method are different from those of the <code>createChild()</code> method.
Accordion.destroyChildAt()	Destroys a child at a specified index position.
Accordion.getChildAt()	Gets a reference to a child at a specified index position.
Accordion.getHeaderAt()	Gets a reference to a header object at a specified index position.

Methods inherited from the UIObject class

The following table lists the methods the Accordion class inherits from the UIObject class. When calling these methods from the Accordion object, use the form *accordionInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from UIComponent class

The following table lists the methods the Accordion class inherits from the UIComponent class. When calling these methods from the Accordion object, use the form *accordionInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the Accordion class

The following table lists properties of the Accordion class.

Property	Description
Accordion.numChildren	The number of children of an Accordion instance.
Accordion.selectedChild	A reference to the selected child.
Accordion.selectedIndex	The index position of the selected child.

Properties inherited from the UIObject class

The following table lists the properties the Accordion class inherits from the UIObject class. When accessing these properties, use the form *accordionInstance.propertyName*.

Property	Description
UIObject.bottom	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
UIObject.height	Read-only; the height of the object, in pixels.
UIObject.left	Read-only; the left edge of the object, in pixels.
UIObject.right	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
UIObject.scaleX	A number indicating the scaling factor in the x direction of the object, relative to its parent.
UIObject.scaleY	A number indicating the scaling factor in the y direction of the object, relative to its parent.
UIObject.top	Read-only; the position of the top edge of the object, relative to its parent.
UIObject.visible	A Boolean value indicating whether the object is visible (<i>true</i>) or not (<i>false</i>).
UIObject.width	Read-only; the width of the object, in pixels.
UIObject.x	Read-only; the left edge of the object, in pixels.
UIObject.y	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the Accordion class inherits from the UIComponent class. When accessing these properties, use the form *accordionInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the Accordion class

The following table lists an event of the Accordion class.

Event	Description
<code>Accordion.change</code>	Broadcast to all registered listeners when the <code>selectedIndex</code> and <code>selectedChild</code> properties of an accordion change because of a user's mouse click or keypress.

Events inherited from the UIObject class

The following table lists the events the Accordion class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Accordion class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Accordion.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // Insert your code here.
};
accordionInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    // Insert your code here.
}
```

Description

Event; broadcast to all registered listeners when the `selectedIndex` and `selectedChild` properties of an accordion change. This event is broadcast only when a user's mouse click or keypress changes the value of `selectedChild` or `selectedIndex`—not when the value is changed with ActionScript. This event is broadcast before the transition animation occurs.

Version 2 components use a dispatcher/event listener model. The Accordion component dispatches a `change` event when one of its buttons is clicked and the event is handled by a function (also called a *handler*) on a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it a reference to the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

The Accordion change event also contains two unique event object properties:

- `newValue` Number; the index of the child that is about to be selected.
- `prevValue` Number; the index of the child that was previously selected.

Example

The following example uses an Accordion instance named `my_acc` containing three child panels labelled “Shipping Address”, “Billing Address”, and “Payment”. The code defines a handler called `my_accListener` and passes the handler to the `my_acc.addEventListener()` method as the second parameter. The event object is captured by the `change` handler in the *eventObject* parameter. When the change event is broadcast, a trace statement is sent to the Output panel.

```
// Create new Listener object.
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // Assign label of child panel to variable.
    var selectedChild_str:String = my_acc.selectedChild.label;
    // Perform action based on selected child.
    switch (selectedChild_str) {
        case "Shipping Address":
            trace("One was selected");
            break;
        case "Billing Address":
            trace("Two was selected");
            break;
        case "Payment":
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

Accordion.createChild()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.createChild(classOrSymbolName, instanceName[,  
    initialProperties])
```

Parameters

classOrSymbolName Either the constructor function for the class of the UIObject to be instantiated, or the linkage name (a reference to the symbol to be instantiated). The class must be UIObject or a subclass of UIObject, but most often it is View object or a subclass of View.

instanceName The instance name of the new instance.

initialProperties An optional parameter that specifies initial properties for the new instance. You can use the following properties:

- `label` A string that specifies the text label that the new child instance uses on its header.
- `icon` A string that specifies the linkage identifier of the library symbol that the child uses for the icon on its header.

Returns

A reference to an instance of the UIObject that is the newly created child.

Description

Method (inherited from View); creates a child for the accordion. The newly created child is added to the end of the list of children owned by the accordion. Use this method to place views inside the accordion. The created child is an instance of the class or movie clip symbol specified in the `classOrSymbolName` parameter. You can use the `label` and `icon` properties to specify a text label and an icon for the associated accordion header for each child in the `initialProperties` parameter.

When each child is created, it is assigned an index number in the order of creation and the `numChildren` property is increased by 1.

Example

Start with an Accordion instance on the Stage named `my_acc`. Add a symbol to the library with the Linkage Identifier `payIcon` to be the icon for the child header. The following code creates a child named `billing` (with the label “Payment”) that is an instance of the

View class:

```
var child_obj:Object = my_acc.createChild(mx.core.View, "billing", {label:
    "Payment", icon: "payIcon"});
```

The following code also creates a child that is an instance of the View class, but it uses `import` to reference the constructor for the View class:

```
import mx.core.View;
var child_obj:Object = my_acc.createChild(View, "billing", {label:
    "Payment", icon: "payIcon"});
```

Or, add a movie clip symbol to the library with the Linkage Identifier `PaymentForm` to be the Accordion child, and the following code creates an instance of `PaymentForm` named `billing` as the child of `my_acc` (this approach is useful for dynamic content where you load the dynamic content into a movie clip symbol, and then make that symbol a child of the Accordion instance):

```
var child_obj:Object = my_acc.createChild("PaymentForm", "billing", {label:
    "Payment", icon: "payIcon"});
```

For a more complex example, keep the Accordion instance `my_acc` on the Stage. Then drag a Label component and a TextInput component from the Components panel to the current document’s library (so that you have both a TextInput symbol and a Label symbol in the library). Paste the following code in the first frame of the main timeline (replacing any code from the previous examples). The following code creates a child that is an instance of the View class named `billing`, and also adds children to `billing` to provide labels and text input fields for a form:

```
import mx.core.View;
import mx.controls.Label;
import mx.controls.TextInput;
var child_obj:Object = my_acc.createChild(View, "billing",
    {label:"Payment", icon: "payIcon"});
// Create labels as children of the view instance.
var cardType_label:Object = child_obj.createChild(Label, "CardType_label",
    {_x:10, _y:50});
var cardNumber_label:Object = child_obj.createChild(Label,
    "CardNumber_label", {_x:10, _y:100});
// Create text inputs as children of the view instance.
var cardTypeInput_ti:Object = child_obj.createChild(TextInput,
    "CardType_ti", {_x:150, _y:50});
var cardNumberInput_ti:Object = child_obj.createChild(TextInput,
    "CardNumber_ti", {_x:150, _y:100});
// Fill in labels.
```

```
cardType_label.text = "Card Type";
cardNumber_label.text = "Card Number";
```

Accordion.createSegment()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.createSegment(classOrSymbolName, instanceName[], label[],  
                                icon[])
```

Parameters

classOrSymbolName Either a reference to the constructor function for the class of the UIObject to be instantiated, or the linkage name of the symbol to be instantiated. The class must be UIObject or a subclass of UIObject, but most often it is View or a subclass of View.

instanceName The instance name of the new instance.

label A string that specifies the text label that the new child instance uses on its header. This parameter is optional.

icon A string reference to the linkage identifier of the library symbol that the child uses for the icon on its header. This parameter is optional.

Returns

A reference to the newly created UIObject instance.

Description

Method; creates a child for the accordion. The newly created child is added to the end of the list of children owned by the accordion. Use this method to place views inside the accordion. The created child is an instance of the class or movie clip symbol specified in the *classOrSymbolName* parameter. You can use the *label* and *icon* parameters to specify a text label and an icon for the associated accordion header for each child.

The `createSegment()` method differs from the `addChild()` method in that *label* and *icon* are passed directly as parameters, not as properties of an *initialProperties* parameter. When each child is created, it is assigned an index number in the order of creation, and the `numChildren` property is increased by 1.

Example

Start with an **Accordion** instance on the Stage named `my_acc`. Add a movie clip symbol to the library with the Linkage Identifier `PaymentForm` to be the **Accordion** child. Then, add a symbol to the library with Linkage Identifier `payIcon` to be the icon for the child header. The following example creates an instance of the `PaymentForm` movie clip symbol named `billing` as the last child of `my_acc` with header label “Payment” and the icon in the library:

```
var child_obj:Object = my_acc.createSegment("PaymentForm", "billing",  
    "Payment", "payIcon");
```

The following code creates a child that is an instance of the `View` class:

```
var child_obj:Object = my_acc.createSegment(mx.core.View, "billing",  
    "Payment", "payIcon");
```

The following code also creates a child that is an instance of the `View` class, but it uses `import` to reference the constructor for the `View` class:

```
import mx.core.View;  
var child_obj:Object = my_acc.createSegment(View, "billing", "Payment",  
    "payIcon");
```

Drag a **Label** component and a **TextInput** component from the **Components** panel to the current document’s library (so that you have both a `TextInput` symbol and a `Label` symbol in the library). The following code creates a child that is an instance of the `View` class named `billing`, and also adds children to `billing` to provide labels and text input fields for a form:

```
import mx.core.View;  
import mx.controls.Label;  
import mx.controls.TextInput;  
var child_obj:Object = my_acc.createSegment(View, "billing", "Payment",  
    "payIcon");  
// Create labels as children of the view instance.  
var cardType_label:Object = child_obj.createChild(Label, "CardType_label",  
    {_x:10, _y:50});  
var cardNumber_label:Object = child_obj.createChild(Label,  
    "CardNumber_label", {_x:10, _y:100});  
// Create text inputs as children of the view instance.  
var cardTypeInput_ti:Object = child_obj.createChild(TextInput,  
    "CardType_ti", {_x:150, _y:50});  
var cardNumberInput_ti:Object = child_obj.createChild(TextInput,  
    "CardNumber_ti", {_x:150, _y:100});  
// Fill in labels.  
cardType_label.text = "Card Type";  
cardNumber_label.text = "Card Number";
```


Accordion.destroyChildAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.destroyChildAt(index)
```

Parameters

index The index number of the accordion child to destroy. Each child of an accordion is assigned a zero-based index number in the order in which it was created.

Returns

Nothing.

Description

Method (inherited from View); destroys one of the accordion's children. The child to be destroyed is specified by its index, which is passed to the method in the *index* parameter. Calling this method destroys the corresponding header as well.

If the destroyed child is selected, a new selected child is chosen. If there is a next child, it is selected. If there is no next child, the previous child is selected. If there is no previous child, the selection is undefined.

NOTE

Calling `destroyChildAt()` decreases the `numChildren` property by 1.

Example

The following code destroys the first child of `my_acc` when the third child is selected:

```
import mx.core.View;

// Create child panels with instances of the View class.
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// Create new Listener object.
my_accListener = new Object();
my_accListener.change = function() {
    if ("myMainItem3"){
        my_acc.destroyChildAt(0);
    }
};

my_acc.addEventListener("change", my_accListener);
```

See also

[Accordion.createChild\(\)](#)

Accordion.getChildAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.getChildAt(index)
```

Parameters

index The index number of an accordion child. Each child of an accordion is assigned a zero-based index in the order in which it was created.

Returns

A reference to the instance of the UIObject at the specified index.

Description

Method; returns a reference to the child at the specified index. Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on.

Example

The following code gets a reference to the last child of `my_acc` and changes the label to “Last Child”:

```
import mx.core.View;

// Create child panels with instances of the View class.
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// Get reference for last child object.
var lastChild_obj:Object = my_acc.getChildAt(my_acc.numChildren - 1);
// Change label of object.
lastChild_obj.label = "Last Child";
```

Accordion.getHeaderAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.getHeaderAt(index)
```

Parameters

index The index number of an accordion header. Each header of an accordion is assigned a zero-based index in the order in which it was created.

Returns

A reference to the instance of the UIObject at the specified index.

Description

Method; returns a reference to the header at the specified index. Each accordion header is given an index number for its position. This index number is zero-based, so the first header is 0, the second header is 1, and so on.

Example

The following code gets a reference to the last header of `my_acc` and displays the label in the Output panel:

```
import mx.core.View;

// Create child panels for each form to be displayed in my_acc object.
my_acc.createChild(View, "shippingAddress", {label: "Shipping Address"});
my_acc.createChild(View, "billingAddress", {label: "Billing Address"});
my_acc.createChild(View, "payment", {label: "Payment"});

var head3:Object = my_acc.getHeaderAt(2);
trace(head3.label);
```

Accordion.numChildren

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.numChildren
```

Description

Property (inherited from `View`); indicates the number of children (of type `UIObject`) in an `Accordion` instance. Headers are not counted as children.

Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on. The code `my_acc.numChildren - 1` always refers to the last child added to an accordion. For example, if there were seven children in an accordion, the last child would have the index 6. The `numChildren` property is not zero-based, so the value of `my_acc.numChildren` would be 7. The result of `7 - 1` is 6, which is the index number of the last child.

Example

The following code uses `numChildren` to get a reference to the last child of `my_acc` and changes the label to “Last Child”:

```
import mx.core.View;

// Create child panels with instances of the View class.
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// Get reference for last child object.
var lastChild_obj:Object = my_acc.getChildAt(my_acc.numChildren - 1);
// Change label of object.
lastChild_obj.label = "Last Child";
```

Accordion.selectedChild

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
accordionInstance.selectedChild
```

Description

Property; the selected child (of type `UIObject`) if one or more children exist; undefined if no children exist.

If the accordion has children, the code `accordionInstance.selectedChild` is equivalent to the code `accordionInstance.getChildAt(accordionInstance.selectedIndex)`.

Setting this property to a child causes the accordion to begin the transition animation to display the specified child.

Changing the value of `selectedChild` also changes the value of `selectedIndex`.

If the accordion has children, the default value is `accordionInstance.getChildAt(0)`. If the accordion doesn't have children, the default value is undefined.

Example

The following example detects when a child is selected and displays the child's order in the Output panel each time a header is selected:

```
// Create new Listener object.
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // Assign label of child panel to variable
    var selectedChild_str:String = my_acc.selectedChild.label;
    // Perform action based on selected child
    switch (selectedChild_str) {
        case "Shipping Address":
            trace("One was selected");
            break;
        case "Billing Address":
            trace("Two was selected");
            break;
        case "Payment":
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

See also

[Accordion.selectedIndex](#)

Accordion.selectedIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`accordionInstance.selectedIndex`

Description

Property; the zero-based index of the selected child in an accordion with one or more children. For an accordion with no child views, the only valid value is `undefined`.

Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on. The valid values of `selectedIndex` are 0, 1, 2, ... , $n - 1$, where n is the number of children.

Setting this property to a child causes the accordion to begin the transition animation to display the specified child.

Changing the value of `selectedIndex` also changes the value of `selectedChild`.

Example

The following example detects when a child is selected and displays the child's order in the Output panel each time a header is selected:

```
// Create new Listener object.
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // Assign label of child panel to variable.
    var selectedChild_num:Number = my_acc.selectedIndex;
    // Perform action based on selected child.
    switch (selectedChild_num) {
        case 0:
            trace("One was selected");
            break;
        case 1:
            trace("Two was selected");
            break;
        case 2:
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

See also

[Accordion.numChildren](#), [Accordion.selectedChild](#)

Alert component (Flash Professional only)

The Alert component lets you display a window that presents the user with a message and response buttons. The window has a title bar that you can fill with text, a message that you can customize, and buttons whose labels you can change. An Alert window can have any combination of Yes, No, OK, and Cancel buttons, and you can change the button labels by using the `Alert.okLabel`, `Alert.yesLabel`, `Alert.noLabel`, and `Alert.cancelLabel` properties. You cannot change the order of the buttons in an Alert window; the button order is always OK, Yes, No, Cancel. An Alert window closes when a user clicks any of its buttons.

To display an Alert window, call the `Alert.show()` method. In order to call the method successfully, the Alert component must be in the library. By dragging the Alert component from the Components panel to the Stage and then deleting the component, you add the component to the library without making it visible in the document.

The live preview for the Alert component is an empty window.

When you add an Alert component to an application, you can use the Accessibility panel to make the component's text and buttons accessible to screen readers. First, add the following line of code to enable accessibility:

```
mx.accessibility.AlertAccImpl.enableAccessibility();
```

NOTE

You enable accessibility for a component only once, regardless of how many instances you have of the component.

Using the Alert component (Flash Professional only)

You can use an Alert component whenever you want to announce something to a user. For example, you could display an alert when a user doesn't fill out a form properly, when a stock hits a certain price, or when a user quits an application without saving the session.

Alert parameters

The Alert component has no authoring parameters. You must call the ActionScript `Alert.show()` method to display an Alert window. You can use other ActionScript properties to modify the Alert window in an application. For more information, see [“Alert class \(Flash Professional only\)” on page 71](#).

Creating an application with the Alert component

The following procedure explains how to add an Alert component to an application while authoring. In this example, the Alert component appears when a stock hits a certain price.

To create an application with the Alert component:

1. Drag the Alert component from the Components panel to the current document's library.
This adds the component to the library, but doesn't make it visible in the application.
2. In the Actions panel, enter the following code in Frame 1 of the to define an event handler for the `click` event:

```
import mx.controls.Alert;

// Define action after alert confirmation.
var myClickHandler:Function = function (evt_obj:Object) {
    if (evt_obj.detail == Alert.OK) {
        trace("start stock app");
    }
};

// Show alert dialog box.
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
    Alert.CANCEL, this, myClickHandler, "stockIcon", Alert.OK);
```

This code creates an Alert window with OK and Cancel buttons. When the user clicks either button, Flash calls the `myClickHandler` function. The `myClickHandler` function instructs Flash to trace “start stock app” when you click the OK button.

NOTE

The `Alert.show()` method includes an optional parameter that displays an icon in the Alert window (in this example, an icon with the linkage identifier “stockIcon”). To include this icon in your test example, create a symbol named `stockIcon` and set it to Export for ActionScript in the Linkage Properties dialog box or the Create New Symbol dialog box. The graphics for the `stockIcon` symbol should be aligned to coordinates (0,0) in the symbol’s coordinate system.

3. Select Control > Test Movie.

Customizing the Alert component (Flash Professional only)

The Alert component positions itself in the center of the component that was passed as its *parent* parameter. The parent must be a `UIComponent` object. If it is a movie clip, you can register the clip as `mx.core.View` so that it inherits from `UIComponent`.

The Alert window automatically stretches horizontally to fit the message text or any buttons that are displayed. If you want to display large amounts of text, include line breaks in the text.

The Alert component does not respond to the `setSize()` method.

Using styles with the Alert component

You can set style properties to change the appearance of an Alert component. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

An Alert component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are “haloGreen”, “haloBlue”, and “haloOrange”. The default value is “haloGreen”.
<code>backgroundColor</code>	Both	The background color. The default color is white for the Halo theme and <code>0xEFEBEF</code> (light gray) for the Sample theme.

Style	Theme	Description
<code>borderStyle</code>	Both	The Alert component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. For more information, see “RectBorder class” on page 1063 . The Alert component has a component-specific <code>borderStyle</code> setting of <code>“alert”</code> with the Halo theme and <code>“outset”</code> with the Sample theme.
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>“_sans”</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is <code>10</code> .
<code>fontStyle</code>	Both	The font style: either <code>“normal”</code> or <code>“italic”</code> . The default value is <code>“normal”</code> .
<code>fontWeight</code>	Both	The font weight: either <code>“none”</code> or <code>“bold”</code> . The default value is <code>“none”</code> . All components can also accept the value <code>“normal”</code> in place of <code>“none”</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>“none”</code> .
<code>textAlign</code>	Both	The text alignment: either <code>“left”</code> , <code>“right”</code> , or <code>“center”</code> . The default value is <code>“left”</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>“none”</code> or <code>“underline”</code> . The default value is <code>“none”</code> .
<code>textIndent</code>	Both	A number indicating the text indent. The default value is <code>0</code> .

The Alert component includes three different categories of text. Setting the text properties for the Alert component itself provides default values for all three categories, as shown here:

```
import mx.controls.Alert;
_global.styles.Alert.setStyle("color", 0x000099);
Alert.show("This is a test alert", "Title");
```

To set the text styles for one category individually, the Alert component provides static properties that are references to a `CSSStyleDeclaration` instance.

Static property	Text affected
<code>buttonStyleDeclaration</code>	Button
<code>messageStyleDeclaration</code>	Message
<code>titleStyleDeclaration</code>	Title

The following example demonstrates how to set the title of an Alert component to be italicized:

```
import mx.controls.Alert;
import mx.styles.CSSStyleDeclaration;

var titleStyles = new CSSStyleDeclaration();
titleStyles.setStyle("fontWeight", "bold");
titleStyles.setStyle("fontStyle", "italic");

Alert.titleStyleDeclaration = titleStyles;

Alert.show("Name is a required field", "Validation Error");
```

The default title style declarations set `fontWeight` to "bold". When you override the `titleStyleDeclaration` property, this default is also overridden, so you must explicitly set `fontWeight` to "bold" if that setting is desired.

NOTE

Text styles set on an Alert component provide default text styles to its components through style inheritance. For more information, see "Setting inheriting styles on a container" in *Using Components*.

Using skins with the Alert component

The Alert component extends the Window component and uses its title background skin for the title background, a `RectBorder` class instance for its border, and Button skins for the visual states of its buttons. To skin the buttons and title bar while authoring, modify the Flash UI Components 2/Themes/MMDefault/Window Assets/Elements/TitleBackground and Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin symbols. For more information, see "About skinning components" in *Using Components*. The border and background are provided by the `RectBorder` class by default. For information on skinning the `RectBorder` class, see "[RectBorder class](#)" on page 1063.

An Alert component uses the following skin properties to dynamically skin the buttons and title bar:

Property	Description	Default value
<code>buttonUp</code>	The up state of the buttons.	<code>ButtonSkin</code>
<code>buttonUpEmphasized</code>	The up state of the default button.	<code>ButtonSkin</code>
<code>buttonDown</code>	The pressed state of the buttons.	<code>ButtonSkin</code>
<code>buttonDownEmphasized</code>	The pressed state of the default button.	<code>ButtonSkin</code>
<code>buttonOver</code>	The rolled-over state of the buttons.	<code>ButtonSkin</code>
<code>buttonOverEmphasized</code>	The rolled-over state of the default button.	<code>ButtonSkin</code>
<code>titleBackground</code>	The window title bar.	<code>TitleBackground</code>

To set the title of an Alert component to a custom movie clip symbol:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `TitleBackground`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.
6. The identifier is automatically filled out with `TitleBackground`.
7. Set the AS 2.0 class to `mx.skins.SkinElement`.

`SkinElement` is a simple class that can be used for all skin elements that don't provide their own ActionScript implementation. It provides movement and sizing functionality for components of version 2 of the Macromedia Component Architecture.
8. Make sure that Export in First Frame is already selected.
9. Click OK.
10. Open the new symbol for editing.
11. Use the drawing tools to create a box with a red fill and black line.
12. Set the border style to hairline.
13. Set the box, including the border, so that is positioned at (0,0) and has a width of 100 and height of 22.

The Alert component sets the proper width of the skin as needed, but it uses the existing height as the height of the title.
14. Click the Back button to return to the main timeline.

15. Drag an Alert component to the Stage and delete it.

This action adds the Alert component to the library and makes it available at runtime.

16. Add ActionScript code to the main timeline to create a sample Alert instance.

```
import mx.controls.Alert;
Alert.show("This is a skinned Alert component", "Title");
```

17. Select Control > Test Movie.

Alert class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > ScrollView > [Window component](#) > Alert

ActionScript Class Name mx.controls.Alert

To use the Alert component, you drag an Alert component to the Stage and delete it so that the component is in the document library but not visible in the application. Then you call `Alert.show()` to display an Alert window. You can pass parameters to `Alert.show()` that add a message, a title bar, and buttons to the Alert window.

Because ActionScript is asynchronous, the Alert component is not blocking, which means that the lines of ActionScript code that follow the call to `Alert.show()` run immediately. You must add listeners to handle the `click` events that are broadcast when a user clicks a button and then continue your code after the event is broadcast.

NOTE

In operating environments that are blocking (for example, Microsoft Windows), a call to `Alert.show()` does not return until the user has taken an action, such as clicking a button.

To understand more about the Alert class, see [“Window component” on page 1465](#) and [“PopUpManager class” on page 987](#).

Method summary for the Alert class

The following table lists the method of the Alert class.

Method	Description
Alert.show()	Creates an Alert window with optional parameters.

Methods inherited from the UIObject class

The following table lists the methods the Alert class inherits from the UIObject class.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Alert class inherits from the UIComponent class.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the Window class

The following table lists the methods the Alert class inherits from the Window class.

Method	Description
<code>Window.deletePopUp()</code>	Removes a window instance created by <code>PopUpManager.createPopUp()</code> .

Property summary for the Alert class

The following table lists properties of the Alert class.

Property	Description
<code>Alert.buttonHeight</code>	The height of each button, in pixels. The default value is 22.
<code>Alert.buttonWidth</code>	The width of each button, in pixels. The default value is 100.
<code>Alert.CANCEL</code>	A constant hexadecimal value indicating whether a Cancel button should be displayed in the Alert window.
<code>Alert.cancelLabel</code>	The label text for the Cancel button.
<code>Alert.NO</code>	A constant hexadecimal value indicating whether a No button should be displayed in the Alert window.
<code>Alert.noLabel</code>	The label text for the No button.
<code>Alert.OK</code>	A constant hexadecimal value indicating whether an OK button should be displayed in the Alert window.
<code>Alert.okLabel</code>	The label text for the OK button.
<code>Alert.YES</code>	A constant hexadecimal value indicating whether a Yes button should be displayed in the Alert window.
<code>Alert.yesLabel</code>	The label text for the Yes button.

Properties inherited from the UIObject class

The following table lists the properties the Alert class inherits from the UIObject class. When calling these properties from the Alert object, use the form `Alert.propertyName`.

Property	Description
<code>UIObject.bottom</code>	Read-only. The position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.

Property	Description
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the Alert class inherits from the UIComponent class. When calling these properties from the Alert object, use the form `Alert.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the Window class

The following table lists the properties the Alert class inherits from the Window class.

Property	Description
<code>Window.closeButton</code>	Indicates whether a close button is (<code>true</code>) or is not (<code>false</code>) included on the title bar.
<code>Window.content</code>	A reference to the content (root movie clip) of the window.
<code>Window.contentPath</code>	Sets the name of the content to display in the window.
<code>Window.title</code>	The text that appears in the title bar.
<code>Window.titleStyleDeclaration</code>	The style declaration that formats the text in the title bar.

Event summary for the Alert class

The following table lists an event of the Alert class.

Event	Description
<code>Alert.click</code>	Broadcast when a button in an Alert window is clicked.

Events inherited from the UIObject class

The following table lists the events the Alert class inherits from the UIObject class. When calling these events from the Alert object, use the form `Alert.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Alert class inherits from the UIComponent class. When calling these events from the Alert object, use the form `Alert.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the Window class

The following table lists the events the Alert class inherits from the Window class.

Event	Description
<code>Window.click</code>	Broadcast when the close button is clicked (released).
<code>Window.complete</code>	Broadcast when a window is created.

Alert.buttonHeight

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.buttonHeight`

Description

Property (class); a class (static) property that changes the height of the buttons. The default value is 22.

Example

With an Alert component already in the library, this example resizes the buttons:

```
import mx.controls.Alert;

// Adjust button sizes.
Alert.buttonHeight = 50;
Alert.buttonWidth = 150;

// Show alert dialog box.
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
    Alert.CANCEL);
```

See also

[Alert.buttonWidth](#)

Alert.buttonWidth

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.buttonWidth`

Description

Property (class); a class (static) property that changes the width of the buttons. The default value is 100.

Example

With an Alert component already in the library, add this ActionScript to the first frame of the main timeline to resize the buttons:

```
import mx.controls.Alert;

// Adjust button sizes.
Alert.buttonHeight = 50;
Alert.buttonWidth = 150;

// Show alert dialog box.
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
    Alert.CANCEL);
```

See also

[Alert.buttonHeight](#)

Alert.CANCEL

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.CANCEL`

Description

Property (constant); a property with the constant hexadecimal value 0x8. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that a Cancel button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the Cancel button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.CANCEL` and `Alert.OK` as values for the *flags* parameter and displays an `Alert` component with an OK button and a Cancel button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.cancelLabel

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.cancelLabel
```

Description

Property (class); a class (static) property that indicates the label text on the Cancel button.

Example

The following example sets the Cancel button's label to "cancellation":

```
Alert.cancelLabel = "cancellation";
```

Alert.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
var clickHandler:Object = function(eventObject:Object) {  
    // Insert code here.  
}  
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,  
    defaultButton]]]]])
```

Description

Event; broadcast to the registered listener when the OK, Yes, No, or Cancel button is clicked.

Version 2 components use a dispatcher/listener event model. The Alert component dispatches a `click` event when one of its buttons is clicked and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You call the `Alert.show()` method and pass it the name of the handler as a parameter. When a button in the Alert window is clicked, the listener is called.

When the event occurs, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Alert.click` event's event object has an additional `detail` property whose value is `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, or `Alert.NO`, depending on which button was clicked. For more information, see [“EventDispatcher class” on page 499](#).

Example

With an Alert component already in the library, add this ActionScript to the first frame of the main timeline to create an event handler called `myClickHandler`. The event handler is passed to the `Alert.show()` method as the fifth parameter. The event object is captured by `myClickHandler` in the `evt` parameter. The `detail` property of the event object is then used in a trace statement to send the name of the button that was clicked (`Alert.OK` or `Alert.CANCEL`) to the Output panel.

```
import mx.controls.Alert;

// Define button actions.
var myClickHandler:Function = function (evt_obj:Object) {
    switch (evt_obj.detail) {
        case Alert.OK :
            trace("You clicked: " + Alert.okLabel);
            break;
        case Alert.CANCEL :
            trace("You clicked: " + Alert.cancelLabel);
            break;
    }
};

// Display dialog box.
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL,
    this, myClickHandler);
```

Alert.NO

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.NO`

Description

Property (constant); a property with the constant hexadecimal value 0x2. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that a No button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the Cancel button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.NO` and `Alert.YES` as values for the *flags* parameter and displays an Alert component with a No button and a Yes button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.noLabel

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.noLabel
```

Description

Property (class); a class (static) property that indicates the label text on the No button.

Example

The following example sets the No button's label to "nyet":

```
Alert.noLabel = "nyet";
```

Alert.NONMODAL

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.NONMODAL`

Description

Property (constant); a property with the constant hexadecimal value 0x8000. This property can be used for the *flags* parameter of the `Alert.show()` method. This property indicates that an Alert window should be nonmodal, which allows users to interact with buttons and instances underneath the displayed window. By default, windows generated with `Alert.show()` are modal, which means that users cannot click anything except the currently open window.

Example

The following example displays two Button component instances on the Stage. Clicking one button opens a modal window, which prevents the user from further clicking the buttons until the Alert window is closed. The second button opens a nonmodal window, which allows the user to continue clicking the buttons underneath the currently open nonmodal Alert window. To test this example, add instances of both the Alert component and the Button component to the current document's library and add the following code to Frame 1 of the main timeline:

```
import mx.controls.Alert;

this.createClassObject(mx.controls.Button, "modal_button", 10, {_x:10,
    _y:10});
this.createClassObject(mx.controls.Button, "nonmodal_button", 20, {_x:120,
    _y:10});

modal_button.label = "modal";
modal_button.addEventListener("click", modalListener);
function modalListener(evt_obj:Object):Void {
    var a:Alert = Alert.show("This is a modal Alert window", "Alert Test",
        Alert.OK, this);
    a.move(100, 100);
}

nonmodal_button.label = "nonmodal";
```

```
nonmodal_button.addEventListener("click", nonmodalListener);
function nonmodalListener(evt_obj:Object):Void {
    var a:MovieClip = Alert.show("This is a nonmodal Alert window", "Alert
    Test", Alert.OK | Alert.NONMODAL, this);
    a.move(100, 100);
}
```

Alert.OK

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.OK

Description

Property (constant); a property with the constant hexadecimal value 0x4. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that an OK button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the OK button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.OK` and `Alert.CANCEL` as values for the *flags* parameter and displays an Alert component with an OK button and a Cancel button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.okLabel

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.okLabel
```

Description

Property (class); a class (static) property that indicates the label text on the OK button.

Example

The following example sets the OK button's label to "okay":

```
Alert.okLabel = "okay";
```

Alert.show()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,  
    defaultButton]]]]]])
```

Parameters

message The message to display.

title The text in the Alert title bar. This parameter is optional; if you omit it, the title bar is blank.

flags An optional parameter that indicates the buttons to display in the Alert window. The default value is `Alert.OK`, which displays an OK button. When you use more than one value, separate the values with a `|` character. Use one or more of the following values: `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`.

You can also use `Alert.NONMODAL` to indicate that the Alert window is nonmodal. A nonmodal window allows a user to interact with other windows in the application.

parent The parent window for the Alert component. The Alert window centers itself in the parent window. Use the value `null` or `undefined` to specify the `_root` timeline. The parent window must be a subclass of the `UIComponent` class, either as another Flash component that is a subclass of `UIComponent`, or as a custom window that is a subclass of the `UIComponent` (for more information see “About inheritance” in *Learning ActionScript 2.0 in Flash*). This parameter is optional.

clickHandler A handler for the `click` events broadcast when the buttons are clicked. In addition to the standard click event object properties, there is an additional `detail` property, which contains the flag value of the button that was clicked (`Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`). This handler can be a function or an object. For more information, see “Using listeners to handle events” in *Using Components*.

icon A string that is the linkage identifier of a symbol in the library; this symbol is used as an icon displayed to the left of the alert text. This parameter is optional.

defaultButton Indicates which button has initial focus and is clicked when a user presses Enter (Windows) or Return (Macintosh). If a user tabs to another button, that button is triggered when the Enter key is pressed.

This parameter can be one of the following values: `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`.

Returns

The Alert instance that is created.

Description

Method (class); a class (static) method that displays an Alert window with a message, an optional title, optional buttons, and an optional icon. The title of the alert appears at the top of the window and is left-aligned. The icon appears to the left of the message text. The buttons are centered below the message text and the icon.

Example

The following code is a simple example of a modal Alert window with an OK button:

```
mx.controls.Alert.show("Hello, world!");
```

The following code defines a click handler that sends a message to the Output panel about which button was clicked. (You must have an Alert component in the library for this code to display an alert; to add the component to the library, drag it to the Stage and then delete it):

```
import mx.controls.Alert;

// Define button actions.
var myClickHandler:Function = function (evt_obj:Object) {
    if (evt_obj.detail == Alert.OK) {
        trace(Alert.okLabel);
    } else if (evt_obj.detail == Alert.CANCEL) {
        trace(Alert.cancelLabel);
    }
};

// Display dialog box.
var dialog_obj:Object = Alert.show("Test Alert", "Test", Alert.OK |
    Alert.CANCEL, null, myClickHandler, "testIcon", Alert.OK);
```

Alert.YES

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.YES

Description

Property (constant); a property with the constant hexadecimal value 0x1. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that a Yes button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the Yes button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.NO` and `Alert.YES` as values for the *flags* parameter and displays an Alert component with a No button and a Yes button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.yesLabel

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.yesLabel
```

Description

Property (class); a class (static) property that indicates the label text on the Yes button.

Example

The following example sets the OK button's label to "da":

```
Alert.yesLabel = "da";
```


The Button component is a resizable rectangular user interface button. You can add a custom icon to a button. You can also change the behavior of a button from push to toggle. A toggle button stays pressed when clicked and returns to its up state when clicked again.

A button can be enabled or disabled in an application. In the disabled state, a button doesn't receive mouse or keyboard input. An enabled button receives focus if you click it or tab to it. When a Button instance has focus, you can use the following keys to control it:

Key	Description
Shift+Tab	Moves focus to the previous object.
Spacebar	Presses or releases the component and triggers the <code>click</code> event.
Tab	Moves focus to the next object.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each Button instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, in the live preview a custom icon is represented on the Stage by a gray square.

When you add the Button component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code:

```
mx.accessibility.ButtonAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component.

Using the Button component

A button is a fundamental part of any form or web application. You can use buttons wherever you want a user to initiate an event. For example, most forms have a Submit button. You could also add Previous and Next buttons to a presentation.

To add an icon to a button, you need to select or create a movie clip or graphic symbol to use as the icon. The symbol should be registered at 0,0 for appropriate layout on the button. Select the icon symbol in the Library panel, open the Linkage dialog box from the Library pop-up menu, and enter a linkage identifier. This is the value to enter for the icon parameter in the Property inspector or Component inspector. You can also enter this value for the `Button.icon` ActionScript property.

NOTE

If an icon is larger than the button, it extends beyond the button's borders.

To designate a button as the default push button in an application (the button that receives the click event when a user presses Enter), use `FocusManager.defaultPushButton`.

Button parameters

You can set the following authoring parameters for each Button component instance in the Property inspector or Component inspector (Window > Component Inspector menu option):

icon adds a custom icon to the button. The value is the linkage identifier of a movie clip or graphic symbol in the library; there is no default value.

label sets the value of the text on the button; the default value is `Button`.

labelPlacement orients the label text on the button in relation to the icon. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`; the default value is `right`. For more information, see `Button.labelPlacement`.

selected if the toggle parameter is `true`, this parameter specifies whether the button is pressed (`true`) or released (`false`). The default value is `false`.

toggle turns the button into a toggle switch. If `true`, the button remains in the down state when clicked and returns to the up state when clicked again. If `false`, the button behaves like a normal push button. The default value is `false`.

You can set the following additional parameters for each Button component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector will have no visible effect.

You can write `ActionScript` to control these and additional options for the `Button` component using its properties, methods, and events. For more information, see [“Button class” on page 101](#).

Creating an application with the `Button` component

The following procedure explains how to add a `Button` component to an application while authoring. In this example, the button displays a message when you click it.

To create an application with the `Button` component:

1. Drag a `Button` component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name `my_button`.
 - Enter `Click me` for the label parameter.
 - Enter `BtnIcon` for the icon parameter.

To use an icon, there must be a movie clip or graphic symbol in the library with a linkage identifier to use as the icon parameter. In this example, the linkage identifier is `BtnIcon`.
 - Set the `toggle` property to `true`.
3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
function clicked(){
    trace("You clicked the button!");
}
my_button.addEventListener("click", clicked);
```

The last line of code calls a `clicked` event handler function for the “click” event. This uses the method [“`EventDispatcher.addEventListener\(\)`” on page 501](#) with a custom function to handle the event.

4. Select Control > Test Movie.
5. When you click the button, Flash displays the message “You clicked the button!”.

To create a Button using ActionScript:

1. Drag the Button component from the Components panel to the current document’s library.

This adds the component to the library, but doesn’t make it visible in the application.

2. In the first frame of the main timeline, add the following ActionScript to the Actions panel to create a Button instance:

```
this.createClassObject(mx.controls.Button, "my_button", 10,
    {label:"Click me"});
my_button.move(20, 20);
```

The method `UIObject.createClassObject()` is used to create the Button instance named `my_button` and specify a label property. Then, the code uses the method `UIObject.move()` to position the button.

3. Now, add the following ActionScript to create an event listener and an event handler function:

```
function clicked() {
    trace("You clicked the button!");
}
my_button.addEventListener("click", clicked);
```

This uses the method “[EventDispatcher.addEventListener\(\)](#)” on page 501 with a custom function to handle the event.

4. Select Control > Test Movie.
5. When you click the button, Flash displays the message “You clicked the button!”.

As you use the Button component with other components, you can create more sophisticated event handler functions. In this example, the “click” event causes the Accordion component to change the display of the panels.

To use a Button event with another component:

1. Drag the Button component from the Components panel to the current document’s library.

This adds the component to the library, but doesn’t make it visible in the application.

2. Drag the Accordion component from the Components panel to the current document’s library.

3. In the first frame of the main timeline, add the following ActionScript to the Actions panel to create a Button instance:

```
this.createClassObject(mx.containers.Accordion, "my_acc", 0);
my_acc.move(10, 40);
my_acc.createChild(mx.core.View, "panelOne", {label: "Panel One"});
my_acc.createChild(mx.core.View, "panelTwo", {label: "Panel Two"});

this.createClassObject(mx.controls.Button, "panelOne_button", 10,
    {label:"Panel One"});
panelOne_button.move(10, 10);
this.createClassObject(mx.controls.Button, "panelTwo_button", 20,
    {label:"Panel Two"});
panelTwo_button.move(120, 10);
```

This process uses the method `UIObject.createClassObject()` to create the Button and Accordion instances. Then, the code uses the method `UIObject.move()` to position the instances.

4. Now, add the following ActionScript to create event listeners and event handler functions:

```
// Create Handler for the button event.
function changePanel(evt_obj:Object):Void {
    // Change Accordion view depending on button selected.
    switch (evt_obj.target._name) {
        case "panelOne_button" :
            my_acc.selectedIndex = 0;
            break;
        case "panelTwo_button" :
            my_acc.selectedIndex = 1;
            break;
    }
}

// Add Listeners for the buttons.
panelOne_button.addEventListener("click", changePanel);
panelTwo_button.addEventListener("click", changePanel);
```

This process uses the method `EventDispatcher.addEventListener()` with a custom function to handle the events.

5. Select Control > Test Movie.
6. When you click a button, the Accordion changes the current panel.

Customizing the Button component

You can transform a Button component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see `UIObject.setSize()`) or any applicable properties and methods of the Button class (see “Button class” on page 101). Resizing the button does not change the size of the icon or label.

The bounding box of a Button instance is invisible and also designates the hit area for the instance. If you increase the size of the instance, you also increase the size of the hit area. If the bounding box is too small to fit the label, the label is clipped to fit.

If an icon is larger than the button, the icon extends beyond the button’s borders.

Using styles with the Button component

You can set style properties to change the appearance of a button instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A Button component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Sample	The background color. The default value is 0xEFEBEF (light gray). The Halo theme uses 0xF8F8F8 (very light gray) for the button background color when the button is up and <code>themeColor</code> when the button is pressed. You can only modify the up background color in the Halo theme by skinning the button. See “Using skins with the Button component” on page 95.
<code>borderStyle</code>	Sample	The Button component uses a <code>RectBorder</code> instance as its border in the Sample theme and responds to the styles defined in that class. See “RectBorder class” on page 1063. With the Halo theme, the Button component uses a custom rounded border whose colors cannot be modified except for <code>themeColor</code> .

Style	Theme	Description
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

Using skins with the Button component

The Button component includes 32 different skins that can be customized to correspond to the border and icon in 16 different states. To skin the Button component while authoring, create new movie clip symbols with the desired graphics and set the symbol linkage identifiers using ActionScript. (For more information, see [“Using ActionScript to draw Button skins” on page 98.](#))

The default implementation of the Button skins provided with both the Halo and Sample themes uses the ActionScript drawing API to draw the button states, and uses a single movie clip symbol associated with one ActionScript class to provide all skins for the Button component.

The Button component has many skins because a button has so many states, and a border and icon for each state. The state of a Button instance is controlled by four properties and user interaction. The following properties affect skins:

Property	Description
<code>emphasized</code>	Provides two different looks for Button instances and is typically used to highlight one button, such as the default button in a form.
<code>enabled</code>	Shows whether or not the button is allowing user interaction.
<code>toggle</code>	Provides a selected and unselected value and uses different skins to demonstrate the current value. For a Button instance whose <code>toggle</code> property is set to <code>false</code> , the <code>false</code> skins are used. When the <code>toggle</code> property is <code>true</code> , the skin depends on the <code>selected</code> property.
<code>selected</code>	When the <code>toggle</code> property is set to <code>true</code> , determines if the Button is selected (<code>true</code> or <code>false</code>). Different skins are used to identify the value and, by default, are the only way this value is depicted on screen.

If a button is enabled, it displays its Over state when the pointer moves over it. The button receives input focus and displays its Down state when it's pressed. The button returns to its Over state when the mouse is released. If the pointer moves off the button while the mouse is pressed, the button returns to its original state and it retains input focus. If the `toggle` parameter is set to `true`, the state of the button does not change until the mouse is released over it.

If a button is disabled, it displays its disabled state, regardless of user interaction.

A Button component supports the following skin properties:

Property	Description
<code>falseUpSkin</code>	The up (normal) state.
<code>falseDownSkin</code>	The pressed state.
<code>falseOverSkin</code>	The over state.
<code>falseDisabledSkin</code>	The disabled state.
<code>trueUpSkin</code>	The toggled state.
<code>trueDownSkin</code>	The pressed-toggled state.
<code>trueOverSkin</code>	The over-toggled state.
<code>trueDisabledSkin</code>	The disabled-toggled state.
<code>falseUpSkinEmphasized</code>	The up (normal) state of an emphasized button.

Property	Description
<code>falseDownSkinEmphasized</code>	The pressed state of an emphasized button.
<code>falseOverSkinEmphasized</code>	The over state of an emphasized button.
<code>falseDisabledSkinEmphasized</code>	The disabled state of an emphasized button.
<code>trueUpSkinEmphasized</code>	The toggled state of an emphasized button.
<code>trueDownSkinEmphasized</code>	The pressed-toggled state of an emphasized button.
<code>trueOverSkinEmphasized</code>	The over-toggled state of an emphasized button.
<code>trueDisabledSkinEmphasized</code>	The disabled-toggled state of an emphasized button.
<code>falseUpIcon</code>	The icon up state.
<code>falseDownIcon</code>	The icon pressed state.
<code>falseOverIcon</code>	The icon over state.
<code>falseDisabledIcon</code>	The icon disabled state.
<code>trueUpIcon</code>	The icon toggled state.
<code>trueOverIcon</code>	The icon over-toggled state.
<code>trueDownIcon</code>	The icon pressed-toggled state.
<code>trueDisabledIcon</code>	The icon disabled-toggled state.
<code>falseUpIconEmphasized</code>	The icon up state of an emphasized button.
<code>falseDownIconEmphasized</code>	The icon pressed state of an emphasized button.
<code>falseOverIconEmphasized</code>	The icon over state of an emphasized button.
<code>falseDisabledIconEmphasized</code>	The icon disabled state of an emphasized button.
<code>trueUpIconEmphasized</code>	The icon toggled state of an emphasized button.
<code>trueOverIconEmphasized</code>	The icon over-toggled state of an emphasized button.
<code>trueDownIconEmphasized</code>	The icon pressed-toggled state of an emphasized button.
<code>trueDisabledIconEmphasized</code>	The icon disabled-toggled state of an emphasized button.

The default value for all skin properties ending in “Skin” is `ButtonSkin`, and the default for all “Icon” properties is `undefined`. The properties with the “Skin” suffix provide a background and border, whereas those with the “Icon” suffix provide a small icon.

In addition to the icon skins, the Button component also supports a standard `icon` property. The difference between the standard property and style property is that through the style property you can set icons for the individual states, whereas with the standard property only one icon can be set and it applies to all states. If a Button instance has both the `icon` property and icon style properties set, the instance may not behave as anticipated.

To see an interactive demo showing when each skin is used, see [Using Components Help](#).

Using ActionScript to draw Button skins

The default skins in both the Halo and Sample themes use the same skin element for all states and draw the actual graphics through ActionScript. The Halo implementation uses an extension of the `RectBorder` class and some custom drawing code to draw the states. The Sample implementation uses the same skin and the same ActionScript class as the Halo theme, with different properties set in ActionScript to alter the appearance of the Button.

To create an ActionScript class to use as the skin and provide different states, the skin can read the `borderStyle` style property of the skin and `emphasized` property of the parent to determine the state. The following table shows the border style that is set for each skin:

Property	Border style
<code>falseUpSkin</code>	<code>falseup</code>
<code>falseDownSkin</code>	<code>falsedown</code>
<code>falseOverSkin</code>	<code>falserollover</code>
<code>falseDisabled</code>	<code>falsedisabled</code>
<code>trueUpSkin</code>	<code>trueup</code>
<code>trueDownSkin</code>	<code>truedown</code>
<code>trueOverSkin</code>	<code>truerollover</code>
<code>trueDisabledSkin</code>	<code>truedisabled</code>

To create an ActionScript customized Button skin:

1. Create a new ActionScript class file.

For this example, name the file `RedGreenBlueSkin.as`.

2. Copy the following ActionScript to the file:

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueSkin extends RectBorder
{
    static var symbolName:String = "RedGreenBlueSkin";
    static var symbolOwner:Object = RedGreenBlueSkin;
```

```

function size():Void
{
    var c:Number; // color
    var borderStyle:String = getStyle("borderStyle");

    switch (borderStyle) {
        case "falseup":
        case "falserover":
        case "falsedisabled":
            c = 0x7777FF;
            break;
        case "falsedown":
            c = 0x77FF77;
            break;
        case "trueup":
        case "truedown":
        case "truerollover":
        case "truedisabled":
            c = 0xFF7777;
            break;
    }

    clear();
    var thickness = _parent.emphasized ? 2 : 0;
   LineStyle(thickness, 0, 100);
    beginFill(c, 100);
    drawRect(0, 0, __width, __height);
    endFill();
}

// Required for skins.
static function classConstruct():Boolean
{
    UIObjectExtensions.Extensions();
    _global.skinRegistry["ButtonSkin"] = true;
    return true;
}
static var classConstructed:Boolean = classConstruct();
static var UIObjectExtensionsDependency = UIObjectExtensions;
}

```

This class creates a square box based on the border style: a blue box for the false up, rollover, and disabled states; a green box for the normal pressed state; and a red box for the expanded child. It draws a hairline border in the normal case and a thick border if the button is emphasized.

3. Save the file.
4. Create a new FLA file and save it in the same folder as the AS file.
5. Create a new symbol by selecting Insert > New Symbol.
6. Set the name to `ButtonSkin`.
7. If the advanced view is not displayed, click the Advanced button.
8. Select Export for ActionScript.
The identifier is automatically filled out with `ButtonSkin`.
9. Set the AS 2.0 class to `RedGreenBlueSkin`.
10. Make sure that Export in First Frame is already selected, and click OK.
11. Drag a Button component to the Stage.
12. Select Control > Test Movie.

Using movie clips to customize Button skins

The example above demonstrates how to use an ActionScript class to customize the Button skin, which is the method used by the skins provided in both the Halo and Sample themes. However, because the example uses simple colored boxes, it is simpler in this case to use different movie clip symbols as the skins.

To create movie clip symbols for Button skins:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `RedButtonSkin`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.
The identifier is automatically filled out with `RedButtonSkin`.
6. Set the AS 2.0 class to `mx.skins.SkinElement`.
7. Make sure that Export in First Frame is already selected, and click OK.
8. Open the new symbol for editing.
9. Use the drawing tools to create a box with a red fill and black line.
10. Set the border style to hairline.
11. Set the box, including the border, so that it is positioned at (0,0) and has a width and height of 100.

The `SkinElement` class resizes the content as appropriate.

12. Repeat steps 2-11 and create green and blue skins, named accordingly.
13. Click the Back button to return to the main timeline.
14. Drag a Button component to the Stage.
15. Set the `toggled` property value to `true` to see all three skins.
16. Copy the following ActionScript code to the Actions panel with the Button instance selected.

```
onClipEvent(initialize) {  
    falseUpSkin = "BlueButtonSkin";  
    falseDownSkin = "GreenButtonSkin";  
    falseOverSkin = "BlueButtonSkin";  
    falseDisabledSkin = "BlueButtonSkin";  
    trueUpSkin = "RedButtonSkin";  
    trueDownSkin = "RedButtonSkin";  
    trueOverSkin = "RedButtonSkin";  
    trueDisabledSkin = "RedButtonSkin";  
}
```

17. Select Control > Test Movie.

Button class

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [SimpleButton class](#) > Button

ActionScript Class Name `mx.controls.Button`

The properties of the Button class let you do the following at runtime: add an icon to a button, create a text label, and indicate whether the button acts as a push button or as a toggle switch.

Setting a property of the Button class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The Button component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in *Using Components*.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Button.version);
```

NOTE

The code `trace(myButtonInstance.version)`; returns `undefined`.

The `Button` component class is different from the built-in `ActionScript Button` object.

Method summary for the `Button` class

There are no methods exclusive to the `Button` class.

Methods inherited from the `UIObject` class

The following table lists the methods the `Button` class inherits from the `UIObject` class. When calling these methods from the `Button` object, use the form `buttonInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Button class inherits from the UIComponent class. When calling these methods from the Button object, use the form *buttonInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the Button class

The following table lists properties of the Button class.

Property	Description
Button.icon	Specifies an icon for a button instance.
Button.label	Specifies the text that appears in a button.
Button.labelPlacement	Specifies the orientation of the label text in relation to an icon.

Properties inherited from the SimpleButton class

The following table lists the properties the Button class inherits from the SimpleButton class. When accessing these properties, use the form *buttonInstance.propertyName*.

Property	Description
SimpleButton.emphasized	Indicates whether a button has the look of a default push button.
SimpleButton.emphasizedStyleDeclaration	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
SimpleButton.selected	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
SimpleButton.toggle	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the UIObject class

The following table lists the properties the Button class inherits from the UIObject class.

When accessing these properties from the Button object, use the form

buttonInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the Button class inherits from the UIComponent class.

When accessing these properties from the Button object, use the form

buttonInstance.propertyName.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the Button class

There are no events exclusive to the Button class.

Events inherited from the SimpleButton class

The following table lists the events the Button class inherits from the SimpleButton class.

Property	Description
<code>SimpleButton.click</code>	Broadcast when a button is clicked.

Events inherited from the UIObject class

The following table lists the events the Button class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Button class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Button.icon

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.icon

Description

Property; a string that specifies the linkage identifier of a symbol in the library to be used as an icon for a button instance. The icon can be a movie clip symbol or a graphic symbol with an upper left registration point. You must resize the button if the icon is too large to fit; neither the button nor the icon resizes automatically. If an icon is larger than a button, the icon extends over the borders of the button.

To create a custom icon, create a movie clip or graphic symbol. Select the symbol on the Stage in symbol-editing mode and enter 0 in both the X and Y boxes in the Property inspector. In the Library panel, select the movie clip and select Linkage from the Library pop-up menu. Select Export for ActionScript, and enter an identifier in the Identifier text box.

The default value is an empty string (""), which indicates that there is no icon.

Use the `labelPlacement` property to set the position of the icon in relation to the button.

NOTE

The icon does not appear on the Stage in Flash. You must select Control > Test Movie to see the icon.

Example

With a button on the Stage with instance name `my_button`, the following code assigns the movie clip from the Library panel with the linkage identifier `happiness` to the Button instance as an icon:

```
my_button.icon = "happiness";
```

You can also create the button and assign the icon entirely in ActionScript using the method `UIObject.createClassObject()` (you still must have already created an icon for the button with the linkage identifier `happiness`). First drag the Button component from the Components panel to the current document's library, so the component appears in the library, but not on the Stage. Then, in the first frame of the main timeline, add the following ActionScript:

```
this.createClassObject(mx.controls.Button, "my_button", 1, {icon:  
    "happiness"});
```

See also

[Button.labelPlacement](#)

Button.label

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.label

Description

Property; specifies the text label for a button instance. By default, the label appears centered on the button. Calling this method overrides the label authoring parameter specified in the Property inspector or the Component inspector. The default value is "Button".

Example

With a button on the Stage with instance name `my_button`, the following code sets the label to "Test Button":

```
my_button.label = "Test Button";
```

You can also create the button and assign the label entirely in ActionScript using the method `UIObject.createClassObject()`. First drag the Button component from the Components panel to the current document's library, so the component appears in the library, but not on the Stage. Then, in the first frame of the main timeline, add the following ActionScript:

```
this.createClassObject(mx.controls.Button, "my_button", 1, {label: "Test Button"});
```

See also

[Button.labelPlacement](#)

Button.labelPlacement

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.labelPlacement

Description

Property; sets the position of the label in relation to the icon. The default value is "right". The following are the four possible values; the icon and label are always centered vertically and horizontally within the bounding area of the button:

- "right" The label is set to the right of the icon.
- "left" The label is set to the left of the icon.
- "bottom" The label is set below the icon.
- "top" The label is set above the icon.

Example

With a button on the Stage with instance name `my_button`, and a symbol in the Library panel with the linkage identifier `happiness`, the following code sets the label alignment to the left of the icon:

```
my_button.icon = "happiness";  
my_button.label = "Test Button";  
my_button.labelPlacement = "left";
```

You can also create the button and set the alignment entirely in ActionScript using the method `UIObject.createClassObject()`. First drag the Button component from the Components panel to the current document's library, so it appears in the library, but not on the Stage. Then, in the first frame of the main timeline, add the following ActionScript:

```
this.createClassObject(mx.controls.Button, "my_button", 1, {label: "Test  
Button", icon: "happiness", labelPlacement: "left"});
```

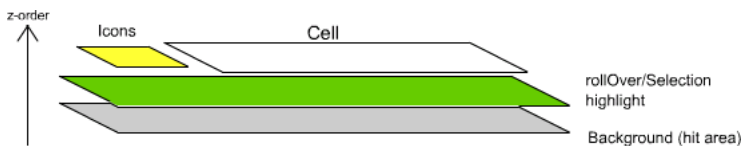

The CellRenderer API is a set of properties and methods that the list-based components (List, DataGrid, Tree, Menu, and ComboBox) use to manipulate and display custom cell content for each of their rows. This customized cell can contain a prebuilt component, such as a CheckBox component, or any class you create.

Understanding the List class

To use the CellRenderer API, you need an advanced understanding of the List class. The DataGrid, Tree, Menu, and ComboBox components are extensions of the List class, so understanding the List class lets you understand them as well.

About the composition of the List component

List components are composed of rows. These rows display rollover and selection highlights, are used as hit states for row selection, and play a vital part in scrolling. Aside from selection highlights and icons (such as the node icons and expander arrows of a Tree component), a row consists of one cell (or, in the case of the DataGrid component, many cells). In the default case, these cells are TextField objects that implement the CellRenderer API. However, you can tell a List component to use a different class of component as the cell for each row. The only requirement is that the class must implement the CellRenderer API, which the List component uses for communicating with the cell.



The stacking order of a row in a List or DataGrid component

NOTE

If a cell has button event handlers (`onPress` and so on), the background hit area may not receive input necessary to trigger the events.

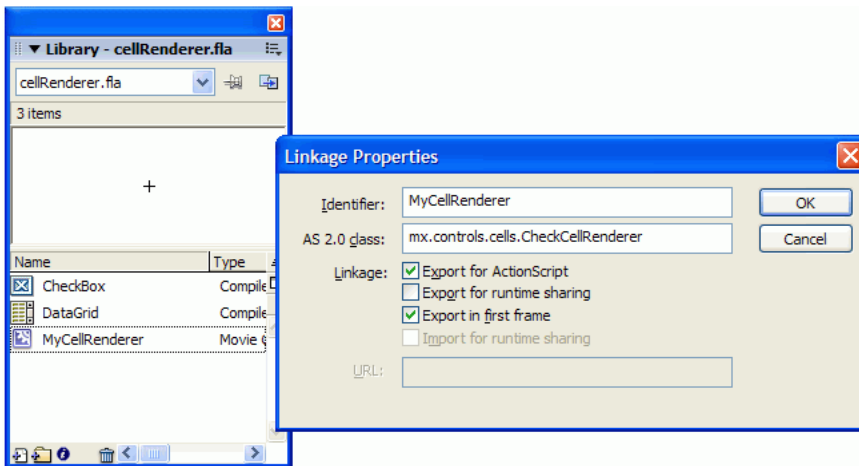
About the scrolling behavior of the List component

The List class uses a fairly complex algorithm for scrolling. A list only lays out as many rows as it can display at once; items beyond the value of the `rowCount` property don't get rows at all. When the list scrolls, it moves all the rows up or down (depending on the scrolling direction). The list then recycles the rows that are scrolled out of view; it reinitializes them and uses them for the new rows being scrolled into view. To do this, it sets the value of the old row to the new item in the view and moves the old row to where the new item is scrolled into view.

Because of this scrolling behavior, you cannot expect a cell to be used for only one value. Recycling of rows means that the cell renderer must know how to completely reset its state when it is set to a new value. For example, if your cell renderer creates an icon to display one item, it might need to remove that icon when another item is rendered with it. Assume your cell renderer is a container that will be filled with numerous item values over time, and it has to know how to completely change itself from displaying one value to displaying another. In fact, your cell should even know how to properly render undefined items, which might mean removing all old content in the cell.

Using the CellRenderer API

You must write a class with four methods (`CellRenderer.getPreferredHeight()`, `CellRenderer.getPreferredWidth()`, `CellRenderer.setSize()` and `CellRenderer.setValue()`) that the list-based component uses to communicate with the cell (if the class extends `UIObject`, you can use `size()` instead of `CellRenderer.setSize()`). The class must be specified in the AS 2.0 Class text box in the Linkage Properties dialog box of a movie clip symbol in your Flash application.



You can look at the `CheckCellRenderer` class that implements the Cell Renderer API for an example; it's located in `First Run/classes/mx/controls/cells`. Also, see the `DataGrid` component documentation for CellRenderer related information, including “[DataGrid performance strategies](#)” on page 256.

There are two methods and a property (`CellRenderer.getCellIndex()`, `CellRenderer.getDataLabel()`, and `CellRenderer.listOwner`) that are given automatically to a cell to allow it to communicate with the list-based component. For example, suppose a cell contains a check box that, when selected, causes a row to be selected. The cell renderer needs a reference to the list-based component that contains it in order to call the component's `selectedIndex` property. Also, the cell needs to know which item index it is currently rendering so that it can set `selectedIndex` to the correct number; the cell can use `CellRenderer.listOwner` and `CellRenderer.getCellIndex()` to do so. You do not need to implement these ActionScript elements; the cell receives them automatically when it is placed in the list-based component.

Simple cell renderer example

This section presents an example of a cell renderer that displays multiple lines of text in a cell. The following tutorial shows how to create a cell renderer class that displays multiple lines of text in the cells of a DataGrid component.

The completed files, MultiLineCell.as and CellRenderer_tutorial.fla are located at www.macromedia.com/go/component_samples.

Creating the MultiLineCell cell renderer class

A cell renderer class must implement the following methods:

- `CellRenderer.getPreferredHeight()`
- `CellRenderer.getPreferredWidth()`

The `CellRenderer.getPreferredWidth()` method is necessary for Menu components or DataGrid headers only; otherwise, comment it out of the code, as shown in the example.

- `CellRenderer.setSize()`

If a cell renderer class extends `UIObject`, use `implement size()` instead, as shown in this example.

- `CellRenderer.setValue()`

A cell renderer class must also declare the methods and property received from the List class:

- `CellRenderer.getCellIndex()`
- `CellRenderer.getDataLabel()`
- `CellRenderer.listOwner`

The following steps show how to create an ActionScript 2.0 cell renderer class file called **MultiLineCell.as** and link it to a new movie clip symbol in a new Flash document. Then, you can add a DataGrid component to the Flash document library. On the first frame, you add ActionScript that creates the DataGrid dynamically and assigns the MultiLineCell class as the cell renderer for one of its columns:

To create the multiLineCell cell renderer class:

1. In Flash, select File > New > ActionScript File (not Flash Document). Save the document as **MultiLineCell.as**.
2. Enter the following code into MultiLineCell.as:

```
// ActionScript 2.0 class.
class MultiLineCell extends mx.core.UIComponent
{
    private var multiLineLabel; // The label to be used for text.
```

```

private var owner; // The row that contains this cell.
private var listOwner; // The List, data grid or tree containing this
cell.

// Cell height offset from the row height total and preferred cell
width.
private static var PREFERRED_HEIGHT_OFFSET = 4;
private static var PREFERRED_WIDTH = 100;
// Starting depth.
private var startDepth: Number = 1;

// Constructor. Should be empty.
public function MultiLineCell()
{
}

/* UIObject expects you to fill in createChildren by instantiating all
the movie clip assets you might need upon initialization. In this case
we are creating one label*/
public function createChildren():Void
{
    // The createLabel method is a useful method of UIObject and a handy
    // way to make labels in components.
    var c = multiLineLabel = this.createLabel("multiLineLabel",
startDepth);
    // Links the style of the label to the style of the grid
    c.styleName = listOwner;
    c.selectable = false;
    c.tabEnabled = false;
    c.background = false;
    c.border = false;
    c.multiline = true;
    c.wordWrap = true;
}

public function size():Void
{
/* By extending UIComponent which imports UIObject, you get setSize
automatically, however, UIComponent expects you to implement size().
Assume __width and __height are set for you now. You're going to
expand the cell to fit the whole rowHeight. The rowHeight itself is a
property of the list type component that we are rendering a cell in.
Since we want the rowHeight to fit two lines, when creating the list
type component using this cellRenderer class, make sure its rowHeight
property is set large enough that two lines of text can render within
it.*/

/*__width and __height are the underlying variables of the getter/
setters .width and .height.*/
    var c = multiLineLabel;

```

```

        c.setSize(__width, __height);
    }

    // Provides the preferred height of the cell. Inherited method.
    public function getPreferredHeight():Number
    {
/* The cell is given a property, "owner", that references the row. It's
always preferred that the cell take up most of the row's height. In
this case we will keep the cell slightly smaller.*/
        return owner.__height - PREFERRED_HEIGHT_OFFSET;
    }

    // Called by the owner to set the value in the cell. Inherited method.
    public function setValue(suggestedValue:String, item:Object,
selected:Boolean):Void
    {
/* If item is undefined, nothing should be rendered in the cell, so set
the label as invisible. Note: For scrolling List type components like
a scrolling datagrid, the cells are intended to be empty as they
scroll just out of sight, and then the cell is reused again and set to
a new value producing an animated effect of scrolling. For this
reason, you cannot rely on any one cell always having data to show or
the same value.*/
        if (item!=undefined){
            multiLineLabel.text._visible = false;
        }
        multiLineLabel.text = suggestedValue;
    }
    // function getPreferredWidth :: only for menus and DataGrid headers
    // function getCellIndex :: not used in this cell renderer
    // function getDataLabel :: not used in this cell renderer
}

```

Creating an application to test the MultiLineCell cell renderer class

In the following steps, you will create the DataGrid instance and implement the MultiLineCell class.

To create an application with a DataGrid component that uses the MultiLineCell cell renderer class:

1. In Flash, select File > New > Flash Document.
2. Select File > Save As, name the file `cellRender_tutorial fla`, and save the file to the same folder as the `MultiLineCell.as` file.
3. To create a new movieClip symbol to link to the `MultiLineCell` class, select Insert > New Symbol.

4. Click the Advanced button in the lower-right corner of the Create New Symbol dialog box to enable more options.

The Advanced button is available when you are in the basic mode of the Create New Symbol dialog box. If you don't see the Advanced button, you are probably already in the Advanced view of the dialog box.

5. In the Name text box, type **MultiLineCell**.

The default value for Type is Movie Clip. Leave Movie Clip selected.

6. Click the Export for ActionScript check box in the Linkage section.

Enabling this option allows you to dynamically attach instances of this symbol to your Flash documents during runtime. The Identifier text box will automatically show MultiLineCell.

7. Set the ActionScript 2.0 Class to MultiLineCell (to match the class name of the MultiLineCell cell renderer class created previously).

8. Enable the Export in first frame check box and click OK to apply your changes and close the dialog box.

NOTE

If you need to modify the MultiLineCell Movie Clip symbol's Linkage properties at a later time, you can right click the symbol in the document's library and select Properties or Linkage from the menu.

9. Drag the DataGrid component from the Components panel to the library.

The DataGrid instance will be created dynamically through ActionScript in the following step.

10. Select the first frame on the main Timeline (make sure you are not still in the MultiLineCell movie-clip editing mode).

11. In the Actions panel for the first frame, enter the following code to create a DataGrid dynamically, assign data to the DataGrid, and assign your new cell renderer class:

```
// Create a new DataGrid component instance
this.createClassObject(mx.controls.DataGrid, "myGrid_dg", 1);

// Build a data provider for the data grid with four columns of data.
myDP = new Array();
var aLongString:String = "An example of a cell renderer class that
    displays a multiple line TextField";
myDP.addItem({firstName:"Winston", lastName:"Elstad", note:aLongString,
    item:100});
myDP.addItem({firstName:"Ric", lastName:"Dietrich", note:aLongString,
    item:101});
myDP.addItem({firstName:"Ewing", lastName:"Canepa", note:aLongString,
    item:102});
```

```

myDP.addItem({firstName:"Kevin", lastName:"Wade", note:aLongString,
    item:103});
myDP.addItem({firstName:"Kimberly", lastName:"Dietrich",
    note:aLongString, item:104});
myDP.addItem({firstName:"AJ", lastName:"Bilow", note:aLongString,
    item:105});
myDP.addItem({firstName:"Chuck", lastName:"Yushan", note:aLongString,
    item:106});
myDP.addItem({firstName:"John", lastName:"Roo", note:aLongString,
    item:107});

/* Assign the data provider to the DataGrid to populate it. Note: This
    has to be done before applying the cellRenderers. */
myGrid_dg.dataProvider = myDP;

/* Set some basic grid properties. Note: The data grid's row height
    should reflect the number of lines you expect to show in the
    MultiLineCell cell renderer. The cell renderer will size to the row
    height. This should be about 40 for 2 lines or 60 for 3 lines at
    default text size.*/
myGrid_dg.setSize(430,200);
myGrid_dg.move(40,40);
myGrid_dg.rowHeight = 40; // Allows for 2 lines of text at default text
    size.
myGrid_dg.getColumnAt(0).width = 70;
myGrid_dg.getColumnAt(1).width = 70;
myGrid_dg.getColumnAt(2).width = 220;
myGrid_dg.resizableColumns = true;
myGrid_dg.vScrollPolicy = "auto";
myGrid_dg.setStyle("backgroundColor", 0xD5D5FF);

// Assign cellRenderers.
myGrid_dg.getColumnAt(2).cellRenderer = "MultiLineCell";

```

12. Save the Flash document, and select **Control > Test Movie**.

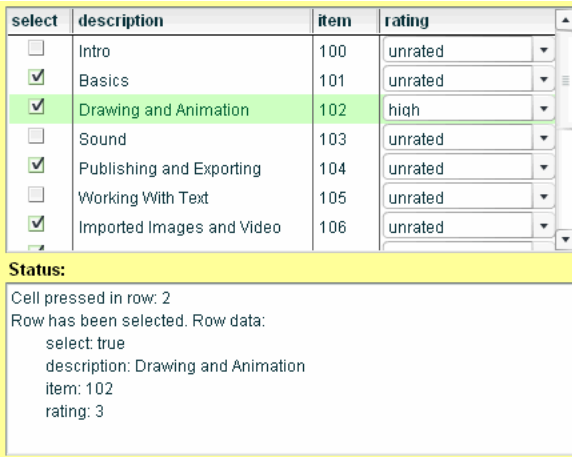
A data grid appears. The third column of the data grid contains a multiple line cell.

firstName	lastName	note	item
Kimberly	Dietrich	An example of a cell renderer class that displays a multiple line TextField	104
AJ	Bilow	An example of a cell renderer class that displays a multiple line TextField	105
Chuck	Yushan	An example of a cell renderer class that displays a multiple line TextField	106
John	Roo	An example of a cell renderer class that displays a multiple line TextField	107

The completed MultiLineCell cell renderer example.

Additional cell renderer examples

Additional examples of cell renderer classes that display a ComboBox and a CheckCell component are also provided. These files are located in the CellRenderers_sample folder within the Samples and Tutorials folder on your hard disk at www.macromedia.com/go/component_samples.



The screenshot shows a table with four columns: 'select', 'description', 'item', and 'rating'. The 'select' column contains checkboxes, and the 'rating' column contains dropdown menus. The row with 'Drawing and Animation' is highlighted in green. Below the table is a yellow status window with the following text:

```
Status:  
Cell pressed in row: 2  
Row has been selected. Row data:  
  select: true  
  description: Drawing and Animation  
  item: 102  
  rating: 3
```

The additional installed sample named CellRenderers_Sample displaying a ComboBox and CheckBox.

Methods to implement for the CellRenderer API

You must write a class with the following methods so that the List, DataGrid, Tree, or Menu component can communicate with the cell.

Method	Description
<code>CellRenderer.getPreferredHeight()</code>	Returns the preferred height of a cell.
<code>CellRenderer.getPreferredWidth()</code>	The preferred width of a cell.
<code>CellRenderer.setSize()</code>	Sets the width and height of a cell.
<code>CellRenderer.setValue()</code>	Sets the content to be displayed in the cell.

Methods provided by the CellRenderer API

The List, DataGrid, Tree, and Menu components give the following methods to the cell when it is created within the component. You do not need to implement these methods.

Method	Description
CellRenderer.getCellIndex()	Returns an object with two fields, <code>columnIndex</code> and <code>itemIndex</code> , that indicate the position of the cell.
CellRenderer.getDataLabel()	Returns a string containing the name of the cell renderer's data field.

Properties provided by the CellRenderer API

The List, DataGrid, Tree, and Menu component give the following properties to the cell when it is created within the component. You do not need to implement these properties.

Property	Description
CellRenderer.listOwner	A reference to the List component that contains the cell.
CellRenderer.owner	A reference to the row that contains the cell.

CellRenderer.getCellIndex()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getCellIndex()
```

Parameters

None.

Returns

An object with two fields: `columnIndex` and `itemIndex`.

Description

Method; returns an object with two fields, `columnIndex` and `itemIndex`, that locate the cell in the component. Each field is an integer that indicates a cell's column position and item position. For any components other than the `DataGrid` component, the value of `columnIndex` is always 0.

This method is provided by the `List` class; you do not have to implement it. Declare it in your cell renderer class as follows, and use it in the functions in your cell renderer:

```
var getCellIndex:Function;
```

Example

This example edits a `DataGrid` component's data provider from within a cell:

```
var index = getCellIndex();  
var colName = listOwner.getColumnAt(index.columnIndex).columnName;  
listOwner.dataProvider.editField(index.itemIndex, colName, someVal);
```

CellRenderer.getDataLabel()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getDataLabel()
```

Parameters

None.

Returns

A string.

Description

Method; returns a string containing the name of the cell renderer's data field. For the `DataGrid` component, this method returns the column name for the current cell.

This method is provided by the `List` class; you do not have to implement it. Declare it in your cell renderer class as follows, and use it in the functions in your cell renderer:

```
var getDataLabel:Function;
```

Example

The following code tells the cell the name of the data field that it is rendering. For example, if the name of the data field currently being rendered by the cell is "Price", the variable `p` is now equal to "Price":

```
var p = getDataLabel();
```

CellRenderer.getPreferredHeight()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getPreferredHeight()
```

Parameters

None.

Returns

The correct height for the cell.

Description

Method; returns the preferred height of a cell. This is especially important for getting the right height of text within the cell. If you set this value higher than the `rowHeight` property of the component, cells will bleed above and below the rows.

This method is not provided by the `List` class; you must implement it. It tells the rows of the list how to center the cell and how to adjust the cell's height if necessary. If necessary, you can return a constant (for example, 22), or you can measure and return the height of the contents. You can also return `owner.height`, which is the height of the row.

Example

This example returns the value 20, which indicates that the cell should be 20 pixels high:

```
function getPreferredHeight(Void) :Number
{
    return 20;
}
```

This example returns a value that is 4 pixels less than the height of the row:

```
function getPreferredHeight():Number
{
    /* You know the cell is given a property, "owner", which is the row. It's
       always preferred for the cell to take up most of the row's height.
    */
    return owner.__height - 4;
}
```

CellRenderer.getPreferredWidth()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.getPreferredWidth()

Parameters

None.

Returns

A value (of type Number) that indicates the correct width of the cell.

Description

Method; the preferred width of a cell. If you specify a width greater than that of the component, the cell may be cut off.

Implement this method for the Menu component. Your cell is sized to whatever the width of the row is, except in a menu, which must measure the text for the width of the row. You can also implement this method for the DataGrid component where the header renderer checks whether or not to show the sort arrow.

Example

This example returns the value multiplied by 3, which indicates that the cell should be three times bigger than the length of the string it is rendering:

```
function getPreferredWidth():Number
{
    return myString.length*3;
}
```

This example comments out the `getPreferredWidth()` method:

```
// function getPreferredWidth :: only for a menu or datagrid
```

CellRenderer.listOwner

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.listOwner
```

Description

Property; a reference to the list that owns the cell. That list can be a DataGrid, Tree, List, or Menu component.

This method is provided by the List class; you do not have to implement it. Declare it in your cell renderer class as follows, and use it as a reference back to the list (or tree, menu, or grid):

```
var listOwner:MovieClip; // or UIObject, etc.
```

Example

This example finds the list's selected item in a cell:

```
var s = listOwner.selectedItem;
```

CellRenderer.owner

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.owner

Description

Property; a reference to the row that contains the cell.

This method is provided by the List class; you do not have to implement it. Declare it in your cell renderer class and use it as a reference:

```
var owner:MovieClip; // or UIObject, etc.
```

CellRenderer.setSize()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.setSize(*width*, *height*)

Parameters

width A number that indicates the width at which to lay out the component.

height A number that indicates the height at which to lay out the component.

Returns

Nothing.

Description

Method; lets the list tell its cells the size at which they should lay themselves out. The cell renderer should do layout so that it fits in the specified area, or the cell may bleed into other parts of the list and appear broken.

If the cell renderer extends the `UIObject` class, you should implement the `size()` method instead. Write the same function that you would write for `setSize()`, but use the `width` and `height` properties instead of parameters.

Example

The following example sizes an image in the cell to fit within the bounds specified by the list:

```
function setSize(w:Number, h:Number):Void
{
    image._width = w-2;
    image._height = h-2;
    image._x = image._y = 1;
}
```

This example is in a cell renderer class that extends `UIComponent` (which extends `UIObject`), so you must implement `size()` instead of `setSize()`, as follows:

```
// By extending UIComponent, you get setSize for free;
// however, UIComponent expects you to implement size().
// Assume __width and __height are set for you now.
// You're going to expand the cell to fit the whole rowHeight.

function size():Void
{
    // __width and __height are the underlying variables
    // of the getters/setters .width and .height.
    var c = multilineLabel;
    c._width = __width;
    c._height = __height;
}
```

CellRenderer.setValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setValue(suggested, item, selected)
```

Parameters

suggested A value to be used for the cell renderer's text, if any is needed.

item An object that is the entire item to be rendered. The cell renderer can use properties of this object for rendering.

selected A string with the following possible values: "normal", "highlighted", and "selected".

Returns

Nothing.

Description

Method; takes the values given and creates a representation of them in the cell. This resolves any difference between what was displayed in the cell and what needs to be displayed in the cell for the new item. (Remember that any cell could display many values during its time in the list.) This is the most important `CellRenderer` method, and you must implement it in every cell renderer.

The `setValue()` method is called frequently (for example, when a rollover, a selection, column resizing, or scrolling occurs). It is important to remember that a cell might not exist on the Stage and should not always be updated with data when `setValue()` is called. For example, at any moment, a particular cell may be scrolled out of the display area or it might be reused to render another value. For this reason, you cannot directly reference a specific cell renderer instance in the grid, and you should write `if` statements in the body of `setValue()` that allow code to run only if the `item` parameter is defined and a change has occurred. An undefined `item` parameter indicates that the cell should be visibly empty and any items in the cell should be assigned a `_visible` property of `false`. Cells can be required to be visibly empty temporarily, such as when scrolling occurs in a `DataGrid`.

If a row is selected and the pointer is over it, the value of the `selected` parameter is "highlighted", not "selected". This can cause problems if you're trying to make the cell renderer behave differently according to whether the row is in a selected state. To test whether the current row is in a selected state, use the following code:

```
var reallySelected:Boolean = selected != "normal" && listOwner.selectedNode
    == item;
```

Example

The following example shows how to use `setValue()` and `editField()` to reference a cell renderer instance in a grid.

Because a cell might not exist on the Stage (it might be scrolled out of the display area or it might be reused to render another value) at any time, you cannot directly reference a specific cell renderer instance in the grid.

Instead, use the data provider to communicate with a specific cell in the grid. The data provider holds all the state information about the grid. To display a given cell as enabled or selected (checked), there should be a corresponding field in the data provider to hold that information. The `setValue()` method of your cell renderer communicates changes in the data provider's state to the cell. The following is a `setValue()` implementation from a theoretical cell renderer that renders a check box in the cells:

```
function setValue(str, itm, sel)
{
  /* Assume the data provider has two relevant fields for this cell : checked
  and enabled.
  The form of such a data provider might look like this:
  [
  {field1:"DisplayMe", field2:"SomeString", checked:true, enabled:false}
  {field1:"DisplayMe", field2:"SomeString", checked:false, enabled:true}
  {field1:"DisplayMe", field2:"SomeString", checked:true, enabled:true}
  ]
  */

  /* Hide anything normally rendered in the cell if itm is undefined.
  Otherwise update the cell contents with the new data.
  */
  if (itm == undefined){
    myCheck._visible = false;
  }else{

    // redundancy checking
    if (myCheck.selected!=itm.checked){
      myCheck.selected = itm.checked;
    }
    if (myCheck.enabled!=itm.enabled){
      myCheck.enabled = itm.enabled;
    }
  }
}
```


If you want to enable the check box on the second row, you communicate through the data provider. Any change to the data provider (when made through a `DataProvider` method such as `DataProvider.editField()`) calls `setValue()` to refresh the display of the grid. This code would be written in the Flash application, either on a frame, on an object, or in another class file (but not in the cell renderer class file):

```
// calls setValue() again
myGrid.editField(1, "enabled", true);
```

The following example loads an image in a loader component within the cell, depending on the value passed:

```
function setValue(suggested, item, selected) : Void
{
    /* Hide anything normally rendered in the cell if item is undefined.
       Otherwise update the cell contents with the new data.
    */
    if (item == undefined){
        loader._visible = false;
    }else{
        // clear the loader
        loader.contentPath = undefined;
        // the list has URLs for different images in its data provider
        if (suggested!=undefined){
            loader.contentPath = suggested;
        }
    }
}
```

The following example is from a multiline text cell renderer:

```
function setValue(suggested:String, item:Object, selected:Boolean):Void
{
    /* Hide anything normally rendered in the cell if item is undefined.
       Otherwise update the cell contents with the new data.
    */
    if (item == undefined){
        multiLineLabel._visible = false;
    }else{
        // adds the text to the label
        multiLineLabel.text = suggested;
    }
}
```

The following example is from a radio button renderer. If the `item` parameter is undefined, then the cell may be scrolled out of the display area and should be visibly empty. An `if` statement is used to determine if the `item` parameter is undefined. If the `item` parameter is undefined, the radio button is hidden by setting its `_visible` property to `false`; otherwise, the radio button is updated with the new data and appears.

```
function setValue(str:String, item:Object, sel:String) : Void {
/* Hide anything normally rendered in the cell if item is undefined.
   Otherwise update the cell contents with the new data.
*/
  if (item == undefined) {
    radio._visible = false; }
  else {
    trace(item.data + " " + item.label + " " + item.state + " " + sel);
    radio.label = item.label;
    radio.data = item.data;
    radio.selected = item.state;
    radio._visible = true;
  }
}
```

A check box is a square box that can be selected or deselected. When it is selected, a check mark appears in the box. You can add a text label to a check box and place it to the left, right, top, or bottom.

A check box can be enabled or disabled in an application. If a check box is enabled and a user clicks it or its label, the check box receives input focus and displays its pressed appearance. If a user moves the pointer outside the bounding area of a check box or its label while pressing the mouse button, the component's appearance returns to its original state and it retains input focus. The state of a check box does not change until the mouse is released over the component. Additionally, the check box has two disabled states, selected and deselected, which do not allow mouse or keyboard interaction.

If a check box is disabled, it displays its disabled appearance, regardless of user interaction. In the disabled state, a button doesn't receive mouse or keyboard input.

A `CheckBox` instance receives focus if a user clicks it or tabs to it. When a `CheckBox` instance has focus, you can use the following keys to control it:

Key	Description
Shift+Tab	Moves focus to the previous element.
Spacebar	Selects or deselects the component and triggers the <code>click</code> event.
Tab	Moves focus to the next element.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each `CheckBox` instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

When you add the `CheckBox` component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.CheckBoxAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the CheckBox component

A check box is a fundamental part of any form or web application. You can use check boxes wherever you need to gather a set of `true` or `false` values that aren't mutually exclusive. For example, a form collecting personal information about a customer could have a list of hobbies for the customer to select; each hobby would have a check box beside it.

CheckBox parameters

You can set the following authoring parameters for each CheckBox component instance in the Property inspector or in the Component inspector:

label sets the value of the text for the check box; the default value is `CheckBox`.

labelPlacement orients the label text for the check box. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`; the default value is `right`. For more information, see [CheckBox.labelPlacement](#).

selected sets the initial value of the check box to checked (`true`) or unchecked (`false`). The default value is `false`.

You can write ActionScript to control these and additional options for the CheckBox component using its properties, methods, and events. For more information, see “[CheckBox class](#)” on page 135.

Creating an application with the CheckBox component

The following procedure explains how to add a CheckBox component to an application while authoring. The following example is a form for an online dating application. The form is a query that searches for possible dating matches for the customer. The query form must have a check box labeled Restrict Age that permits customers to restrict their search to a specified age group. When the Restrict Age check box is selected, the customer can then enter the minimum and maximum ages into two text fields. (These text fields are enabled only when the check box is selected.)

To create an application with the **CheckBox** component:

1. Drag two **TextInput** components from the Components panel to the Stage.
2. In the Property inspector, enter the instance names **minimumAge** and **maximumAge**.
3. Drag a **CheckBox** component from the Components panel to the Stage.
4. In the Property inspector, do the following:
 - Enter **restrictAge** for the instance name.
 - Enter **Restrict Age** for the label parameter.
5. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
var restrictAgeListener:Object = new Object();
restrictAgeListener.click = function (evt:Object) {
    minimumAge.enabled = evt.target.selected;
    maximumAge.enabled = evt.target.selected;
};
restrictAge.addEventListener("click", restrictAgeListener);
```

This code creates a `click` event handler that enables and disables the `minimumAge` and `maximumAge` text field components, which have already been placed on Stage. For more information, see [CheckBox.click](#) and “[TextInput component](#)” on page 1209.

To create a check box using **ActionScript**:

1. Drag the **CheckBox** component from the Components panel to the current document’s library.

This adds the component to the library, but doesn’t make it visible in the application.
2. Drag the **TextInput** component from the Components panel to the current document’s library.
3. In the first frame of the main Timeline, add the following ActionScript to the Actions panel to create and position component instances:

```
this.createClassObject(mx.controls.CheckBox, "testAge_ch", 1,
    {label:'Age Range', selected:true});
this.createClassObject(mx.controls.TextInput, "minimumAge_ti", 2,
    {restrict:[0-9], text:18, maxChars:2});
minimumAge_ti.move(20, 30);
this.createClassObject(mx.controls.TextInput, "maximumAge_ti", 3,
    {restrict:[0-9], text:55, maxChars:2});
maximumAge_ti.move(20, 60);
```

This script uses the method “[UIObject.createClassObject\(\)](#)” on page 1362 to create the **CheckBox** instance named **restrictAge**, and specifies a label property. Then, the code uses the method “[UIObject.move\(\)](#)” on page 1375 to position the button.

4. Now, add the following ActionScript to create an event listener and an event handler function:

```
// Create handler for checkBox event.
function checkBoxHandler(evt_obj:Object) {
    minimumAge_ti.enabled = evt_obj.target.selected;
    maximumAge_ti.enabled = evt_obj.target.selected;
}
// Add Listener.
testAge_ch.addEventListener("click", checkBoxHandler);
```

This code creates a `click` event handler that enables and disables the `minimumAge` and `maximumAge` text field components. For more information, see [CheckBox.click](#), “[EventDispatcher.addEventListener\(\)](#)” on page 501 and “[TextInput component](#)” on page 1209.

Customizing the CheckBox component

You can transform a `CheckBox` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (`UIObject.setSize()`) or any applicable properties and methods of the [CheckBox class](#). Resizing the check box does not change the size of the label or the check box icon; it only changes the size of the bounding box.

The bounding box of a `CheckBox` instance is invisible and also designates the hit area for the instance. If you increase the size of the instance, you also increase the size of the hit area. If the bounding box is too small to fit the label, the label is clipped to fit.

Using styles with the CheckBox component

You can set style properties to change the appearance of a `CheckBox` instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A CheckBox component supports the following styles:

Style	Theme	Description
themeColor	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
color	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
embedFonts	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".
symbolBackgroundColor	Sample	The background color of the check box. The default value is 0xFFFFFFFF (white).
symbolBackgroundDisabledColor	Sample	The background color of the check box when disabled. The default value is 0xEFEEEE (light gray).
symbolBackgroundPressedColor	Sample	The background color of the check box when pressed. The default value is 0xFFFFFFFF (white).

Style	Theme	Description
symbolColor	Sample	The color of the check mark. The default value is 0x000000 (black).
symbolDisabledColor	Sample	The color of the disabled check mark. The default value is 0x848384 (dark gray).

Using skins with the CheckBox component

The CheckBox component uses symbols in the library to represent the button states. To skin the CheckBox component while authoring, modify symbols in the Library panel. The CheckBox component skins are located in the Flash UI Components 2/Themes/MMDefault/CheckBox Assets/states folder in the library of either the HaloTheme.fla file or the SampleTheme.fla file. For more information, see “About skinning components” in *Using Components*.

A CheckBox component uses the following skin properties:

Property	Description
falseUpSkin	The up (normal) unchecked state. The default is <code>CheckFalseUp</code> .
falseDownSkin	The pressed unchecked state. The default is <code>CheckFalseDown</code> .
falseOverSkin	The over unchecked state. The default is <code>CheckFalseOver</code> .
falseDisabledSkin	The disabled unchecked state. The default is <code>CheckFalseDisabled</code> .
trueUpSkin	The toggled checked state. The default is <code>CheckTrueUp</code> .
trueDownSkin	The pressed checked state. The default is <code>CheckTrueDown</code> .
trueOverSkin	The over checked state. The default is <code>CheckTrueOver</code> .
trueDisabledSkin	The disabled checked state. The default is <code>CheckTrueDisabled</code> .

Each of these skins corresponds to the icon indicating the CheckBox state. The CheckBox component does not have a border or background.

To create movie clip symbols for CheckBox skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the CheckBox Assets folder to the library for your document.

4. Expand the CheckBox Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the CheckFalseDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white square to a light gray.
7. Repeat steps 5-6 for all symbols you want to customize.
For example, repeat the color change for the inner box of the CheckTrueDisabled symbol.
8. Click the Back button to return to the main timeline.
9. Drag a CheckBox component to the Stage.
For this example, drag two instances to show the two new skin symbols.
10. Set the CheckBox instance properties as desired.
For this example, set one CheckBox instance to `true`, and use ActionScript to set both CheckBox instances to disabled.
11. Select Control > Test Movie.

CheckBox class

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [SimpleButton class](#) > [Button component](#) > [CheckBox](#)

ActionScript Class Name `mx.controls.CheckBox`

The properties of the CheckBox class let you create a text label and position it to the left, right, top, or bottom of a check box at runtime.

Setting a property of the CheckBox class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The CheckBox component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in *Using Components*.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.CheckBox.version);
```

NOTE

The code `trace(myCheckBoxInstance.version);` returns `undefined`.

Method summary for the CheckBox class

There are no methods exclusive to the CheckBox class.

Methods inherited from the UIObject class

The following table lists the methods the CheckBox class inherits from the UIObject class. When calling these methods from the CheckBox object, use the form *checkBoxInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the CheckBox class inherits from the UIComponent class. When calling these methods from the CheckBox object, use the form *checkBoxInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the CheckBox class

The following table lists properties of the CheckBox class.

Property	Description
<code>CheckBox.label</code>	Specifies the text that appears next to a check box.
<code>CheckBox.labelPlacement</code>	Specifies the orientation of the label text in relation to a check box.
<code>CheckBox.selected</code>	Specifies whether the check box is selected (<code>true</code>) or deselected (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the CheckBox class inherits from the UIObject class.

When accessing these properties from the CheckBox object, use the form

checkBoxInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the CheckBox class inherits from the UIComponent class. When accessing these properties from the CheckBox object, use the form *checkBoxInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the SimpleButton class

The following table lists the properties the CheckBox class inherits from the SimpleButton class. When accessing these properties from the CheckBox object, use the form *checkBoxInstance.propertyName*.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the Button class

The following table lists the properties the CheckBox class inherits from the Button class. When accessing these properties from the CheckBox object, use the form *checkBoxInstance.propertyName*.

Property	Description
<code>Button.label</code>	Specifies the text that appears in a button.
<code>Button.labelPlacement</code>	Specifies the orientation of the label text in relation to an icon.

Event summary for the CheckBox class

The following table lists an event of the CheckBox class.

Event	Description
<code>CheckBox.click</code>	Triggered when the mouse is clicked (released) over the check box, or if the check box has focus and the Spacebar is pressed.

Events inherited from the UIObject class

The following table lists the events the CheckBox class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the CheckBox class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the SimpleButton class

The following table lists the event the CheckBox class inherits from the SimpleButton class.

Event	Description
SimpleButton.click	Broadcast when a button is clicked.

CheckBox.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObject:Object) {
    // ...
};
checkBoxInstance.addEventListener("click", listenerObject);
```

Usage 2:

```
on (click) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the check box, or if the check box has focus and the Spacebar is pressed.

The first usage example uses a dispatcher-listener event model. A component instance (*checkboxInstance*) dispatches an event (in this case, `click`), and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method (see [EventDispatcher.addEventListener\(\)](#)) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `CheckBox` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the check box `myCheckBox`, sends `“_level0.myCheckBox”` to the Output panel:

```
on (click) {
    trace(this);
}
```

Example

The following example enables a button when the check box is selected. This example assumes you have a `Button` component instance on the Stage with instance name `submit_button`, and a `CheckBox` component instance on the Stage with the instance name `agree_ch`. Add the following code to the first frame of the main timeline:

```
agree_ch.label = "I agree";
submit_button.enabled = false;

// Create Listener Object.
var form_obj:Object = new Object();

// Assign function to Listener Object.
form_obj.click = function(event_obj:Object) {
    submit_button.enabled = event_obj.target.selected;
};

// Add Listener.
agree_ch.addEventListener("click", form_obj);
```

The following code sends a message to the Output panel when `checkBoxInstance` is clicked. The `on()` handler must be attached directly to `checkBoxInstance`:

```
on (click) {  
    trace("check box component was clicked");  
}
```

See also

[EventDispatcher.addEventListener\(\)](#)

CheckBox.label

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`checkBoxInstance.label`

Description

Property; indicates the text label for the check box. By default, the label appears to the right of the check box. Setting this property overrides the label parameter specified in the Parameters tab of the Component inspector.

The `CheckBox` component does not allow multiline labels.

Example

The following code sets the text that appears beside the `CheckBox` component and sends the value to the Output panel:

```
checkBox.label = "Remove from list";  
trace(checkBox.label)
```


This example creates the check box in ActionScript, and then resizes the label when checked. For this example, drag a CheckBox component from the Components panel to the current document's library (so the CheckBox component appears in your library, but not on the Stage). Then add the following ActionScript to the first frame of the main timeline:

```
this.createClassObject(mx.controls.CheckBox, "my_ch", 10, {label:"Resize
    CheckBox instance"});

function checkboxHandler(evt_obj:Object):Void {
    trace("before: " + evt_obj.target.width + "px wide");
    evt_obj.target.setSize(200, evt_obj.target.height);
    trace("after: " + evt_obj.target.width + "px wide");
}
my_ch.addEventListener("click", checkboxHandler);
```

See also

[CheckBox.labelPlacement](#)

CheckBox.labelPlacement

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

checkboxInstance.labelPlacement

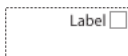
Description

Property; a string that indicates the position of the label in relation to the check box. The following are the four possible values (the dotted lines represent the bounding area of the component; they are invisible in a document):

- "right" The check box is pinned to the upper-left corner of the bounding area. The label is set to the right of the check box. This is the default value.



- "left" The check box is pinned to the upper-right corner of the bounding area. The label is set to the left of the check box.



- "bottom" The label is set below the check box. The check box and label are centered horizontally and vertically.



- "top" The label is placed below the check box. The check box and label are centered horizontally and vertically.



You can change the bounding area of a component while authoring by using the Transform command or at runtime using the `UIObject.setSize()` property. For more information, see [“Customizing the CheckBox component” on page 132](#).

Example

The following example sets the placement of the label to the left of the check box:

```
checkBox_mc.labelPlacement = "left";
```

The following example uses `ActionScript` to create check box instances. The check box instance `right_ch` has its label and `labelPlacement` properties set within the method [“`UIObject.createClassObject\(\)`” on page 1362](#). The check box instance `left_ch` has its label and `labelPlacement` properties set in separate declarations. Drag the `CheckBox` component from the Components panel to the current document’s library (so the component appears in your library, but not on the Stage). Then add the following `ActionScript` to the first frame of the main timeline:

```
this.createClassObject(mx.controls.CheckBox, "right_ch", 1, {label:"Right",
    labelPlacement:"right"});
right_ch.move(10, 10);
this.createClassObject(mx.controls.CheckBox, "left_ch", 2);
left_ch.label= "Left";
left_ch.labelPlacement = "left";
left_ch.move(10, 30);
```

See also

[CheckBox.label](#)

CheckBox.selected

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

checkboxInstance.selected

Description

Property; a Boolean value that selects (*true*) or deselects (*false*) the check box.

Example

The following example shows a check box that has its *selected* property set to *true*, by default, and then uses the *selected* property within an event handler function to respond to the user clicking the check box. Drag the *CheckBox* component to the Stage. Give the component instance the name *my_ch*. Then, add the following code to the Actions panel of the first frame of the main timeline:

```
my_ch.selected = true;

var checkboxListener:Object = new Object();
checkboxListener.click = function(evt_obj:Object) {
    if (evt_obj.target.selected) {
        evt_obj.target.label = "Selected!";
    } else {
        evt_obj.target.label = "Unselected!";
    }
};
my_ch.addEventListener("click", checkboxListener);
```


Collection interface (Flash Professional only)

The collection class is distributed in the common classes library as a compiled clip symbol. To access this class, select Window > Common Libraries > Classes, which contains the compiled clip `UtilsClasses`.

Collection class (Flash Professional only)

ActionScript Class Name `mx.utils.Collection`

The collection interface lets you programmatically manage a group of related items, called *collection items*. Each collection item in this set has properties that are described in the metadata of the collection item class definition.

Components can expose properties as collections, which you can manipulate while authoring by using the Values dialog box from the Component inspector. Using this dialog box, you can add items, remove items, change properties of items, and change the position of items within the collection. For more information on collections and collection items, see “About the Collection tag” in *Using Components*.

You typically use the collection interface with components that use the Collection metadata tag to create collection properties. Although you can create, access, and delete Collection instances programmatically, collections are most often used in the context of a component. Flash MX Professional 2004 provides implementations of both collection-related interfaces (`CollectionImpl` for Collection, and `IteratorImpl` for Iterator).

Method summary for the Collection interface

The following table lists the methods of the Collection interface.

Method	Description
<code>Collection.addItem()</code>	Adds a new item to the end of the collection.
<code>Collection.contains()</code>	Indicates whether the collection contains the specified item.
<code>Collection.clear()</code>	Removes all elements from the collection.
<code>Collection.getItemAt()</code>	Returns an item within the collection by using its index.
<code>Collection.getIterator()</code>	Returns an iterator over the elements in the collection.
<code>Collection.getLength()</code>	Returns the number of items in the collection.
<code>Collection.isEmpty()</code>	Indicates whether the collection is empty.
<code>Collection.removeItem()</code>	Removes the specified item from the collection.

Collection.addItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.addItem(item)
```

Parameters

item The object to be added to the collection. If *item* is `null`, it is not added to the collection.

Returns

A Boolean value of `true` if the collection was changed as a result of the operation.

Description

Method; adds a new item to the end of the collection.

Example

The following example calls `addItem()`:

```
on (click) {
import CompactDisc;

    var myColl:mx.utils.Collection;
    myColl = _parent.thisShelf.MyCompactDiscs;
    myCD = new CompactDisc();
    myCD.Artist = "John Coltrane";
    myCD.Title = "Giant Steps";

    var wasAdded:Boolean = myColl.addItem(myCD);
}
```

Collection.contains()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

collection.contains(item)

Parameters

item The object whose presence in the collection is to be tested.

Returns

A Boolean value of `true` if the collection contains *item*.

Description

Method; indicates whether the collection contains the specified item. For Flash to consider the objects as equal, they must refer to the same object. If *item* is a different object, `Collection.contains()` returns `false`, even if the object's properties are all equal.

Example

The following example calls `contains()`:

```
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;

var itr:mx.utils.Iterator = myColl.getIterator();
while (itr.hasNext()) {
    var cd:CompactDisc = CompactDisc(itr.next());
    var title:String = cd.Title;
    var artist:String = cd.Artist;

    if(myColl.contains(cd)) {
        trace("myColl contains " + title);
    }
    else {
        trace("myColl does not contain " + title);
    }
}
```

Collection.clear()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.clear()
```

Returns

Nothing.

Description

Method; removes all of the elements from the collection.

Example

The following example calls `clear()`:

```
on (click) {
    var myColl:mx.utils.Collection;
    myColl = _parent.thisShelf.MyCompactDiscs;
    myColl.clear();
}
```


Collection.getItemAt()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.getItemAt(index)
```

Parameters

index A number that indicates the location of *item* within the collection. This is a zero-based index, so 0 retrieves the first item, 1 retrieves the second item, and so on.

Returns

An object containing a reference to the specified collection item, or `null` if *index* is out of bounds.

Description

Method; returns an item within the collection by using its index.

Example

The following example calls `getItemAt()`:

```
//...
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;
var myCD = CompactDisc(myColl.getItemAt(0));
if (myCD !=null) {
    trace("Retrieved " + myCD.Title);
}
//...
```

Collection.iterator()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.iterator()
```

Returns

An Iterator object that you can use to step through the collection.

Description

Method; returns an iterator over the elements in the collection. There are no guarantees concerning the order in which the elements are returned (unless this collection is an instance of a class that provides a guarantee).

Example

The following example calls `iterator()`:

```
on (click) {  
    var myColl:mx.utils.Collection;  
    myColl = _parent.thisShelf.MyCompactDiscs;  
  
    var itr:mx.utils.Iterator = myColl.iterator();  
    while (itr.hasNext()) {  
        var cd:CompactDisk = CompactDisc(itr.next());  
        var title:String = cd.Title;  
        var artist:String = cd.Artist;  
  
        trace("Title: " + title + " - Artist: " + artist);  
    }  
}
```

Collection.getLength()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.getLength()
```

Returns

The number of items in the collection.

Description

Method; returns the number of items in the collection.

Example

The following example calls `getLength()`:

```
//...  
var myColl:mx.utils.Collection;  
myColl = _parent.thisShelf.MyCompactDiscs;  
trace ("Collection size is: " + myColl.getLength());  
//...
```

Collection.isEmpty()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.isEmpty()
```

Returns

A Boolean value of `true` if the collection is empty.

Description

Method; indicates whether the collection is empty.

Example

The following example calls `isEmpty()`:

```
on (click) {
    var myColl:mx.utils.Collection;
    myColl = _parent.thisShelf.MyCompactDiscs;
    if (myColl.isEmpty()) {
        trace("No CDs in the collection");
    }
}
//...
```

Collection.removeItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
collection.removeItem(item)
```

Parameters

item The object to be removed from the collection.

Returns

A Boolean value of `true` if *item* was removed successfully.

Description

Method; removes the specified item from the collection. Because `Collection.removeItem()` dynamically reduces the size of the collection, do not call this method while looping through an iterator.

Example

The following example calls `removeItem()`:

```
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;

// get this from a text input box
var removeArtist:String = _parent.tArtistToRemove.text;
var removeSize:Number = 0;

if (myColl.isEmpty()) {
    trace("No CDs in the collection");
}
else {
    var toRemove:Array = new Array();
    var itr:mx.utils.Iterator = myColl.getIterator();
    var cd:CompactDisc = new CompactDisc();
    var title:String = "";
    var artist:String = "";
    while (itr.hasNext()) {
        cd = CompactDisc(itr.next());
        title = cd.Title;
        artist = cd.Artist;
        if(artist == removeArtist) {
            // mark this artist for deletion
            removeSize = toRemove.push(cd);
            trace("*** Marked for deletion: " + artist + "|" + title);
        }
    }
    // after while loop, remove the bad ones
    var removeCD:CompactDisc = new CompactDisc();
    for(i = 0; i < removeSize; i++) {
        removeCD = toRemove[i];
        trace("Removing: " + removeCD.Artist + "|" + removeCD.Title);
        myColl.removeItem(removeCD);
    }
}
```


A combo box allows a user to make a single selection from a pop-up list. A combo box can be static or editable. An editable combo box allows a user to enter text directly into a text field at the top of the list, as well as selecting an item from a pop-up list. If the pop-up list hits the bottom of the document, it opens up instead of down. The combo box is composed of three subcomponents: a Button component, a TextInput component, and a List component.

When a selection is made in the list, the label of the selection is copied to the text field at the top of the combo box. It doesn't matter if the selection is made with the mouse or the keyboard.

A ComboBox component receives focus if you click the text box or the button. When a ComboBox component has focus and is editable, all keystrokes go to the text box and are handled according to the rules of the TextInput component (see [“TextInput component” on page 1209](#)), with the exception of the following keys:

Key	Description
Control+Down Arrow	Opens the drop-down list and gives it focus.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

When a ComboBox component has focus and is static, alphanumeric keystrokes move the selection up and down the drop-down list to the next item with the same first character. You can also use the following keys to control a static combo box:

Key	Description
Control+Down Arrow	Opens the drop-down list and gives it focus.
Control+Up Arrow	Closes the drop-down list, if open in the stand-alone and browser versions of Flash Player.
Down Arrow	Moves the selection down one item.

Key	Description
End	Selection moves to the bottom of the list.
Escape	Closes the drop-down list and returns focus to the combo box in test mode.
Enter	Closes the drop-down list and returns focus to the combo box.
Home	Moves the selection to the top of the list.
Page Down	Moves the selection down one page.
Page Up	Moves the selection up one page.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

When the drop-down list of a combo box has focus, alphanumeric keystrokes move the selection up and down the drop-down list to the next item with the same first character. You can also use the following keys to control a drop-down list:

Key	Description
Control+Up Arrow	If the drop-down list is open, focus returns to the text box and the drop-down list closes in the stand-alone and browser versions of Flash Player.
Down Arrow	Moves the selection down one item.
End	Moves the insertion point to the end of the text box.
Enter	If the drop-down list is open, focus returns to the text box and the drop-down list closes.
Escape	If the drop-down list is open, focus returns to the text box and the drop-down list closes in test mode.
Home	Moves the insertion point to the beginning of the text box.
Page Down	Moves the selection down one page.
Page Up	Moves the selection up one page.
Tab	Moves focus to the next object.
Shift+End	Selects the text from the insertion point to the End position.
Shift+Home	Selects the text from the insertion point to the Home position.
Shift+Tab	Moves focus to the previous object.
Up Arrow	Moves the selection up one item.

NOTE

The page size used by the Page Up and Page Down keys is one less than the number of items that fit in the display. For example, paging down through a ten-line drop-down list will show items 0-9, 9-18, 18-27, and so on, with one item overlapping per page.

For more information about controlling focus, see “[FocusManager class](#)” on page 721 or “Creating custom focus navigation” in *Using Components*.

A live preview of each ComboBox component instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, the drop-down list does not open in the live preview, and the first item is displayed as the selected item.

When you add the ComboBox component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.ComboBoxAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the ComboBox component

You can use a ComboBox component in any form or application that requires a single choice from a list. For example, you could provide a drop-down list of states in a customer address form. You can use an editable combo box for more complex scenarios. For example, in an application that provides driving directions, you could use an editable combo box for a user to enter her origin and destination addresses. The drop-down list would contain her previously entered addresses.

ComboBox parameters

You can set the following authoring parameters for each ComboBox component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

data associates a data value with each item in the ComboBox component. The data parameter is an array.

editable determines if the ComboBox component is editable (`true`) or only selectable (`false`). The default value is `false`.

labels populates the ComboBox component with an array of text values.

rowCount sets the maximum number of items that can be displayed in the list. The default value is 5.

You can set the following additional parameters for each ComboBox component instance in the Component inspector (Window > Component Inspector):

restrict indicates the set of characters that a user can enter in the text field of a combo box. The default value is `undefined`. See [“ComboBox.restrict” on page 194](#).

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to set additional options for ComboBox instances using the methods, properties, and events of the ComboBox class. For more information, see [“ComboBox class” on page 165](#).

Creating an application with the ComboBox component

The following procedure explains how to add a ComboBox component to an application while authoring. In this example, the combo box presents a list of cities to select from in its pop-up list.

To create an application with the ComboBox component:

1. Drag a ComboBox component from the Components panel to the Stage.
2. Select the Transform tool and resize the component on the Stage.

The combo box can only be resized on the Stage during authoring. Typically, you would only change the width of a combo box to fit its entries.
3. Select the combo box and, in the Property inspector, enter the instance name **comboBox**.
4. In the Component inspector or Property inspector, do the following:
 - Enter **Minneapolis**, **Portland**, and **Keene** for the label parameter. Double-click the label parameter field to open the Values dialog box. Then click the plus sign to add items.
 - Enter **MN.swf**, **OR.swf**, and **NH.swf** for the data parameter.

These are imaginary SWF files that, for example, you could load when a user selects a city from the combo box.

5. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
function change(evt){
    trace(evt.target.selectedItem.label);
}
comboBox.addEventListener("change", this);
```

The last line of code adds a `change` event handler to the `ComboBox` instance. For more information, see [ComboBox.change](#).

To create a `ComboBox` component using `ActionScript`:

1. Drag the `ComboBox` component from the Components panel to the current document's library.

This adds the component to the library, but doesn't make it visible in the application.

2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.ComboBox, "my_cb", 10);
```

```
my_cb.addItem({data:1, label:"One"});
my_cb.addItem({data:2, label:"Two"});
```

This script uses the method “[UIObject.createClassObject\(\)](#)” on page 1362 to create the `ComboBox` instance, and then uses “[ComboBox.addItem\(\)](#)” on page 171 to add list items to the `ComboBox`.

3. Now add an event listener and event handler function to respond when a `ComboBox` item is selected:

```
// Create listener object.
var cbListener:Object = new Object();
// Create event handler function.
cbListener.change = function (evt_obj:Object) {
    trace("Currently selected item is: " +
        evt_obj.target.selectedItem.label);
}
// Add event listener.
my_cb.addEventListener("change", cbListener);
```

4. Select `Control >Test Movie`, and click an item in the combo box to see a message in the Output panel.

Customizing the ComboBox component

You can transform a ComboBox component horizontally and vertically while authoring. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands.

If text is too long to fit in the combo box, the text is clipped to fit. You must resize the combo box while authoring to fit the label text.

In editable combo boxes, only the button is the hit area—not the text box. For static combo boxes, the button and the text box constitute the hit area. The hit area responds by opening or closing the drop-down list.

Using styles with the ComboBox component

You can set style properties to change the appearance of a ComboBox component. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

The combo box has two unique styles: `openDuration` and `openEasing`. Other styles are passed to the button, text box, and drop-down list of the combo box through those individual components, as follows:

- The button is a Button instance and uses its styles. (See “Using styles with the Button component” on page 94.)
- The text is a TextInput instance and uses its styles. (See “Using styles with the TextInput component” on page 1212.)
- The drop-down list is an List instance and uses its styles. (See “Using styles with the List component” on page 766.)

A ComboBox component uses the following styles:

Style	Theme	Description
themeColor	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
backgroundColor	Both	The background color. The default color is white.
borderStyle	Both	The Button subcomponent uses two RectBorder instances for its borders and responds to the styles defined on that class. See "RectBorder class" on page 1063 . In the Halo theme, the ComboBox component uses a custom rounded border for the collapsed portion of the ComboBox. The colors of this portion of the ComboBox can be modified only through skinning. See "Using skins with the ComboBox component" on page 164 .
color	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
embedFonts	Both	Boolean value that indicates whether the font specified in fontFamily is an embedded font. This style must be set to true if fontFamily refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to true and fontFamily does not refer to an embedded font, no text is displayed. The default value is false.
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a setStyle() call, but subsequent calls to getStyle() return "none".
textAlign	Both	The text alignment: either "left", "right", or "center". The default value is "left".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".

Style	Theme	Description
<code>openDuration</code>	Both	The duration, in milliseconds, of the transition animation. The default value is 250.
<code>openEasing</code>	Both	A reference to a tweening function that controls the animation. Defaults to sine in/out. For more information, see “Customizing component animations” in <i>Using Components</i> .

The following example demonstrates how to use List styles to control the behavior of the pop-up portion of a `ComboBox` component.

```
// comboBox is an instance of the ComboBox component on Stage.
comboBox.setStyle("alternatingRowColors", [0xFFFFFFFF, 0xBFBBFB]);
```

Using skins with the `ComboBox` component

The `ComboBox` component uses symbols in the library to represent the button states and has skin variables for the down arrow. These skins are located in the Flash UI Components 2/Themes/MMDefault/ComboBox Assets/States folder of the `HaloTheme.fla` and `SampleTheme.fla` files. The information below describes these skins and provides steps for customizing them.

The `ComboBox` component also uses scroll bar skins for the drop-down list’s scroll bar and two `RectBorder` class instances for the border around the text input and drop-down list. For information on customizing these skins, see “Using skins with the `UIScrollBar` component” on page 1394 and “`RectBorder` class” on page 1063. For more information on the methods available to skin components, see “About skinning components” in *Using Components*.

A `ComboBox` component uses the following skin properties:

Property	Description
<code>ComboDownArrowDisabledName</code>	The down arrow’s disabled state. The default is <code>ComboDownArrowDisabled</code> .
<code>ComboDownArrowDownName</code>	The down arrow’s down state. The default is <code>ComboDownArrowDown</code> .
<code>ComboDownArrowUpName</code>	The down arrow’s up state. The default is <code>ComboDownArrowOver</code> .
<code>ComboDownArrowOverName</code>	The down arrow’s over state. The default is <code>ComboDownArrowUp</code> .

To create movie clip symbols for ComboBox skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the ComboBox Assets folder to the library for your document.
4. Expand the ComboBox Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the ComboDownArrowDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white square to a light gray.
7. Repeat steps 5-6 for all symbols you want to customize.
8. Click the Back button to return to the main timeline.
9. Drag a ComboBox component to the Stage.
10. Set the ComboBox instance properties as desired.
For this example, use ActionScript to set the ComboBox to disabled.
11. Select Control > Test Movie.

ComboBox class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `ComboBase` > `ComboBox`

ActionScript Class Name `mx.controls.ComboBox`

The `ComboBox` component combines three separate subcomponents: `Button`, `TextInput`, and `List`. Most of the methods, properties, and events of each subcomponent are available directly from the `ComboBox` component and are listed in the summary tables for the `ComboBox` class.

The drop-down list in a combo box is provided either as an array or as a data provider. If you use a data provider, the list changes at runtime. You can change the source of the `ComboBox` data dynamically by switching to a new array or data provider.

Items in a combo box list are indexed by position, starting with the number 0. An item can be one of the following:

- A primitive data type.
- An object that contains a `label` property and a `data` property

NOTE

An object may use the `ComboBox.labelFunction` or `ComboBox.labelField` property to determine the `label` property.

If the item is a primitive data type other than `String`, it is converted to a string. If an item is an object, the `label` property must be a string and the `data` property can be any `ActionScript` value.

`ComboBox` methods to which you supply items have two parameters, `label` and `data`, that refer to the properties above. Methods that return an item return it as an object.

A combo box defers the instantiation of its drop-down list until a user interacts with it. Therefore, a combo box may appear to respond slowly on first use.

Use the following code to programmatically access the `ComboBox` component's drop-down list and override the delay:

```
var foo = myComboBox.dropdown;
```

Accessing the pop-up list may cause a pause in the application. This may occur when the user first interacts with the combo box, or when the above code runs.

Method summary for the `ComboBox` class

The following table lists methods of the `ComboBox` class.

Method	Description
<code>ComboBox.addItem()</code>	Adds an item to the end of the list.
<code>ComboBox.addItemAt()</code>	Adds an item to the end of the list at the specified index.
<code>ComboBox.close()</code>	Closes the drop-down list.
<code>ComboBox.getItemAt()</code>	Returns the item at the specified index.
<code>ComboBox.open()</code>	Opens the drop-down list.
<code>ComboBox.removeAll()</code>	Removes all items in the list.
<code>ComboBox.removeItemAt()</code>	Removes an item from the list at the specified location.
<code>ComboBox.replaceItemAt()</code>	Replaces the content of the item at the specified index.
<code>ComboBox.sortItems()</code>	Sorts the list using a compare function.
<code>ComboBox.sortItemsBy()</code>	Sorts the list using a field of each item.

Methods inherited from the UIObject class

The following table lists the methods the ComboBox class inherits from the UIObject class. When calling these methods from the ComboBox object, use the form *comboBoxInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the ComboBox class inherits from the UIComponent class. When calling these methods from the ComboBox object, use the form *comboBoxInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the ComboBox class

The following table lists properties of the ComboBox class.

Property	Description
<code>ComboBox.dataProvider</code>	The data model for the items in the list.
<code>ComboBox.dropdown</code>	Returns a reference to the List component contained by the combo box.
<code>ComboBox.dropdownWidth</code>	The width of the drop-down list, in pixels.
<code>ComboBox.editable</code>	Indicates whether a combo box is editable.
<code>ComboBox.labelField</code>	Indicates which data field to use as the label for the drop-down list.
<code>ComboBox.labelFunction</code>	Specifies a function to compute the label field for the drop-down list.
<code>ComboBox.length</code>	Read-only; the length of the drop-down list.
<code>ComboBox.restrict</code>	The set of characters that a user can enter in the text field of a combo box.
<code>ComboBox.rowCount</code>	The maximum number of list items to display at one time.
<code>ComboBox.selectedIndex</code>	The index of the selected item in the drop-down list.
<code>ComboBox.selectedItem</code>	The value of the selected item in the drop-down list.
<code>ComboBox.text</code>	The string of text in the text box.
<code>ComboBox.textField</code>	A reference to the TextInput component in the combo box.
<code>ComboBox.value</code>	The value of the text box (editable) or drop-down list (static).

Properties inherited from the UIObject class

The following table lists the properties the ComboBox class inherits from the UIObject class. When accessing these properties from the ComboBox object, use the form *comboBoxInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only. The position of the right edge of the object, relative to the right edge of its parent.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the `ComboBox` class inherits from the `UIComponent` class. When accessing these properties from the `ComboBox` object, use the form `comboBoxInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the ComboBox class

The following table lists events of the `ComboBox` class.

Event	Description
<code>ComboBox.change</code>	Broadcast when the value of the combo box changes as a result of user interaction.
<code>ComboBox.close</code>	Broadcast when the list of the combo box begins to retract.
<code>ComboBox.enter</code>	Broadcast when the Enter key is pressed.
<code>ComboBox.itemRollOut</code>	Broadcast when the pointer rolls off a pop-up list item.
<code>ComboBox.itemRollOver</code>	Broadcast when a drop-down list item is rolled over.

Event	Description
<code>ComboBox.open</code>	Broadcast when the drop-down list begins to open.
<code>ComboBox.scroll</code>	Broadcast when the drop-down list is scrolled.

Events inherited from the UIObject class

The following table lists the events the ComboBox class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the ComboBox class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

ComboBox.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.addItem(label[, data])  
comboBoxInstance.addItem({label:label[, data:data]})  
comboBoxInstance.addItem(obj);
```

Parameters

label A string that indicates the label for the new item.

data The data for the item; it can be of any data type. This parameter is optional.

obj An object with a `label` property and an optional `data` property.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list.

Example

With a `ComboBox` component instance named `my_cb`, add the following ActionScript to the Actions panel for the first frame of the main timeline. This ActionScript creates a `ComboBox` with three items; each has a data value and a label string. When you test the SWF file, and click one of the items, the Output panel displays the identity of the “target” the data value and the label:

```
// Add Items to Combo Box.  
my_cb.addItem("this is an Item");  
my_cb.addItem({data:2, label:"second value"});  
my_cb.addItem({data:3, label:"third value"});  
  
// Add event listener and event handler function.  
var cbListener:Object = new Object();  
cbListener.change = function(evt_obj:Object):Void {  
    var currentlySelected:Object = evt_obj.target.selectedItem;  
    trace(evt_obj.target);  
    trace("data: "+currentlySelected.data);  
    trace("label: "+currentlySelected.label);  
};  
my_cb.addEventListener("change", cbListener);
```

ComboBox.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.addItemAt(index, label[, data])
comboBoxInstance.addItemAt(index, {label:label[, data:data]})
comboBoxInstance.addItemAt(index, obj);
```

Parameters

index A number 0 or greater that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item; it can be of any data type. This parameter is optional.

obj An object with *label* and *data* properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list at the index specified by the *index* parameter. Indices greater than `ComboBox.length` are ignored.

Example

Start with a `ComboBox` component instance named `my_cb`, and a `Button` component instance named `my_btn`. Add the following ActionScript to the Actions panel for the first frame of the main timeline. When you test the SWF file, click the combo box to see two items in it. Then click the button, and the next time you click the combo box, you'll see that it added another item labeled "first value":

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
    my_cb.addItemAt(0, {data:1, label:"first value"});
};
my_btn.addEventListener("click", btnListener);
```

ComboBox.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when the `ComboBox.selectedIndex` or `ComboBox.selectedItem` property changes as a result of user interaction.

Using a dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see [EventDispatcher.addEventListener\(\)](#)) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

With a `ComboBox` component instance `my_cb` on the Stage, the following example sends the instance name of the component that generated the `change` event to the Output panel:

```
// Add Item to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create Listener Object.
var cbListener:Object = new Object();
```

```
// Assign function to Listener Object.
cbListener.change = function(event_obj:Object) {
    trace("Value changed to: "+event_obj.target.selectedItem.label);
};

// Add Listener.
my_cb.addEventListener("change", cbListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

ComboBox.close()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.close()

Parameters

None.

Returns

Nothing.

Description

Method; closes the drop-down list.

Example

With a ComboBox component instance `my_cb` on the Stage, and a Button component instance `my_button`, the following example closes the drop-down list of the `my_cb` combo box when the `my_button` button is clicked:

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
    my_cb.close();
};
my_button.addEventListener("click", btnListener);
```


See also

[ComboBox.open\(\)](#)

ComboBox.close

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("close", listenerObject)
```

Description

Event; broadcast to all registered listeners when the drop-down list of the combo box is fully retracted.

Using a dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, `close`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

With a ComboBox component instance `my_cb` on the Stage, the following example sends a message to the Output panel when the drop-down list opens or closes:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create Listener Object.
var cbListener:Object = new Object();
cbListener.open = function(evt_obj:Object) {
    trace("The ComboBox has opened.");
}
cbListener.close = function(evt_obj:Object){
    trace("The ComboBox has closed.");
}

// Add Listener.
my_cb.addEventListener("open", cbListener);
my_cb.addEventListener("close", cbListener);

// Open the combo box.
my_cb.open();
```

See also

[EventDispatcher.addEventListener\(\)](#)

ComboBox.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.dataProvider

Description

Property; the data model for items viewed in a list. The value of this property can be an array or any object that implements the DataProvider API. The default value is []. The List component and the ComboBox component share the `dataProvider` property, and changes to this property are immediately available to both components.

The List component, like other data-aware components, adds methods to the Array object's prototype so that they conform to the DataProvider API (see `DataProvider.as` for details). Therefore, any array that exists at the same time as a list automatically has all the methods (`addItem()`, `getItemAt()`, and so on) needed for it to be the model of a list, and can be used to broadcast model changes to multiple components.

If the array contains objects, the `labelField` or `labelFunction` property is accessed to determine what parts of the item to display. The default value is "label", so if such a field exists, it is chosen for display; if not, a comma-separated list of all fields is displayed.

NOTE

If the array contains strings at each index, and not objects, the list is not able to sort the items and maintain the selection state. Any sorting causes the selection to be lost.

Any instance that implements the DataProvider API is eligible as a data provider for a List component. This includes Flash Remoting RecordSet objects, Firefly DataSet components, and so on.

Example

This example uses an array of strings to populate the drop-down list for the ComboBox component instance `my_cb`:

```
my_cb.dataProvider = [{data:1, label:"First Item"}, {data:2, label:"Second
  Item"}];
/* is the same as
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
*/
```

This example creates a data provider array and assigns it to the `dataProvider` property:

```
var myDP:Array = new Array();
list.dataProvider = myDP;

for (var i:Number = 0; i < accounts.length; i++) {
    // These changes to the DataProvider will be broadcast to the list.
    myDP.addItem({label: accounts[i].name,
                  data: accounts[i].accountID});
}
```

ComboBox.dropdown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.dropdown

Description

Property (read-only); returns a reference to the list contained by the combo box. The List subcomponent isn't instantiated in the combo box until it needs to be displayed. However, when you access the `dropdown` property, the list is created.

Example

With a ComboBox component instance `my_cb` on the Stage, and two movie clip symbols in the library with Linkage ID values set to `dw_id` and `fl_id`, the following ActionScript uses the `dropdown` property to add icons to each item in the drop-down list:

```
// Set the dropdown width to accommodate the label sizes.
my_cb.dropdownWidth = 200;

// Set the iconField style within the ComboBox's dropdown property.
// The dropdown property is a reference to the List component within the
//   ComboBox
// so we can set List styles for the CB.
my_cb.dropdown.setStyle("iconField", "pIcon");

// Add Items to List.
my_cb.addItem({label:"Dreamweaver 1", pIcon:"dw_id"});
my_cb.addItem({label:"Flash 1", pIcon:"fl_id"});
my_cb.addItem({label:"Flash 2", pIcon:"fl_id"});
```

See also

[ComboBox.dropdownWidth](#)

ComboBox.dropdownWidth

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.dropdownWidth

Description

Property; the width limit of the drop-down list, in pixels. The default value is the width of the ComboBox component (the TextInput instance plus the SimpleButton instance).

Example

With a ComboBox component instance `my_cb` on the Stage, the following ActionScript sets the drop-down list width to accommodate the labels:

```
// Set the dropdown width to accommodate the label sizes.
my_cb.dropdownWidth = 200;

// Add Items to List.
my_cb.addItem("ComboBox");
my_cb.addItem({data:2, label:"This is a long label"});
my_cb.addItem({data:3, label:"This has an even longer label"});
```

See also

[ComboBox.dropdown](#)

ComboBox.editable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.editable

Description

Property; indicates whether the combo box is editable (`true`) or not (`false`). In an editable combo box, a user can enter values into the text box that do not appear in the drop-down list. If a combo box is not editable, you cannot enter text into the text box. The text box displays the text of the item in the list. The default value is `false`.

Making a combo box editable clears the combo box text field. It also sets the selected index (and item) to undefined. To make a combo box editable and still retain the selected item, use the following code:

```
var ix:Number = myComboBox.selectedIndex;
myComboBox.editable = true; // Clears the text field.
myComboBox.selectedIndex = ix; // Copies the label back into the text field.
```

Example

With a `ComboBox` component instance `my_cb` on the Stage, the following `ActionScript` creates a combo box list and two listeners. The first listener handles clicking the “Add new item” label to make the combo box field editable. The second listener handles the user pressing the Enter key to add their entry to the combo box list:

```
// Add items to the combo box list.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:-1, label:"Add new item..."});

// Respond to the user clicking "Add new item".
function changeListener(evt_obj:Object) {
    if (evt_obj.target.selectedItem.data == -1) {
        evt_obj.target.editable = true;
    } else if (evt_obj.target.selectedIndex != undefined) {
        evt_obj.target.editable = false;
        evt_obj.target.setFocus();
    }
}
my_cb.addEventListener("change", changeListener);

// Respond to the user pressing the Enter key after adding a new item name.
function enterListener(evt_obj:Object) {
    if (evt_obj.target.value != '') {
        evt_obj.target.addItem({data:'', label:evt_obj.target.value});
    }
    evt_obj.target.editable = false;
    evt_obj.target.selectedIndex = evt_obj.target.dataProvider.length-1;
    evt_obj.target.setFocus();
}
my_cb.addEventListener("enter", enterListener);
```

ComboBox.enter

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.enter = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("enter", listenerObject)
```

Description

Event; broadcast to all registered listeners when the user presses the Enter key in the text box. This event is a `TextInput` event that is broadcast only from editable combo boxes. For more information, see [TextInput.enter](#).

Using a dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, *enter*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

With a `ComboBox` component instance `my_cb` on the Stage, the following ActionScript creates a combo box list and two listeners. The first listener handles clicking the “Add new item” label to make the combo box field editable. The second listener handles the user pressing the Enter key to add their entry to the combo box list:

```
// Add items to the combo box list.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:-1, label:"Add new item..."});

// Respond to the user clicking "Add new item".
function changeListener(evt_obj:Object) {
    if (evt_obj.target.selectedItem.data == -1) {
        evt_obj.target.editable = true;
    } else if (evt_obj.target.selectedIndex != undefined) {
        evt_obj.target.editable = false;
        evt_obj.target.setFocus();
    }
}
my_cb.addEventListener("change", changeListener);

// Respond to the user pressing the Enter key after adding a new item name.
function enterListener(evt_obj:Object) {
    if (evt_obj.target.value != '') {
        evt_obj.target.addItem({data:'', label:evt_obj.target.value});
    }
    evt_obj.target.editable = false;
    evt_obj.target.selectedIndex = evt_obj.target.dataProvider.length-1;
    evt_obj.target.setFocus();
}
my_cb.addEventListener("enter", enterListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

ComboBox.getItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.getItemAt(*index*)

Parameters

index The index of the item to retrieve. The index must be a number greater than or equal to 0, and less than the value of `ComboBox.length`.

Returns

The indexed item object or value. The value is undefined if the index is out of range.

Description

Method; retrieves the item at a specified index.

Example

With a `ComboBox` component instance `my_cb` on the Stage, the following `ActionScript` displays the label for the first combo box item in the Output panel:

```
//Add Item to List.  
my_cb.addItem({data:1, label:"First Item"});  
my_cb.addItem({data:2, label:"Second Item"});  
  
trace(my_cb.getItemAt(1).label);
```

ComboBox.itemRollOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();  
listenerObject.itemRollOut = function(eventObject:Object) {  
    // Your code here.  
};  
comboBoxInstance.addEventListener("itemRollOut", listenerObject)
```

Event object

In addition to the standard properties of the event object, the `itemRollOut` event has an `index` property. The index is the number of the item that the pointer rolled off.

Description

Event; broadcast to all registered listeners when the pointer rolls off pop-up list items. This is a `List` event that is broadcast from a combo box. For more information, see

[List.itemRollOut](#).

Using a dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, *itemRollOut*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

With a `ComboBox` instance `my_cb` on the Stage, the following ActionScript sends a message to the Output panel that indicates the item index and the event when the pointer rolls on or off an item:

```
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create Listener Object.
var cbListener:Object = new Object();
cbListener.itemRollOver = function(evt_obj:Object) {
    trace("index: "+evt_obj.index+", event: "+evt_obj.type);
};
cbListener.itemRollOut = function(evt_obj:Object) {
    trace("index: "+evt_obj.index+", event: "+evt_obj.type);
};

// Add Listener.
my_cb.addEventListener("itemRollOver", cbListener);
my_cb.addEventListener("itemRollOut", cbListener);
```

See also

[ComboBox.itemRollOver](#), [EventDispatcher.addEventListener\(\)](#)

ComboBox.itemRollOver

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.itemRollOver = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("itemRollOver", listenerObject)
```

Event object

In addition to the standard properties of the event object, the `itemRollOver` event has an `index` property. The index is the number of the item that the pointer rolled over.

Description

Event; broadcast to all registered listeners when the pointer rolls over pop-up list items. This is a List event that is broadcast from a combo box. For more information, see

[List.itemRollOver](#).

Using the dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, `itemRollOver`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “[EventDispatcher class](#)” on page 499.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

With a `ComboBox` instance `my_cb` on the Stage, the following `ActionScript` sends a message to the Output panel that indicates the item index and the event when the pointer rolls on or off an item:

```
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create Listener Object.
var cbListener:Object = new Object();
cbListener.itemRollOver = function(evt_obj:Object) {
    trace("index: " + evt_obj.index + ", event: " + evt_obj.type);
};
cbListener.itemRollOut = function(evt_obj:Object) {
    trace("index: " + evt_obj.index + ", event: " + evt_obj.type);
};

// Add Listener.
my_cb.addEventListener("itemRollOver", cbListener);
my_cb.addEventListener("itemRollOut", cbListener);
```

See also

[ComboBox.itemRollOut](#), [EventDispatcher.addEventListener\(\)](#)

ComboBox.labelField

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.labelField

Description

Property; the name of the field in `dataProvider` array objects to use as the label field. This is a property of the `List` component that is available from a `ComboBox` component instance. For more information, see [List.labelField](#).

The default value is `undefined`.

Example

The following example sets the `dataProvider` property to an array of strings and sets the `labelField` property to indicate that the `name` field should be used as the label for the drop-down list:

```
my_cb.dataProvider = [
    {name:"Gary", gender:"male"},
    {name:"Susan", gender:"female"} ];

my_cb.labelField = "name";
```

See also

[List.labelFunction](#)

ComboBox.labelFunction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.labelFunction

Description

Property; a function that computes the label of a data provider item. You must define the function. The default value is `undefined`.

Example

The following example creates a data provider and then defines a function to specify what to use as the label in the drop-down list:

```
myComboBox.dataProvider = [
    {firstName:"Nigel", lastName:"Pegg", age:"really young"},
    {firstName:"Gary", lastName:"Grossman", age:"young"},
    {firstName:"Chris", lastName:"Walcott", age:"old"},
    {firstName:"Greg", lastName:"Yachuk", age:"really old"} ];

myComboBox.labelFunction = function(itemObj){
    return (itemObj.lastName + ", " + itemObj.firstName);
}
```

See also

[List.labelField](#)

ComboBox.length

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.length

Description

Property (read-only); the length of the drop-down list. This is a property of the List component that is available from a ComboBox instance. For more information, see [List.length](#). The default value is 0.

Example

The following example stores the value of `length` to a variable:

```
var dropdownItemCount:Number = myComboBox.length;
```

ComboBox.open()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.open()

Parameters

None.

Returns

Nothing.

Description

Method; opens the drop-down list.

Example

With a `ComboBox` component instance `my_cb` on the Stage, and a `Button` component instance `my_button`, the following example opens the drop-down list of the `my_cb` combo box when the `my_button` button is clicked:

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
    my_cb.open();
};
my_button.addEventListener("click", btnListener);
```

See also

[ComboBox.close\(\)](#)

ComboBox.open

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.open = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("open", listenerObject)
```

Description

Event; broadcast to all registered listeners when the drop-down list is completely open.

Using the dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, *open*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

With a `ComboBox` component instance `my_cb` on the Stage, the following example sends a message to the Output panel when the drop-down list opens or closes:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create Listener Object.
var cbListener:Object = new Object();
cbListener.open = function(evt_obj:Object) {
    trace("The ComboBox has opened.");
}
cbListener.close = function(evt_obj:Object){
    trace("The ComboBox has closed.");
}

// Add Listener.
my_cb.addEventListener("open", cbListener);
my_cb.addEventListener("close", cbListener);

// Open the combo box.
my_cb.open();
```

See also

[ComboBox.close](#), [EventDispatcher.addEventListener\(\)](#)

ComboBox.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items in the list. This is a method of the List component that is available from an instance of the ComboBox component.

Example

With a ComboBox instance `my_cb` on the Stage, and a Button component instance `clear_button` on the Stage, the following ActionScript positions the combo box and button beside each other. When you click the combo box, you'll see a list of items. When you click the button, it clears the combo box's items:

```
my_cb.move(10, 10);
clear_button.move(120, 10);

// Create dataprovider.
var myDP_array:Array = new Array();
myDP_array.push({data:1, label:"First Item"});
myDP_array.push({data:2, label:"Second Item"});

my_cb.dataProvider = myDP_array;

// Define event listener object.
var clearListener:Object = new Object();
clearListener.click = function(evt_obj:Object){
    my_cb.removeAll();
}

// Add Listener.
clear_button.addEventListener("click", clearListener);
```

See also

[ComboBox.removeItemAt\(\)](#), [ComboBox.replaceItemAt\(\)](#)

ComboBox.removeItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.removeItemAt(index)
```

Parameters

index A number that indicates the position of the item to remove. The index is zero-based.

Returns

An object; the removed item (undefined if no item exists).

Description

Method; removes the item at the specified index position. The list indices after the index indicated by the *index* parameter collapse by one. This is a method of the List component that is available from an instance of the ComboBox component.

Example

With a ComboBox instance `my_cb` on the Stage, and a Button component instance `clear_button` on the Stage, the following ActionScript positions the combo box and button beside each other. When you click the combo box, you'll see a list of two items. When you click the button, it clears the combo box's second item (at index position 1, because the value is zero-based):

```
my_cb.move(10, 10);
clear_button.move(120, 10);

// Create data provider.
var myDP_array:Array = new Array();
myDP_array.push({data:1, label:"First Item"});
myDP_array.push({data:2, label:"Second Item"});

my_cb.dataProvider = myDP_array;
```

```
// Define event listener object.
var clearListener:Object = new Object();
clearListener.click = function(evt_obj:Object){
    my_cb.removeItemAt(1);
}

// Add Listener.
clear_button.addEventListener("click", clearListener);
```

See also

[ComboBox.removeAll\(\)](#), [ComboBox.replaceItemAt\(\)](#)

ComboBox.replaceItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.replaceItemAt(index, label[, data])
comboBoxInstance.replaceItemAt(index, {label:label[, data:data])
comboBoxInstance.replaceItemAt(index, obj);
```

Parameters

index A number 0 or greater that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional.

obj An object with *label* and *data* properties.

Returns

Nothing.

Description

Method; replaces the content of the item at the specified index. This is a method of the List component that is available from the ComboBox component.

Example

With a `ComboBox` component instance `my_cb`, and a `TextInput` component instance `label_ti` on the Stage, the following `ActionScript` code adds the user input to the combo box when the user presses the Enter key:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// Create listener for user pressing Enter key on the Text Input field.
var tiListener:Object = new Object();
tiListener.enter = function(evt_obj:Object) {
    my_cb.replaceItemAt(my_cb.selectedIndex, {label:evt_obj.target.text});
    // Needed to refresh recently modified ComboBox entry
    my_cb.selectedIndex = my_cb.selectedIndex;
};
label_ti.addEventListener("enter", tiListener);
```

See also

[ComboBox.removeAll\(\)](#), [ComboBox.removeItemAt\(\)](#)

ComboBox.restrict

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.restrict

Description

Property; indicates the set of characters that a user can enter in the text field of a combo box. The default value is `undefined`. If this property is `null` or an empty string (`""`), a user can enter any character. If this property is a string of characters, the user can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (`-`).

If the string begins with a caret (`^`), all characters that follow the caret are considered unacceptable characters. If the string does not begin with a caret, the characters in the string are considered acceptable.

You can use the backslash (\) to enter a hyphen (-), caret (^), or backslash (\) character, as shown here:

```
\^  
\-  
\\
```

When you enter a backslash in the Actions panel within double quotation marks, it has a special meaning for the Actions panel's double-quote interpreter. It signifies that the character following the backslash should be treated "as is." For example, you could use the following code to enter a single quotation mark:

```
var leftQuote = "\'";
```

The Actions panel's restrict interpreter also uses the backslash as an escape character. Therefore, you may think that the following should work:

```
myText.restrict = "0-9\-\^\\";
```

However, since this expression is surrounded by double quotation marks, the value 0-9-\^\ is sent to the restrict interpreter, and the restrict interpreter doesn't understand this value.

Because you must enter this expression within double quotation marks, you must not only provide the expression for the restrict interpreter, but you must also escape the expression so that it will be read correctly by the Actions panel's built-in interpreter for double quotation marks. To send the value 0-9\-\^\ to the restrict interpreter, you must enter the following code:

```
myCombo.restrict = "0-9\\-\\^\\\\\";
```

The `restrict` property restricts only user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

With a ComboBox component instance `my_cb`, the following ActionScript restricts the entry of characters to numbers 0-9, dashes, and dots:

```
// Add Items to List.  
my_cb.addItem({data:1, label:"First Item"});  
my_cb.addItem({data:2, label:"Second Item"});  
  
// Enable editing of combo box.  
my_cb.editable = true;  
  
// Restrict the characters that can be entered into combo box.  
my_cb.restrict = "0-9\\-\\.\\\"";
```

In the following example, the first line of code limits the text field to uppercase letters, numbers, and spaces. The second line of code allows all characters except lowercase letters.

```
my_combo.restrict = "A-Z 0-9";  
my_combo.restrict = "^a-z";
```

The following code allows a user to enter the characters “0 1 2 3 4 5 6 7 8 9 - ^ \” in the instance `myCombo`. You must use a double backslash to escape the characters `-`, `^`, and `\`. The first `\` escapes the double quotation marks, and the second `\` tells the interpreter that the next character should not be treated as a special character.

```
myCombo.restrict = "0-9\\-\\^\\\\";
```

ComboBox.rowCount

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.rowCount

Description

Property; the maximum number of rows visible in the drop-down list before the combo box inserts a scroll bar. The default value is 5.

If the number of items in the drop-down list is greater than the `rowCount` property, the list resizes and a scroll bar is displayed if necessary. If the drop-down list contains fewer items than the `rowCount` property, it resizes to the number of items in the list.

This behavior differs from the List component, which always shows the number of rows specified by its `rowCount` property, even if some empty space is shown.

If the value is negative or fractional, the behavior is undefined.

Example

With a `ComboBox` component instance `my_cb`, the following `ActionScript` sets the combo box to show the first three items, and add a scrollbar to see the fourth:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});

// Display scroll bar if ComboBox has more than 3 items.
my_cb.rowCount = 3;
```

ComboBox.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
    // Your code here.
};
comboBoxInstance.addEventListener("scroll", listenerObject);
```

Event object

Along with the standard event object properties, the `scroll` event has one additional property, `direction`. It is a string with two possible values, "horizontal" or "vertical". For a `ComboBox` `scroll` event, the value is always "vertical".

Description

Event; broadcast to all registered listeners when the drop-down list is scrolled. This is a `List` component event that is available to the `ComboBox` component.

Using a dispatcher/listener event model, a component instance (*comboBoxInstance*) dispatches an event (in this case, *scroll*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

With a `ComboBox` component instance `my_cb`, the following example sends a message to the Output panel that indicates the index of the item that the list scrolled to:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});

// Display scroll bar if ComboBox has more than 2 items.
my_cb.rowCount = 3;

// Create Listener Object.
var cbListener:Object = new Object();
cbListener.scroll = function(evt_obj:Object) {
    trace("The list had been scrolled to item # "+evt_obj.position);
};

// Add Listener.
my_cb.addEventListener("scroll", cbListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

ComboBox.selectedIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.selectedIndex

Description

Property; the index number of the selected item in the drop-down list. The default value is 0. Assigning this property clears the current selection, selects the indicated item, and displays the label of that item in the combo box's text box.

If you assign an out-of-range value to this property, Flash ignores it. Entering text into the text field of an editable combo box sets selectedIndex to undefined.

Example

With a ComboBox component instance `my_cb`, the following code selects the last item in the list (otherwise, by default it would display the first item):

```
// Add Items to List.  
my_cb.addItem({data:1, label:"First Item"});  
my_cb.addItem({data:2, label:"Second Item"});  
my_cb.addItem({data:3, label:"Third Item"});  
my_cb.addItem({data:4, label:"Fourth Item"});  
  
// Select last item on the list.  
my_cb.selectedIndex = my_cb.length-1;
```

See also

[ComboBox.selectedItem](#)

ComboBox.selectedItem

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.selectedItem

Description

Property; the value of the selected item in the drop-down list.

If the combo box is editable, `selectedItem` returns `undefined` if the user enters any text in the text box. The property only has a value if you select an item from the drop-down list or set the value using ActionScript. If the combo box is static, the value of `selectedItem` is always valid; it returns `undefined` if there are no items in the list.

Example

With a `ComboBox` component instance `my_cb`, the following example shows the values for the `selectedItem` data and label properties:

```
// Add Items to List.
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});

var cbListener:Object = new Object();
cbListener.change = function(evt_obj:Object) {
    var item_obj:Object = my_cb.selectedItem;
    var i:String;
    for (i in item_obj) {
        trace(i + ":\t" + item_obj[i]);
    }
    trace("");
};
my_cb.addEventListener("change", cbListener);
```

See also

[ComboBox.dataProvider](#), [ComboBox.selectedIndex](#)

ComboBox.sortItems()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
comboBoxInstance.sortItems([compareFunc], [optionsFlag])
```

Parameters

compareFunc A reference to a function that compares two items to determine their sort order. For details, see `Array.sort()` in *ActionScript 2.0 Language Reference*. This parameter is optional.

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or re-sort it repeatedly. This parameter is optional.

The following are possible values for *optionsFlag*:

- `Array.DECENDING`, which sorts highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the combo box according to the specified compare function or according to the specified sort options.

Example

This example sorts according to uppercase labels. The items `a` and `b` are passed to the function and contain `label` and `data` fields:

```
myComboBox.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
    return a.label.toUpperCase() > b.label.toUpperCase();
}
```

The following example uses the `upperCaseFunc()` function defined above, along with the `optionsFlag` parameter to sort the elements of a `ComboBox` instance named `myComboBox`:

```
myComboBox.addItem("Mercury");
myComboBox.addItem("Venus");
myComboBox.addItem("Earth");
myComboBox.addItem("planet");
myComboBox.sortItems(upperCaseFunc, Array.DESENDING);
// The resulting sort order of myComboBox will be:
// Venus
// planet
// Mercury
// Earth
```

ComboBox.sortItemsBy()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
comboBoxInstance.sortItemsBy(fieldName, order [optionsFlag])
```

Parameters

fieldName A string that specifies the name of the field to use for sorting. This value is usually `"label"` or `"data"`.

order A string that specifies whether to sort the items in ascending order (`"ASC"`) or descending order (`"DESC"`).

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or re-sort it repeatedly. This parameter is optional, but if used, should replace the *order* parameter.

The following are possible values for *optionsFlag*:

- `Array.DESENDING`, which sorts highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the combo box alphabetically or numerically, in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but advanced programmers may specify any primitive value. If you want, you can use the *optionsFlag* parameter to specify a sorting style.

Example

The following examples are based on a `ComboBox` instance named `myComboBox`, which contains four elements labeled "Apples", "Bananas", "cherries", and "Grapes":

```
// First, populate the ComboBox with the elements.
myComboBox.addItem("Bananas");
myComboBox.addItem("Apples");
myComboBox.addItem("cherries");
myComboBox.addItem("Grapes");

// The following statement sorts using the order parameter set to "ASC",
// and results in a sort that places "cherries" at the bottom of the list
// because the sort is case-sensitive.
myComboBox.sortItemsBy("label", "ASC");
// resulting order: Apples, Bananas, Grapes, cherries

// The following statement sorts using the order parameter set to "DESC",
// and results in a sort that places "cherries" at the top of the list
// because the sort is case-sensitive.
myComboBox.sortItemsBy("label", "DESC");
// resulting order: cherries, Grapes, Bananas, Apples

// The following statement sorts using the optionsFlag parameter set to
// Array.CASEINSENSITIVE. Note that an ascending sort is the default
// setting.
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE);
// resulting order: Apples, Bananas, cherries, Grapes

// The following statement sorts using the optionsFlag parameter set to
// Array.CASEINSENSITIVE | Array.DECENDING.
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE | Array.DECENDING);
// resulting order: Grapes, cherries, Bananas, Apples
```

ComboBox.text

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.text

Description

Property; the text of the text box. You can get and set this value for editable combo boxes. For static combo boxes, the value is read-only.

Example

The following example sets the current text value of an editable combo box:

```
my_cb.addItem("Arkansas");  
my_cb.addItem("Georgia");
```

```
my_cb.editable = true;  
my_cb.text = "California";
```

ComboBox.textField

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.textField

Description

Property (read-only); a reference to the TextInput component contained by the ComboBox component.

This property lets you access the underlying TextInput component so that you can manipulate it. For example, you might want to change the selection of the text box or restrict the characters that can be entered in it.

Example

The following code restricts the text box of `myComboBox` so that it only accept numbers to maximum of six characters:

```
// Add Items to List.
my_cb.addItem({data:0xFFFFF, label:"white"});
my_cb.addItem({data:0x000000, label:"black"});

my_cb.editable = true;

// Restrict what can be entered into textfield to only 0-9.
my_cb.restrict = "0-9";

// Limit input to 6 characters.
my_cb.textField.maxChars = 6;
```

ComboBox.value

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.value

Description

Read-only property; if the combo box is editable, `value` returns the item label. If the combo box is static, `value` returns the item data.

Example

The following example puts the data into the combo box by setting the `dataProvider` property. It then displays the `value` in the Output panel. Finally, it selects "California" and displays it in the Output panel while the combo box is editable, and then displays "CA" when the combo box is not editable.

```
my_cb.dataProvider = [
    {label:"Alaska", data:"AK"},
    {label:"California", data:"CA"},
    {label:"Washington", data:"WA"}];
my_cb.editable = true;
my_cb.selectedIndex = 1;
trace('Editable value is "California": ' + my_cb.value);
my_cb.editable = false;
my_cb.selectedIndex = 1;
trace('Non-editable value is "CA": ' + my_cb.value);
```


Data binding classes (Flash Professional only)

The data binding classes provide the runtime functionality for the data binding feature in Flash Professional 8. You can visually create and configure data bindings in the Flash authoring environment by using the Bindings tab in the Component inspector, or you can programmatically create and configure bindings by using the classes in the `mx.data.binding` package.

For an overview of data binding and how to visually create data bindings in the Flash authoring tool, see “Data binding (Flash Professional only)” in *Using Flash*.

Making data binding classes available at runtime (Flash Professional only)

To compile your SWF file, your library must contain SWC files that contain the byte code for the data binding classes and web service classes. If you create data bindings in Flash while authoring, the relevant component classes are automatically added to the library. If you work with data binding and web services at runtime, you must add the classes to your FLA file’s library. You can get these SWC files from the Classes common library.

To add the SWC files to your library:

1. Select the Classes library (Window > Common Libraries > Classes).
2. Open the library for your document (Window > Library).
3. Drag the appropriate SWC files (DataBindingClasses, WebServiceClasses, or both) from the Classes library into your document’s library.

For more information on these classes, see [“Binding class \(Flash Professional only\)” on page 208](#) and [“Web service classes \(Flash Professional only\)” on page 1413](#).

Classes in the mx.data.binding package (Flash Professional only)

The following table lists the classes in the mx.data.binding package:

Class	Description
Binding class (Flash Professional only)	Creates a binding between two endpoints.
ComponentMixins class (Flash Professional only)	Adds data binding functionality to components.
CustomFormatter class (Flash Professional only)	The base class for creating custom formatter classes.
CustomValidator class (Flash Professional only)	The base class for creating custom validator classes.
DataType class (Flash Professional only)	Provides read and write access to data fields of a component property.
EndPoint class (Flash Professional only)	Defines the source or destination of a binding.
TypedValue class (Flash Professional only)	Contains a data value and information about the value's data type.

Binding class (Flash Professional only)

ActionScript Class Name mx.data.binding.Binding

The Binding class defines an association between two endpoints, a source and a destination. It listens for changes to the source endpoint and copies the changed data to the destination endpoint each time the source changes.

You can write custom bindings by using the Binding class (and supporting classes), or use the Bindings tab in the Component inspector.

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file. For more information, see [“Making data binding classes available at runtime \(Flash Professional only\)” on page 207](#).

For an overview of the classes in the mx.data.binding package, see [“Classes in the mx.data.binding package \(Flash Professional only\)” on page 208](#).

Method summary for the Binding class

The following table lists the methods of the Binding class.

Method	Description
Binding.execute()	Fetches the data from the source component, formats it, and assigns it to the destination component.

Constructor for the Binding class

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
new Binding(source, destination, [format], [isTwoWay])
```

Parameters

source A source endpoint of the binding. This parameter is nominally of type `mx.data.binding.EndPoint`, but can be any ActionScript object that has the required Endpoint fields (see “[EndPoint class \(Flash Professional only\)](#)” on page 220).

destination The destination endpoint of the binding. This parameter is nominally of type `mx.data.binding.EndPoint`, but can be any ActionScript object that has the required Endpoint fields.

format An optional object that contains formatting information. The object must have the following properties:

- *cls* An ActionScript class that extends the class `mx.data.binding.DataAccessor`.
- *settings* An object whose properties provide optional settings for the formatter class specified by *cls*.

isTwoWay An optional Boolean value that specifies whether the new Binding object is bidirectional (`true`) or not (`false`). The default value is `false`.

Returns

Nothing.

Description

Constructor; creates a new Binding object. You can bind data to any ActionScript object that has properties and emits events including, but not limited to, components.

A binding object exists as long as the innermost movie clip contains both the source and destination components. For example, if movie clip named A contains components X and Y, and there is a binding between X and Y, then the binding is in effect as long as movie clip A exists.

NOTE

It's not necessary to retain a reference to the new Binding object. As soon as the Binding object is created, it immediately begins listening for "changed" events emitted by either endpoint. In some cases, however, you might want to save a reference to the new Binding object, so that you can call its `execute()` method at a later time (see [Binding.execute\(\)](#)).

Example

In this example, the `text` property of a `TextInput` component (`src_txt`) is bound to the `text` property of another `TextInput` component (`dest_txt`). When the `src_txt` text field loses focus (that is, when the `focusOut` event is generated), the value of its `text` property is copied into `dest_txt.text`.

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";
```

```
var dest = new EndPoint();
dest.component = dest_txt;
dest.property = "text";
```

```
new Binding(src, dest);
```

The following example demonstrates how to create a Binding object that uses a custom formatter class. For more information, see ["CustomFormatter class \(Flash Professional only\)" on page 212](#).

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";
```

```
var dest = new EndPoint();
dest.component = text_dest;
dest.property = "text";
```

```
new Binding(src, dest, {cls: mx.data.formatters.Custom, settings:
    {classname: "com.mycompany.SpecialFormatter"}});
```

Binding.execute()

Availability

Flash Player 6.

Edition

Flash MX Professional 2004.

Usage

```
myBinding.execute([reverse])
```

Parameters

reverse A Boolean value that specifies whether the binding should also be executed from the destination to the source (`true`), or only from the source to the destination (`false`). By default, this value is `false`.

Returns

A `null` value if the binding executed successfully; otherwise, the method returns an array of error message strings that describe the errors that prevented the binding from executing.

Description

Method; fetches the data from the source component and assigns it to the destination component. If the binding uses a formatter, then the data is formatted before being assigned to the destination.

This method also validates the data and causes either a `valid` or `invalid` event to be emitted by the destination and source components. Data is assigned to the destination even if it's invalid, unless the destination is read-only.

If the *reverse* parameter is set to `true` and the binding is two-way, then the binding is executed in reverse (from the destination to the source).

Example

The following code, attached to a Button component instance, executes the binding in reverse (from the destination component to the source component) when the button is clicked.

```
on(click) {  
    _root.myBinding.execute(true);  
}
```

CustomFormatter class (Flash Professional only)

ActionScript Class Name mx.data.binding.CustomFormatter

The CustomFormatter class defines two methods, `format()` and `unformat()`, that provide the ability to transform data values from a specific data type to String, and vice versa. By default, these methods do nothing; you must implement them in a subclass of `mx.data.binding.CustomFormatter`.

To create your own custom formatter, you first create a subclass of CustomFormatter that implements `format()` and `unformat()` methods. You can then assign that class to a binding between components either by creating a new Binding object with ActionScript (see [“Binding class \(Flash Professional only\)” on page 208](#)), or by using the Bindings tab in the Component inspector. For information on assigning a formatter class using the Component inspector, see “Schema formatters” in *Using Flash*.

You can also assign a formatter class to a component property on the Schema tab of the Component inspector. However, in that case, the formatter is used only when the data is needed in the form of a string. In contrast, formatters assigned with the Bindings panel, or created with ActionScript, are used whenever when the binding is executed.

For an example of writing and assigning a custom formatter using ActionScript, see [“Sample custom formatter” on page 212](#).

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the `mx.data.binding` package, see [“Classes in the mx.data.binding package \(Flash Professional only\)” on page 208](#).

Sample custom formatter

The following example demonstrates how to create a custom formatter class and then apply it to a binding between two components by using ActionScript. In this example, the current value of a NumericStepper component (its `value` property) is bound to the current value of a TextInput component (its `text` property). The custom formatter class formats the current numeric value of the NumericStepper component (for example, 1, 2, or 3) as its English word equivalent (for example, “one”, “two”, or “three”) before assigning it to the TextInput component.

To create and use a custom formatter:

1. In Flash, create a new ActionScript file.
2. Add the following code to the file:

```
// NumberFormatter.as
class NumberFormatter extends mx.data.binding.CustomFormatter {
    // Format a Number, return a String
    function format(rawValue) {
        var returnValue;
        var strArray = new Array('one', 'two', 'three');
        var numArray = new Array(1, 2, 3);
        returnValue = 0;
        for (var i = 0; i<strArray.length; i++) {
            if (rawValue == numArray[i]) {
                returnValue = strArray[i];
                break;
            }
        }
        return returnValue;
    } // convert a formatted value, return a raw value
    function unformat(formattedValue) {
        var returnValue;
        var strArray = new Array('one', 'two', 'three');
        var numArray = new Array(1, 2, 3);
        returnValue = "invalid";
        for (var i = 0; i<strArray.length; i++) {
            if (formattedValue == strArray[i]) {
                returnValue = numArray[i];
                break;
            }
        }
        return returnValue;
    }
}
```

3. Save the ActionScript file as NumberFormatter.as.
4. Create a new Flash (FLA) file.
5. From the Components panel, drag a TextInput component to the Stage and name it **textInput**. Then drag a NumericStepper component to the Stage and name it **stepper**.
6. Open the Timeline and select the first frame on Layer 1.
7. In the Actions panel, add the following code to the Actions panel:

```
import mx.data.binding.*;
var x:NumberFormatter;
var customBinding = new Binding({component:stepper, property:"value",
    event:"change"}, {component:textInput, property:"text",
    event:"enter,change"}, {cls:mx.data.formatters.Custom,
    settings:{classname:"NumberFormatter"}});
```

The second line of code (`var x:NumberFormatter`) ensures that the byte code for your custom formatter class is included in the compiled SWF file.

8. Select Window > Common Libraries > Classes to open the Classes library.
9. Select Window > Library to open your document's library.
10. Drag DataBindingClasses from the Classes library to your document's library.
This makes the data binding runtime classes available for your document.
11. Save the FLA file to the same folder that contains NumberFormatter.as.
12. Test the file (Control > Test Movie).
Click the buttons on the NumericStepper component and watch the contents of the TextInput component update.

Method summary for the CustomFormatter class

The following table lists the methods of the CustomFormatter class.

Method	Description
<code>CustomFormatter.format()</code>	Converts from a raw data type to a new object.
<code>CustomFormatter.unformat()</code>	Converts from a string, or other data type, to a raw data type.

CustomFormatter.format()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

This method is called automatically; you don't invoke it directly.

Parameters

rawData The data to be formatted.

Returns

A formatted value.

Description

Method; converts from a raw data type to a new object.

This method is not implemented by default. You must define it in your subclass of `mx.data.binding.CustomFormatter`.

For more information, see [“Sample custom formatter” on page 212](#).

CustomFormatter.unformat()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

This method is called automatically; you don't invoke it directly.

Parameters

formattedData The formatted data to convert back to the raw data type.

Returns

An unformatted value.

Description

Method; converts from a string, or other data type, to the raw data type. This transformation should be the exact inverse transformation of `CustomFormatter.format()`.

This method is not implemented by default. You must define it in your subclass of `mx.data.binding.CustomFormatter`.

For more information, see [“Sample custom formatter” on page 212](#).

CustomValidator class (Flash Professional only)

ActionScript Class Name mx.data.binding.CustomValidator

You use the CustomValidator class when you want to perform custom validation of a data field contained by a component.

To create a custom validator class, you first create a subclass of mx.data.binding.CustomValidator that implements a method named `validate()`. This method is automatically passed a value to be validated. For more information about how to implement this method, see [CustomValidator.validate\(\)](#).

Next, you assign your custom validator class to a field of a component by using the Component inspector's Schema tab. For an example of creating and using a custom validator class, see the Example section in the [CustomValidator.validate\(\)](#) entry.

To assign a custom validator:

1. In the Component inspector, select the Schema tab.
2. Select the field you want to validate, and then select Custom from the Data Type pop-up menu.
3. Select the Validation Options field (at the bottom of the Schema tab), and click the magnifying glass icon to open the Custom Validation Settings dialog box.
4. In the ActionScript Class text box, enter the name of the custom validator class you created.
In order for the class you specify to be included in the published SWF file, it must be in the classpath.

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the mx.data.binding package, see [“Classes in the mx.data.binding package \(Flash Professional only\)” on page 208](#).

Method summary for the CustomValidator class

The following table lists the methods of the CustomValidator class.

Method	Description
CustomValidator.validate()	Performs validation on data.
CustomValidator.validationError()	Reports validation errors.

CustomValidator.validate()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

This method is called automatically; you don't invoke it directly.

Parameters

value The data to be validated; it can be of any type.

Returns

Nothing.

Description

Method; called automatically to validate the data contained by the *value* parameter. You must implement this method in your subclass of CustomValidator; the default implementation does nothing.

You can use any ActionScript code to examine and validate the data. If the data is not valid, this method should call `this.validationError()` with an appropriate message. You can call `this.validationError()` more than once if there are several validation problems with the data.

Since `validate()` might be called repeatedly, avoid adding code that takes a long time to complete. Your implementation of this method should only check for validity, and then report any errors using `CustomValidator.validationError()`. Similarly, your implementation should not take any action as a result of the validation test, such as alerting the end user. Instead, create event listeners for `valid` and `invalid` events and alert the end user from those event listeners (see the example below).

Example

The following procedure demonstrates how to create and use a custom validator class. The `validate()` method of the CustomValidator class `OddNumbersOnly.as` determines any value that is not an odd number to be invalid. The validation occurs whenever a change occurs in the value of a `NumericStepper` component, which is bound to the `text` property of a `Label` component.

To create and use a custom validator class:

1. In Flash, create a new ActionScript (AS) file.
2. Add the following code to the AS file:

```
class OddNumbersOnly extends mx.data.binding.CustomValidator
{
    public function validate(value) {
        // make sure the value is of type Number
        var n = Number(value);
        if (String(n) == "NaN") {
            this.validationError("'" + value + "' is not a number.");
            return;
        }
        // make sure the number is odd
        if (n % 2 == 0) {
            this.validationError("'" + value + "' is not an odd number.");
            return;
        }
        // data is OK, no need to do anything, just return
    }
}
```

3. Save the AS file as `OddNumbersOnly.as`.

NOTE

The name of the AS file must match the name of the class.

4. Create a new Flash (FLA) file.
5. Open the Components panel.
6. Drag a NumericStepper component from the Components panel to the Stage and name it **stepper**.
7. Drag a Label component to the Stage and name it **textLabel**.
8. Drag a TextArea component to the Stage and name it **status**.
9. Select the NumericStepper component, and open the Component inspector.
10. Select the Bindings tab in the Component inspector, and click the Add Binding (+) button.
11. Select the Value property (the only one) in the Add Bindings dialog box, and click OK.
12. In the Component inspector, double-click Bound To in the Binding Attributes pane of the Bindings tab to open the Bound To dialog box.
13. In the Bound To dialog box, select the Label component in the Component Path pane and its `text` property in the Schema Location pane. Click OK.
14. Select the Label component on the Stage, and click the Schema tab in the Component inspector.

15. In the Schema Attributes pane, select Custom from the Data Type pop-up menu.
16. Double-click the Validation Options field in the Schema Attributes pane to open the Custom Validation Settings dialog box.
17. In the ActionScript Class text box, enter **OddNumbersOnly**, which is the name of the ActionScript class you created previously. Click OK.
18. Open the Timeline and select the first frame on Layer 1.
19. Open the Actions panel.
20. Add the following code to the Actions panel:

```
function dataIsValid(evt) {  
    if (evt.property == "text") {  
        status.text = evt.messages;  
    }  
}  
  
function dataIsInvalid(evt) {  
    if (evt.property == "text") {  
        status.text = "OK";  
    }  
}  
  
textField.addEventListener("valid", dataIsValid);  
textField.addEventListener("invalid", dataIsInvalid);
```

21. Save the FLA file as **OddOnly.fla** to the same folder that contains **OddNumbersOnly.as**.
22. Test the SWF file (Control > Test Movie).

Click the arrows on the **NumericStepper** component to change its value. Notice the message that appears in the **TextArea** component when you choose even and odd numbers.

CustomValidator.validationError()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`this.validationError(errorMessage)`

NOTE

This method can be invoked only from within a custom validator class; the keyword `this` refers to the current `CustomValidator` object.

Parameters

errorMessage A string that contains the error message to be reported.

Returns

Nothing.

Description

Method; called from the `validate()` method of your subclass of `CustomValidator` to report validation errors. If you don't call `validationError()`, a valid event is generated when `validate()` finishes executing. If you call `validationError()` one or more times from within the `validate()`, an invalid event is generated after `validate()` returns.

Each message you pass to `validationError()` is available in the `messages` property of the event object that was passed to the `invalid` event handler.

Example

See the Example section for [CustomValidator.validate\(\)](#).

EndPoint class (Flash Professional only)

ActionScript Class Name `mx.data.binding.EndPoint`

The `EndPoint` class defines the source or destination of a binding. `EndPoint` objects define a constant value, component property, or particular field of a component property, from which you can get data, or to which you can assign data. They can also define an event, or list of events, that a `Binding` object listens for; when the specified event occurs, the binding executes.

When you create a new binding with the `Binding` class constructor, you pass it two `EndPoint` objects: one for the source and one for the destination.

```
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

The `EndPoint` objects, `srcEndPoint` and `destEndPoint`, might be defined as follows:

```
var srcEndPoint = new mx.data.binding.EndPoint();
var destEndPoint = new mx.data.binding.EndPoint();
srcEndPoint.component = source_txt;
srcEndPoint.property = "text";
srcEndPoint.event = "focusOut";
destEndPoint.component = dest_txt;
destEndPoint.property = "text";
```

In English, the above code means “When the source text field loses focus, copy the value of its `text` property into the `text` property of the destination text field.”

You can also pass generic `ActionScript` objects to the `Binding` constructor, rather than passing explicitly constructed `EndPoint` objects. The only requirement is that the objects define the required `EndPoint` properties, `component` and `property`. The following code is equivalent to that shown above.

```
var srcEndPoint = {component:source_txt, property:"text"};
var destEndPoint = {component:dest_txt, property:"text"};
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the `mx.data.binding` package, see “[Classes in the mx.data.binding package \(Flash Professional only\)](#)” on page 208.

Property summary for the `EndPoint` class

The following table lists the properties of the `EndPoint` class.

Method	Description
<code>EndPoint.component</code>	A reference to a component instance.
<code>EndPoint.constant</code>	A constant value.
<code>EndPoint.event</code>	The name of an event, or array of event names, that the component emits when the data changes.
<code>EndPoint.location</code>	The location of a data field within the property of the component instance.
<code>EndPoint.property</code>	The name of a property of the component instance specified by <code>EndPoint.component</code> .

Constructor for the EndPoint class

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
new EndPoint()
```

Returns

Nothing.

Description

Constructor; creates a new EndPoint object.

Example

This example creates a new EndPoint object named `source_obj` and assigns values to its `component` and `property` properties:

```
var source_obj = new mx.data.binding.EndPoint();
source_obj.component = myTextField;
source_obj.property = "text";
```

EndPoint.component

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
endPointObj.component
```

Description

Property; a reference to a component instance.

Example

This example assigns an instance of the List component (`listBox1`) as the component parameter of an `EndPoint` object.

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.component = listBox1;
```

EndPoint.constant

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

endPoint_src.constant

Description

Property; a constant value assigned to an `EndPoint` object. This property can be applied only to `EndPoint` objects that are the source, not the destination, of a binding between components. The value can be of any data type that is compatible with the destination of the binding. If this property is specified, all other `EndPoint` properties for the specified `EndPoint` object are ignored.

Example

In this example, the string constant value “hello” is assigned to an `EndPoint` object’s constant property:

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.constant = "hello";
```

EndPoint.event

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

endPointObj.event

Description

Property; specifies the name of an event, or an array of event names, generated by the component when data assigned to the bound property changes. When the event occurs, the binding executes.

The specified event only applies to components that are used as the source of a binding, or as the destination of a two-way binding. For more information about creating two-way bindings, see [“Binding class \(Flash Professional only\)” on page 208](#).

Example

In this example, the `text` property of one `TextInput` (`src_txt`) component is bound to the same property of another `TextInput` component (`dest_txt`). The binding is executed when either the `focusOut` or `enter` event is emitted by the `src_txt` component.

```
var source = {component:src_txt, property:"text", event:["focusOut",
    "enter"]};
var dest = {component:myTextArea, property:"text"};
var newBind = new mx.data.binding.Binding(source, dest);
```

EndPoint.location

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

endPointObj.location

Description

Property; specifies the location of a data field within the property of the component instance. There are four ways to specify a location: as a string that contains an XPath expression, as a string that contains an ActionScript path, as an array of strings, or as an object.

XPath expressions can only be used when the data is an XML object. (See Example 1 below.) For a list of supported XPath expressions, see [“Adding bindings using path expressions” in *Using Flash*](#).

For XML and ActionScript objects, you can also specify a string that contains an ActionScript path. An ActionScript path contains the names of fields separated by dots (for example, "a.b.c").

You can also specify an array of strings as a location. Each string in the array “drills down” another level of nesting. You can use this technique with both XML and ActionScript data. (See Example 2 below.) When used with ActionScript data, an array of strings is equivalent to use of an ActionScript path; that is, the array ["a", "b", "c"] is equivalent to "a.b.c".

If you specify an object as the location, the object must specify two properties: `path` and `indices`. The `path` property is an array of strings, as discussed above, except that one or more of the specified strings may be the special token "[n]". For each occurrence of this token in `path`, there must be a corresponding index item in `indices`. As the path is evaluated, the `indices` are used to index into arrays. The index item can be any EndPoint object. This type of location can be applied to ActionScript data only—not XML. (See Example 3 below.)

Example

Example 1: This example uses an XPath expression to specify the location of a node named `zip` in an XML object:

```
var sourceEndPoint = new mx.databinding.EndPoint();
var sourceObj = new Object();
sourceObj.xml = new XML("<zip>94103</zip>");
sourceEndPoint.component = sourceObj;
sourceEndPoint.property = "xml";
sourceEndPoint.location = "/zip";
```

Example 2: This example uses an array of strings to “drill down” to a nested movie clip property:

```
var sourceEndPoint = new mx.data.binding.EndPoint();
// Assume movieClip1.ball.position exists.
sourceEndPoint.component = movieClip1;
sourceEndPoint.property = "ball";
// Access movieClip1.ball.position.x.
sourceEndPoint.location = ["position", "x"];
```

Example 3: This example shows how to use an object to specify the location of a data field in a complex data structure:

```
var city = new Object();
city.theaters = [{theater: "t1", movies: [{name: "Good,Bad,Ugly"},
    {name:"Matrix Reloaded"}]}, {theater: "t2", movies: [{name: "Gladiator"},
    {name: "Catch me if you can"}]};
var srcEndPoint = new EndPoint();
srcEndPoint.component = city;
srcEndPoint.property = "theaters";
srcEndPoint.location = {path: ["[n]", "movies", "[n]", "name"], indices:
    [{constant:0}, {constant:0}]};
```

EndPoint.property

Availability

Flash Player 6 (6.0.79.0)

Edition

Flash MX Professional 2004.

Usage

endPointObj.property

Description

Property; specifies a property name of the component instance specified by [EndPoint.component](#) that contains the bindable data.

NOTE

[EndPoint.component](#) and [EndPoint.property](#) must combine to form a valid ActionScript object/property combination.

Example

This example binds the `text` property of one `TextInput` component (`text_1`) to the same property in another `TextInput` component (`text_2`).

```
var sourceEndPoint = {component:text_1, property:"text"};
var destEndPoint = {component:text_2, property:"text"};
new Binding(sourceEndPoint, destEndPoint);
```

ComponentMixins class (Flash Professional only)

ActionScript Class Name `mx.data.binding.ComponentMixins`

The `ComponentMixins` class defines properties and methods that are automatically added to any object that is the source or destination of a binding, or to any component that's the target of a `ComponentMixins.initComponent()` method call. These properties and methods do not affect normal component functionality; rather, they add functionality that is useful with data binding.

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the `mx.data.binding` package, see [“Classes in the mx.data.binding package \(Flash Professional only\)” on page 208](#).

Method summary for the ComponentMixins class

The following table lists the methods of the ComponentMixins class.

Method	Description
<code>ComponentMixins.getField()</code>	Returns an object for getting and setting the value of a field at a specific location in a component property.
<code>ComponentMixins.initComponent()</code>	Adds the ComponentMixins methods to a component.
<code>ComponentMixins.refreshDestinations()</code>	Executes all the bindings that have this object as the source endpoint.
<code>ComponentMixins.refreshFromSources()</code>	Executes all bindings that have this component as the destination endpoint.
<code>ComponentMixins.validateProperty()</code>	Checks to see if the data in the indicated property is valid.

ComponentMixins.getField()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.getField(propertyName, [location])
```

Parameters

propertyName A string that contains the name of a property of the specified component.

location An optional parameter that indicates the location of a field within the component property. This is useful if *propertyName* specifies a complex data structure and you are interested in a particular field of that structure. The *location* property can take one of three forms:

- A string that contains an XPath expression. This is only valid for XML data structures. For a list of supported XPath expressions, see “Adding bindings using path expressions” in *Using Flash*.

- A string that contains field names, separated by dots—for example, "a.b.c". This form is permitted for any complex data (ActionScript or XML).
- An array of strings, where each string is a field name—for example, ["a", "b", "c"]. This form is permitted for any complex data (ActionScript or XML).

Returns

A `DataType` object.

Description

Method; returns a `DataType` object whose methods you can use to get or set the data value in the component property at the specified field location. For more information, see [“DataType class \(Flash Professional only\)” on page 233](#).

Example

This example uses the `DataType.setAsString()` method to set the value of a field located in a component’s property. In this case the property (`results`) is a complex data structure.

```
import mx.data.binding.*;
var field : DataType = myComponent.getField("results", "po.address.name1");
field.setAsString("Teri Randall");
```

See also

[DataType.setAsString\(\)](#)

ComponentMixins.initComponent()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mx.data.binding.ComponentMixins.initComponent(componentInstance)
```

Parameters

componentInstance A reference to a component instance.

Returns

Nothing.

Description

Method (static); adds all the ComponentMixins methods to the component specified by *componentInstance*. This method is called automatically for all components involved in a data binding. To make the ComponentMixins methods available for a component that is not involved in a data binding, you must explicitly call this method for that component.

Example

The following code makes the ComponentMixins methods available to a DataSet component:

```
mx.data.binding.ComponentMixins.initComponent(_root.myDataSet);
```

ComponentMixins.refreshDestinations()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.refreshDestinations()
```

Parameters

None.

Returns

Nothing.

Description

Method; executes all the bindings for which *componentInstance* is the source EndPoint object. This method lets you execute bindings whose sources do not emit a “data changed” event.

Example

The following example executes all the bindings for which the DataSet component instance named *user_data* is the source EndPoint object:

```
user_data.refreshDestinations();
```

ComponentMixins.refreshFromSources()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.refreshFromSources()
```

Parameters

None.

Returns

Nothing.

Description

Method; executes all bindings for which *componentInstance* is the destination EndPoint object. This method lets you execute bindings that have constant sources, or sources that do not emit a “data changed” event.

Example

The following example executes all the bindings for which the ListBox component instance named `cityList` is the destination EndPoint object:

```
cityList.refreshFromSources();
```

ComponentMixins.validateProperty()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.validateProperty(propertyName)
```


Parameters

propertyName A string that contains the name of a property that belongs to *componentInstance*.

Returns

An array, or `null`.

Description

Method; determines if the data in *propertyName* is valid based on the property's schema settings. The property's schema settings are those specified on the Schema tab in the Component inspector.

The method returns `null` if the data is valid; otherwise, it returns an array of error messages as strings.

Validation applies only to fields that have schema information available. If a field is an object that contains other fields, each "child" field is validated, and so on, recursively. Each individual field dispatches a `valid` or `invalid` event, as necessary. For each data field contained by *propertyName*, this method dispatches `valid` or `invalid` events, as follows:

- If the value of the field is `null`, and is *not* required, the method returns `null`. No events are generated.
- If the value the field is `null`, and *is* required, an error is returned and an `invalid` event is generated.
- If the value of the field is not `null` and the field's schema does *not* have a validator, the method returns `null`; no events are generated.
- If the value is not `null` and the field's schema *does* define a validator, the data is processed by the validator object. If the data is valid, a `valid` event is generated and `null` is returned; otherwise, an `invalid` event is generated and an array of error strings is returned.

Example

The following example shows how to use `validateProperty()` to make sure that text entered by a user is of a valid length. You'll determine the valid length by setting the Validation Options for the String data type in the Component inspector's Schema tab. If the user enters a string of invalid length in the text field, the error messages returned by `validateProperty()` are displayed in the Output panel.

To validate text entered by a user in a TextInput component:

1. Drag a TextInput component from the Components panel to the Stage, and name it `zipCode_txt`.
2. Select the TextInput component and, in the Component inspector, click the Schema tab.
3. In the Schema Tree pane (the top pane of the Schema tab), select the `text` property.
4. In the Schema Attributes pane (the bottom pane of the Schema tab), select `ZipCode` from the Data Type pop-up menu.
5. Open the Timeline if it is not already open.
6. Click the first frame on Layer 1 in the Timeline, and open the Actions panel (Window > Actions).

7. Add the following code to the Actions panel:

```
// Add ComponentMixin methods to TextInput component.  
// Note that this step is only necessary if the component  
// isn't already involved in a data binding,  
// either as the source or destination.  
mx.data.binding.ComponentMixins.initComponent(zipCode_txt);  
// Define event listener function for component:  
validateResults = function (eventObj) {  
    var errors:Array = eventObj.target.validateProperty("text");  
    if (errors != null) {  
        trace(errors);  
    }  
};  
// Register listener function with component:  
zipCode_txt.addEventListener("enter", validateResults);
```

8. Select Window > Common Libraries > Classes to open the Classes library.
9. Select Window > Library to open your document's library.
10. Drag `DataBindingClasses` from the Classes library to your document's library.

This step makes the data binding runtime classes available to the SWF file at runtime.

11. Test the SWF file by selecting Control > Test Movie.

In the TextInput component on the Stage, enter an invalid United States zip code—for example, one that contains all letters, or one that contains fewer than five numbers.

Notice the error messages displayed in the Output panel.

DataType class (Flash Professional only)

ActionScript Class Name mx.data.binding.DataType

The `DataType` class provides read and write access to data fields of a component property. To get a `DataType` object, you call the `ComponentMixins.getField()` method on a component. You can then call methods of the `DataType` object to get and set the value of the field.

If you get and set field values directly on the component instance instead of using `DataType` class methods, the data is provided in its “raw” form. In contrast, when you get or set field values using `DataType` methods, the values are processed according to the field’s schema settings.

For example, the following code gets the value of a component’s property directly and assigns it to a variable. The variable, `propVar`, contains whatever “raw” value is the current value of the property `propName`.

```
var propVar = myComponent.propName;
```

The next example gets the value of the same property by using the `DataType.getAsString()` method. In this case, the value assigned to `stringVar` is the value of `propName` after being processed according to its schema settings, and then returned as a string.

```
var dataTypeObj:mx.data.binding.DataType =  
    myComponent.getField("propName");  
var stringVar:String = dataTypeObj.getAsString();
```

For more information about how to specify a field’s schema settings, see “Working with schemas in the Schema tab (Flash Professional only)” in *Using Flash*.

You can also use the methods of the `DataType` class to get or set fields in various data types. The `DataType` class automatically converts the raw data to the requested type, if possible. For example, in the code example above, the data that’s retrieved is converted to the `String` type, even if the raw data is a different type.

The `ComponentMixins.getField()` method is available for components that have been included in a data binding (either as a source, destination, or an index), or that have been initialized with `ComponentMixins.initComponent()`. For more information, see “[ComponentMixins class \(Flash Professional only\)](#)” on page 226.

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the `mx.data.binding` package, see “[Classes in the mx.data.binding package \(Flash Professional only\)](#)” on page 208.

Method summary for the DataType class

The following table lists the methods of the DataType class.

Method	Description
DataType.getAnyTypedValue()	Fetches the current value of the field.
DataType.getAsBoolean()	Fetches the current value of the field as a Boolean value.
DataType.getAsNumber()	Fetches the current value of the field as a number.
DataType.getAsString()	Fetches the current value of the field as a String value.
DataType.getTypedValue()	Fetches the current value of the field in the form of the requested data type.
DataType.setAnyTypedValue()	Sets a new value in the field.
DataType.setAsBoolean()	Sets the field to the new value, which is given as a Boolean value.
DataType.setAsNumber()	Sets the field to the new value, which is given as a number.
DataType.setAsString()	Sets the field to the new value, which is given as a string.
DataType.setTypedValue()	Sets a new value in the field.

Property summary for the DataType class

The following table lists the properties of the DataType class.

Property	Description
DataType.encoder	Provides a reference to the encoder object associated with this field.
DataType.formatter	Provides a reference to the formatter object associated with this field.
DataType.kind	Provides a reference to the Kind object associated with this field.

DataType.encoder

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dataTypeObject.encoder

Description

Property; provides a reference to the encoder object associated with this field, if one exists. You can use this property to access any properties and methods defined by the specific encoder applied to the field in the Component inspector's Schema tab.

If no encoder was applied to the field in question, then this property returns *undefined*.

For more information about the encoders provided with Flash, see "Schema encoders" in *Using Flash*.

Example

The following example assumes that the field being accessed (*isValid*) uses the Boolean encoder (*mx.data.encoders.Boolean*). This encoder is provided with Flash and contains a property named *trueStrings* that specifies which strings should be interpreted as true values. The code below sets the *trueStrings* property for a field's encoder to be the strings "Yes" and "Oui".

```
var myField:mx.data.binding.DataType = dataSet.getField("isValid");
myField.encoder.trueStrings = "Yes,Oui";
```

DataType.formatter

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dataTypeObject.formatter

Description

Property; provides a reference to the formatter object associated with this field, if one exists. You can use this property to access any properties and methods for the formatter object applied to the field in the Component inspector's Schema tab.

If no formatter was applied to the field in question, this property returns *undefined*.

For more information about the formatters provided with Flash, see "Schema formatters" in *Using Flash*.

Example

This example assumes that the field being accessed is using the Number Formatter (`mx.data.formatters.NumberFormatter`) provided with Flash Professional 8. This formatter contains a property named `precision` that specifies how many digits to display after the decimal point. This code sets the `precision` property to two decimal places for a field using this formatter.

```
var myField:DataType = dataGrid.getField("currentBalance");
myField.formatter.precision = 2;
```

DataType.getAnyTypedValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.getAnyTypedValue(suggestedTypes)
```

Parameters

suggestedTypes An array of strings that specify, in descending order of desirability, the preferred data types for the field.

Returns

The current value of the field, in the form of one of the data types specified in the *suggestedTypes* array.

Description

Method; fetches the current value of the field, using the information in the field's schema to process the value. If the field can provide a value as the first data type specified in the *suggestedTypes* array, the method returns the field's value as that data type. If not, the method attempts to extract the field's value as the second data type specified in the *suggestedTypes* array, and so on.

If you specify `null` as one of the items in the *suggestedTypes* array, the method returns the value of the field in the data type specified in the Schema tab of the Component inspector. Specifying `null` always results in a value being returned, so only use `null` at the end of the array.

If a value can't be returned in the form of the one of the suggested types, it is returned in the type specified in the Schema tab.

Example

This example attempts to get the value of a field (`productInfo.available`) in an `XMLConnector` component's `results` property first as a number or, if that fails, as a string.

```
import mx.data.binding.DataType;
import mx.data.binding.TypedValue;
var f: DataType = myXmlConnector.getField("results",
    "productInfo.available");
var b: TypedValue = f.getAnyTypedValue(["Number", "String"]);
```

See also

[ComponentMixins.getField\(\)](#)

DataType.getAsBoolean()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.getAsBoolean()
```

Parameters

None.

Returns

A Boolean value.

Description

Method; fetches the current value of the field and converts it to Boolean form, if necessary.

Example

In this example, a field named `propName` that belongs to a component named `myComponent` is retrieved as a Boolean value, and assigned to a variable:

```
var dataTypeObj:mx.data.binding.DataType =
    myComponent.getField("propName");
var propValue:Boolean = dataTypeObj.getAsBoolean();
```

DataType.getAsNumber()

Availability

Flash Player 6.

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.getAsNumber()
```

Parameters

None.

Returns

A number.

Description

Method; fetches the current value of the field and converts it to Number form, if necessary.

Example

In this example, a field named `propName` that belongs to a component named `myComponent` is retrieved as a number, and assigned to a variable:

```
var dataTypeObj:mx.data.binding.DataType =  
    myComponent.getField("propName");  
var propValue:Number = dataTypeObj.getAsNumber();
```

See also

[DataType.getAnyTypedValue\(\)](#)

DataType.getAsString()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.getAsString()
```


Parameters

None.

Returns

A string.

Description

Method; fetches the current value of the field and converts it to String form, if necessary.

Example

In this example, a property named `propName` that belongs to a component named `myComponent` is retrieved as a string and assigned to a variable:

```
var dataTypeObj:mx.data.binding.DataType =
    myComponent.getField("propName");
var propValue:String = dataTypeObj.getAsString();
```

See also

[DataType.getAnyTypedValue\(\)](#)

DataType.getTypedValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.getTypedValue(requestedType)
```

Parameters

requestedType A string containing the name of a data type, or `null`.

Returns

A `TypedValue` object (see [“TypedValue class \(Flash Professional only\)” on page 245](#)).

Description

Method; returns the value of the field in the specified form, if the field can provide its value in that form. If the field cannot provide its value in the requested form, the method returns `null`.

If `null` is specified as *requestedType*, the method returns the value of the field in its default type.

Example

The following example returns the value of the field converted to the Boolean data type. This is stored in the `bool` variable.

```
var bool:TypedValue = field.getTypedValue("Boolean");
```

DataType.kind

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.kind
```

Description

Property; provides a reference to the Kind object associated with this field. You can use this property to access properties and methods of the Kind object.

DataType.setAnyTypedValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.setAnyTypedValue(newTypedValue)
```

Parameters

newTypedValue A TypedValue object value to set in the field. For more information, see [“TypedValue class \(Flash Professional only\)” on page 245](#).

Returns

An array of strings describing any errors that occurred while attempting to set the new value. Errors can occur under any of the following conditions:

- The data provided cannot be converted to the data type of this field (for example, "abc" cannot be converted to Number).
- The data is an acceptable type but does not meet the validation criteria of the field.
- The field is read-only.

NOTE

The actual text of an error message varies depending on the data type, formatters, and encoders that are defined in the field's schema.

Description

Method; sets a new value in the field, using the information in the field's schema to process the field.

This method operates by first calling `DataType.setTypedValue()` to set the value. If that fails, the method checks to see if the destination object is willing to accept String, Boolean, or Number data, and if so, attempts to use the corresponding ActionScript conversion functions.

Example

The following example creates a new TypedValue object (a Boolean value), and then assigns that value to a DataType object named `field`. Any errors that occur are assigned to the `errors` array.

```
import mx.data.binding.*;
var t:TypedValue = new TypedValue (true, "Boolean");
var errors: Array = field.setAnyTypedValue (t);
```

See also

[DataType.setTypedValue\(\)](#)

DataType.setAsBoolean()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.setAsBoolean(newBooleanValue)
```

Parameters

newBooleanValue A Boolean value.

Returns

Nothing.

Description

Method; sets the field to the new value, which is given as a Boolean value. The value is converted to, and stored as, the data type that is appropriate for this field.

Example

The following example sets a variable named `bool` to the Boolean value `true`. It then sets the value referenced by `field` to `true`.

```
var bool: Boolean = true;
field.setAsBoolean (bool);
```

DataType.setAsNumber()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.setAsNumber(newNumberValue)
```

Parameters

newNumberValue A number.

Returns

Nothing.

Description

Method; sets the field to the new value, which is given as a number. The value is converted to, and stored as, the data type that is appropriate for this field.

Example

The following example sets a variable named `num` to the `Number` value of `32`. It then sets the value referenced by `field` to `num`.

```
var num: Number = 32;
field.setAsNumber (num);
```

DataType.setAsString()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.setAsString(newStringValue)
```

Parameters

newStringValue A string.

Returns

Nothing.

Description

Method; sets the field to the new value, which is given as a string. The value is converted to, and stored as, the data type that is appropriate for this field.

Example

The following example sets the variable `stringValue` to the string "The new value". It then sets the value of `field` to the string.

```
var stringValue: String = "The new value";
field.setAsString (stringValue);
```

DataType.setTypedValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dataTypeObject.setTypedValue(newTypedValue)
```

Parameters

newTypedValue A TypedValue object value to set in the field.

For more information about TypedValue objects, see [“TypedValue class \(Flash Professional only\)” on page 245](#).

Returns

An array of strings describing any errors that occurred while attempting to set the new value. Errors can occur under any of the following conditions:

- The data provided is not an acceptable type.
- The data provided cannot be converted to the data type of this field (for example, "abc" cannot be converted to Number).
- The data is an acceptable type but does not meet the validation criteria of the field.
- The field is read-only.

NOTE

The actual text of an error message varies depending on the data type, formatters, and encoders that are defined in the field's schema.

Description

Method; sets a new value in the field, using the information in the field's schema to process the field. This method behaves similarly to [DataType.setAnyTypedValue\(\)](#), except that it doesn't try as hard to convert the data to an acceptable data type. For more information, see [DataType.setAnyTypedValue\(\)](#).

Example

The following example creates a new `TypedValue` object (a Boolean value), and then assigns that value to a `DataType` object named `field`. Any errors that occur are assigned to the `errors` array.

```
import mx.data.binding.*;
var bool:TypedValue = new TypedValue (true, "Boolean");
var errors: Array = field.setTypedValue (bool);
```

See also

[DataType.setTypedValue\(\)](#)

TypedValue class (Flash Professional only)

ActionScript Class Name `mx.data.binding.TypedValue`

A `TypedValue` object contains a data value, along with information about the value's data type. `TypedValue` objects are provided as parameters to, and are returned from, various methods of the `DataType` class. The data type information in the `TypedValue` object helps `DataType` objects decide when and how they need to do type conversion.

NOTE

To make this class available at runtime, you must include the data binding classes in your FLA file.

For an overview of the classes in the `mx.data.binding` package, see [“Classes in the mx.data.binding package \(Flash Professional only\)”](#) on page 208.

Property summary for the TypedValue class

The following table lists the properties of the `TypedValue` class.

Property	Description
<code>TypedValue.type</code>	Contains the schema associated with the <code>TypedValue</code> object's value.
<code>TypedValue.typeName</code>	Names the data type of the <code>TypedValue</code> object's value.
<code>TypedValue.value</code>	Contains the data value of the <code>TypedValue</code> object.

Constructor for the TypedValue class

Availability

Flash Player 6 (6.0.79.0).

Usage

```
new mx.data.binding.TypedValue(value, typeName, [type])
```

Parameters

value A data value of any type.

typeName A string that contains the name of the value's data type.

type An optional Schema object that describes in more detail the schema of the data. This field is required only in certain circumstances, such as when setting data into a DataSet component's `dataProvider` property.

Description

Constructor; creates a new TypedValue object.

TypedValue.type

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
typedValueObject.type
```

Description

Property; contains the schema associated with the TypedValue object's value.

Example

This example displays `null` in the Output panel:

```
var t: TypedValue = new TypedValue (true, "Boolean", null);  
trace(t.type);
```


TypedValue.typeName

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

typedValueObject.typeName

Description

Property; contains the name of the data type of the TypedValue object's value.

Example

This example displays `Boolean` in the Output panel:

```
var t: TypedValue = new TypedValue (true, "Boolean", null);  
trace(t.typeName);
```

TypedValue.value

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

typedValueObject.value

Description

Property; contains the data value of the TypedValue object.

Example

This example displays `true` in the Output panel:

```
var t: TypedValue = new TypedValue (true, "Boolean", null);  
trace(t.value);
```


DataGrid component (Flash Professional only)

The DataGrid component lets you create powerful data-enabled displays and applications. You can use the DataGrid component to instantiate a recordset (retrieved from a database query in Macromedia ColdFusion, Java, or .Net) using Macromedia Flash Remoting and display it in columns. You can also use data from a data set or from an array to fill a DataGrid component. Version 2 of the DataGrid component has been improved to include horizontal scrolling, better event support (including event support for editable cells), enhanced sorting capabilities, and performance optimizations.

You can resize and customize characteristics such as the font, color, and borders of columns in a grid. You can use a custom movie clip as a cell renderer for any column in a grid. (A cell renderer displays the contents of a cell.) You can use scroll bars to move through data in a grid; you can also turn off scroll bars and use the DataGrid methods to create a page view style display. For more information about customization, see [“DataGridColumn class \(Flash Professional only\)” on page 300](#).

When you add the DataGrid component to an application, you can use the Accessibility panel to make the component accessible to screen readers. First, you must add the following line of code to enable accessibility for the DataGrid component:

```
mx.accessibility.DataGridAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Interacting with the DataGrid component (Flash Professional only)

You can use the mouse and the keyboard to interact with a DataGrid component.

If `DataGrid.sortableColumns` and `DataGridColumn.sortOnHeaderRelease` are both `true`, clicking in a column header causes the grid to sort based on the column's cell values.

If `DataGrid.resizableColumns` is `true`, clicking in the area between columns lets you resize columns.

Clicking in an editable cell sends focus to that cell; clicking a non-editable cell has no effect on focus. An individual cell is editable when both the `DataGrid.editable` and `DataGridColumn.editable` properties of the cell are `true`.

When a DataGrid instance has focus either from clicking or tabbing, you can use the following keys to control it:

Key	Description
Down Arrow	When a cell is being edited, the insertion point shifts to the end of the cell's text. If a cell is not editable, the Down Arrow key handles selection as the List component does.
Up Arrow	When a cell is being edited, the insertion point shifts to the beginning of the cell's text. If a cell is not editable, the Up Arrow key handles selection as the List component does.
Right Arrow	When a cell is being edited, the insertion point shifts one character to the right. If a cell is not editable, the Right Arrow key does nothing.
Left Arrow	When a cell is being edited, the insertion point shifts one character to the left. If a cell is not editable, the Left Arrow key does nothing.
Return/Enter/ Shift+Enter	When a cell is editable, the change is committed, and the insertion point is moved to the cell on the same column, next row (up or down, depending on the shift toggle).
Shift+Tab/Tab	Moves focus to the previous item. When the Tab key is pressed, focus cycles from the last column in the grid to the first column on the next line. When Shift+Tab is pressed, cycling is reversed. All the text in the focused cell is selected.

Using the DataGrid component (Flash Professional only)

You can use the DataGrid component as the foundation for numerous types of data-driven applications. You can easily display a formatted tabular view of a database query (or other data), but you can also use the cell renderer capabilities to build more sophisticated and editable user interface pieces. The following are practical uses for the DataGrid component:

- A webmail client
- Search results pages
- Spreadsheet applications such as loan calculators and tax form applications

Understanding the design of the DataGrid component

The DataGrid component extends the [List component](#). When you design an application with the DataGrid component, it is helpful to understand how the List class underlying it was designed. The following are some fundamental assumptions and requirements that Macromedia used when developing the List class:

- Keep it small, fast, and simple.
Don't make something more complicated than absolutely necessary. This was the prime design directive. Most of the requirements listed below are based on this directive.
- Lists have uniform row heights.
Every row must be the same height; the height can be set during authoring or at runtime.
- Lists must scale to thousands of records.
- Lists don't measure text.

This creates a horizontal scrolling issue for List and Tree components; for more information, see “[Understanding the design of the List component](#)” on page 762. The DataGrid component, however, supports "auto" as an `hScrollPolicy` value, because it measures columns (which are the same width per item), not text.

The fact that lists don't measure text explains why lists have uniform row heights. Sizing individual rows to fit text would require intensive measuring. For example, if you wanted to accurately show the scroll bars on a list with nonuniform row height, you'd need to premeasure every row.

- Lists perform worse as a function of their visible rows.
Although lists can display 5000 records, they can't render 5000 records at once. The more visible rows (specified by the `rowCount` property) you have on the Stage, the more work the list must do to render. Limiting the number of visible rows, if at all possible, is the best solution.
- Lists aren't tables.
DataGrid components are intended to provide an interface for many records. They're not designed to display complete information; they're designed to display enough information so that users can drill down to see more. The message view in Microsoft Outlook is a prime example. You don't read the entire e-mail in the grid; the message would be difficult to read and the client would perform terribly. Outlook displays enough information so that a user can drill into the post to see the details.

Understanding the DataGrid component: data model and view

Conceptually, the DataGrid component is composed of a data model and a view that displays the data. The data model consists of three main parts:

- DataProvider
This is a list of items with which to fill the data grid. Any array in the same frame as a DataGrid component is automatically given methods (from the DataProvider API) that let you manipulate data and broadcast changes to multiple views. Any object that implements the DataProvider API can be assigned to the `DataGrid.dataProvider` property (including recordsets, data sets, and so on). The following code creates a data provider called `myDP`:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel", price:"Cheap"});
```
- Item
This is an ActionScript object used for storing the units of information in the cells of a column. A data grid is really a list that can display more than one column of data. A list can be thought of as an array; each indexed space of the list is an item. For the DataGrid component, each item consists of fields. In the following code, the content between curly braces (`{}`) is an item:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel", price:"Cheap"});
```

- **Field**

Identifiers that indicate the names of the columns within the items. This corresponds to the `columnName` property in the columns list. In the List component, the fields are usually `label` and `data`, but in the DataGrid component the fields can be any identifier.

In the following code, the fields are `name` and `price`:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
price:"Cheap"});
```

The view consists of three main parts:

- **Row**

This is a view object responsible for rendering the items of the grid by laying out cells. Each row is laid out horizontally below the previous one.

- **Column**

Columns are fields that are displayed in the grid; the fields each correspond to the `columnName` property of each column.

Each column is a view object (an instance of the `DataGridColumn` class) responsible for displaying each column—for example, width, color, size, and so on.

There are three ways to add columns to a data grid: assign a `DataProvider` object to `DataGrid.dataProvider` (this automatically generates a column for each field in the first item), set `DataGrid.columnNames` to specify which fields are displayed, or use the constructor for the `DataGridColumn` class to create columns and call `DataGrid.addColumn()` to add them to the grid.

To format columns, either set up style properties for the entire data grid, or define `DataGridColumn` objects, set up their style formats individually, and add them to the data grid.

- **Cell**

This is a view object responsible for rendering the individual fields of each item. To communicate with the data grid, these components must implement the `CellRenderer` API (see “[CellRenderer API](#)” on page 109). For a basic data grid, a cell is a built-in ActionScript `TextField` object.

DataGrid parameters

You can set the following authoring parameters for each DataGrid component instance in the Property inspector or in the Component inspector:

editable is a Boolean value that indicates whether the grid is editable (`true`) or not (`false`). The default value is `false`.

multipleSelection is a Boolean value that indicates whether multiple items can be selected (`true`) or not (`false`). The default value is `false`.

rowHeight indicates the height of each row, in pixels. Changing the font size does not change the row height. The default value is 20.

You can write ActionScript to control these and additional options for the DataGrid component using its properties, methods, and events. For more information, see [“DataGrid class \(Flash Professional only\)” on page 262](#).

Creating an application with the DataGrid component

To create an application with the DataGrid component, you must first determine where your data is coming from. The data for a grid can come from a recordset that is fed from a database query in Macromedia ColdFusion, Java, or .Net using Flash Remoting. Data can also come from a data set or an array. To pull the data into a grid, you set the `DataGrid.dataProvider` property to the recordset, data set, or array. You can also use the methods of the DataGrid and DataGridColumn classes to create data locally. Any Array object in the same frame as a DataGrid component copies the methods, properties, and events of the DataProvider API.

NOTE

When you bind data to the DataGrid component using the Data components, the object binds columns backward (similar to looping over an object or array). Therefore, to order the data in the DataGrid component differently, you must explicitly define columns.

To use Flash Remoting to add a DataGrid component to an application:

1. In Flash, select File > New and select Flash Document.
2. In the Components panel, double-click the DataGrid component to add it to the Stage.
3. In the Property inspector, enter the instance name **myDataGrid**.
4. In the Actions panel on Frame 1, enter the following code:

```
myDataGrid.dataProvider = recordSetInstance;
```

The Flash Remoting recordset `recordSetInstance` is assigned to the `dataProvider` property of `myDataGrid`.

5. Select Control > Test Movie.

To use a local data provider to add a DataGrid component to an application:

1. In Flash, select File > New and then select Flash Document.
2. In the Components panel, double-click the DataGrid component to add it to the Stage.
3. In the Property inspector, enter the instance name **myDataGrid**.

4. In the Actions panel on Frame 1, enter the following code:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
  price:"Cheap"});  
myDataGrid.dataProvider = myDP;
```

The name and price fields are used as the column headings, and their values fill the cells in each row.

5. Select Control > Test Movie.

To specify columns and add sorting for a DataGrid component in an application:

1. In Flash, select File > New and then select Flash Document.
2. In the Components panel, double-click the DataGrid component to add it to the Stage.
3. In the Property inspector, enter the instance name **myDataGrid**.
4. In the Actions panel on Frame 1, enter the following code:

```
var myDataGrid:mx.controls.DataGrid;  
  
// Create columns to enable sorting of data.  
myDataGrid.addColumn("name");  
myDataGrid.addColumn("score");  
  
var myDP_array:Array = new Array({name:"Clark", score:3135},  
  {name:"Bruce", score:403}, {name:"Peter", score:25})  
  
myDataGrid.dataProvider = myDP_array;  
  
// Create listener object for DataGrid.  
var listener_obj:Object = new Object();  
listener_obj.headerRelease = function(evt_obj:Object) {  
  switch (evt_obj.target.columns[evt_obj.columnIndex].columnName) {  
    case "name" :  
      myDP_array.sortOn("name", Array.CASEINSENSITIVE);  
      break;  
    case "score" :  
      myDP_array.sortOn("score", Array.NUMERIC);  
      break;  
  }  
};  
  
// Add listener to DataGrid.  
myDataGrid.addEventListener("headerRelease", listener_obj);
```

5. Select Control > Test Movie.

To create a DataGrid component instance using ActionScript:

1. Drag the DataGrid component from the Components panel to the current document's library.

This adds the component to the library, but doesn't make it visible in the application.

2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.DataGrid, "my_dg", 10,
    {columnNames:["name", "score"]});
my_dg.setSize(140, 100);
my_dg.move(10, 40);
```

This script uses the `UIObject.createClassObject()` method to create the DataGrid instance and then sizes and positions the grid.

3. Create an array, add data to the array, and identify the array as the data provider for the data grid:

```
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;
```

4. Select Control > Test Movie.

DataGrid performance strategies

Performance can quickly become a major concern when you are using the DataGrid component because the size of the data it displays can be scalable. A data grid that displays a hundred rows on a fast computer with a fast connection to the data source may look acceptable to you. A month later, when the data has increased to several thousand rows, a user may have a much different experience. Also, the user may have a slower computer on a slow connection to your data source.

Here are some suggestions for avoiding common performance pitfalls when using the DataGrid component.

- Build and bind a data structure rather than add columns directly.

There are two ways to add columns and data to the DataGrid component: by binding a pre-made data structure (an array of objects) through the `DataGrid.dataProvider` property or by using DataGrid class methods such as `DataGrid.addColumn()` and `DataGrid.addItem()`. Whenever possible, you should bind to a pre-made data structure using the `DataGrid.dataProvider` property because it allows DataGrid to create all the columns it needs before attempting to draw them on the screen.

You may be tempted to make a `for` loop to call `DataGrid.addColumn()` for all the columns needed. Although this seems like a simple and obvious approach, do not use it. Each time that `DataGrid.addColumn()` is called, the data grid adds event listeners, sorts, and redraws itself to present the new column. Creating 20 columns using `DataGrid.addColumn()` causes `DataGrid` to sort itself and redraw 20 times needlessly. Building your data structure in ActionScript requires no rendering or events to account for. When you assign it to the `dataProvider` property of the `DataGrid` component, all of the drawing is completed in just one pass.

- Provide a drill-down mechanism for detailed data.

The `DataGrid` component interface allows users to search quickly so that they can search for more details. Provide only the data needed to perform the initial search, and detailed information for any particular row or cell can be provided in a second search step. This process minimizes not only the initial data required to fill the data grid but also minimizes the amount of information that users must read to locate what they are looking for. When a row or item of interest is selected in the data grid, a second call can be made to the data source to get related details. Those details can be appear better in some other UI mechanism, such as a collection of multiline text fields and graphics.

- Avoid cycles of data manipulation between the data source and the data grid.

If it is possible, and if it meets long-term database needs, storing the data in much the same format and order in which it appears can avoid unnecessary memory allocation and processing time on the user's computer and speed up data-grid response time.

- Avoid queries that return every row in the database.

Users rarely want to see every record that is available every time they access the data. It's important to understand what the consumers of your data are looking for and give them the means to narrow down their search. If they usually look only at the most recent records for a given week for a particular subject, display that smaller group of data as a default, with the ability to widen the view of the data.

Consider paging potentially large amounts of data to limit its size by providing a subset of data that might normally be returned from a query. For instance, rather than viewing all 10,000 rows of data that might be returned by a query from your database, a subset of the first 20 rows might be called for, and additional navigation buttons might trigger a call to fill the data grid with the next 20 records.

- Separate data processing from `CellRenderer` processing.
The `CellRenderer` API lets you display custom cell content in a data grid. A functional requirement might require that you populate the data grid with a `ComboBox` component or other UI control conditionally. For example, based on a selection in column two, you may repopulate or auto-select options in column four. Whenever possible, it is important to separate this conditional logic and repopulating of controls from the process of rendering the content of the cell. Each time the mouse rolls over the cell, the cell is selected, or data is changed, the content of the cell or the entire cell is likely to be redrawn to keep it up to date. This means that any code you put in `CellRenderer` is run over and over again, so you should keep processing in `CellRenderer` as light as possible. If you do have to add code to `CellRenderer`, it is better to call a function from `CellRenderer` that detects what updates need to be made and makes them in the most efficient manner.
- Use `UIObject.doLater()` to access properties after the data grid has finished drawing.
A data grid instance needs to finish drawing and loading data before you can access all the properties of the data grid (such as `focusedCell` and others). Because there is no “complete” event for a `DataGrid`, you can use `UIObject.doLater()`, instead, to call a function that accesses the data grid properties. `UIObject.doLater()` will execute the function only after the data grid properties are available. See `DataGrid.focusedCell` for an example.

Customizing the `DataGrid` component (Flash Professional only)

You can transform a `DataGrid` component horizontally and vertically during authoring and runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see `UIObject.setSize()`). If there is no horizontal scroll bar, column widths adjust proportionally. If column (and therefore, cell) size adjustment occurs, text in the cells may be clipped.

Using styles with the DataGrid component

You can set style properties to change the appearance of a DataGrid component. The DataGrid component inherits styles from the List component. (See [“Using styles with the List component” on page 766.](#)) The DataGrid component also supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>	Both	The background color, which can be set for the whole grid or for each column.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to <code>false</code> . The default value is <code>0xDDDDDD</code> (medium gray).
<code>borderStyle</code>	Both	The DataGrid component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063. The default border style value is <code>inset</code> .
<code>headerColor</code>	Both	The color of the column headers. The default value is <code>0xEAEAEA</code> (light gray)
<code>headerStyle</code>	Both	A CSS style declaration for the column header that can be applied to a grid or column to customize the header styles.
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. For example (using a DataGrid instance <code>my_dg</code>): <pre>my_dg.setStyle("fontFamily", "yourFont"); my_dg.embedFonts=true;</pre> Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .

Style	Theme	Description
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
textAlign	Both	The text alignment: either "left", "right", or "center". The default value is "left".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".
vGridLines	Both	A Boolean value that indicates whether to show vertical grid lines (<code>true</code>) or not (<code>false</code>). The default value is <code>true</code> .
hGridLines	Both	A Boolean value that indicates whether to show horizontal grid lines (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
vGridLineColor	Both	The color of the vertical grid lines. The default value is <code>Ox666666</code> (medium gray).
hGridLineColor	Both	The color of the horizontal grid lines. The default value is <code>Ox666666</code> (medium gray).

Setting styles for an individual column

Color and text styles can be set for the grid as a whole or for a column. You can use the following syntax to set a style for a particular column:

```
grid.getColumnAt(3).setStyle("backgroundColor", 0xFF00AA);
```

Setting header styles

You can set header styles through `headerStyle`, which is a style property itself. To do this, you create an instance of `CSSStyleDeclaration`, set the appropriate properties on that instance for the header, and then assign the `CSSStyleDeclaration` to the `headerStyle` property, as shown in the following example.

```
import mx.styles.CSSStyleDeclaration;
var headerStyles = new CSSStyleDeclaration();
headerStyles.setStyle("fontStyle", "italic");
grid.setStyle("headerStyle", headerStyles);
```

Setting styles for all DataGrid components in a document

The DataGrid class inherits from the List class, which inherits from the ScrollSelectList class. The default class-level style properties are defined on the ScrollSelectList class, which the Menu component and all List-based components extend. You can set new default style values on this class directly, and these new settings are reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the DataGrid components only, you can create a new instance of CSSStyleDeclaration and store it in `_global.styles.DataGrid`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
    _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFF00AA);
```

When you create a new class-level style declaration, you lose all default values provided by the ScrollSelectList declaration, including `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.DataGrid;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

For more information about class-level styles, see “Setting styles for a component class” in *Using Components*.

Using skins with the DataGrid component

The skins that the DataGrid component uses to represent its visual states are included in the subcomponents that constitute the data grid (scroll bars and RectBorder). For information about their skins, see [“Using skins with the UI ScrollBar component” on page 1394](#) and [“RectBorder class” on page 1063](#).

DataGrid class (Flash Professional only)

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `ScrollSelectList` > [List component](#) > `DataGrid`

ActionScript Class Name `mx.controls.DataGrid`

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DataGrid.version);
```

NOTE

The code `trace(myDataGridInstance.version);` returns `undefined`.

Method summary for the DataGrid class

The following table lists methods of the `DataGrid` class.

Method	Description
<code>DataGrid.addColumn()</code>	Adds a column to the data grid.
<code>DataGrid.addColumnAt()</code>	Adds a column to the data grid at a specified location.
<code>DataGrid.addItem()</code>	Adds an item to the data grid.
<code>DataGrid.addItemAt()</code>	Adds an item to the data grid at a specified location.
<code>DataGrid.editField()</code>	Replaces the cell data at a specified location.
<code>DataGrid.getColumnAt()</code>	Gets a reference to a column at a specified location.
<code>DataGrid.getColumnIndex()</code>	Gets a reference to the <code>DataGridColumn</code> object at the specified index.
<code>DataGrid.removeAllColumns()</code>	Removes all columns from a data grid.
<code>DataGrid.removeColumnAt()</code>	Removes a column from a data grid at a specified location.
<code>DataGrid.replaceItemAt()</code>	Replaces an item at a specified location with another item.
<code>DataGrid.spaceColumnsEqually()</code>	Spaces all columns equally.

Methods inherited from the UIObject class

The following table lists the methods the DataGrid class inherits from the UIObject class. When calling these methods, use the form *dataGridInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the DataGrid class inherits from the UIComponent class. When calling these methods, use the form *dataGridInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the List class

The following table lists the methods the DataGrid class inherits from the List class. When calling these methods, use the form *dataGridInstance.methodName*.

Method	Description
<code>List.addItem()</code>	Adds an item to the end of the list.
<code>List.addItemAt()</code>	Adds an item to the list at the specified index.
<code>List.getItemAt()</code>	Returns the item at the specified index.

Method	Description
<code>List.removeAll()</code>	Removes all items from the list.
<code>List.removeItemAt()</code>	Removes the item at the specified index.
<code>List.replaceItemAt()</code>	Replaces the item at the specified index with another item.
<code>List.setPropertiesAt()</code>	Applies the specified properties to the specified item.
<code>List.sortItems()</code>	Sorts the items in the list according to the specified compare function.
<code>List.sortItemsBy()</code>	Sorts the items in the list according to a specified property.

Property summary for the DataGrid class

The following table lists the properties of the DataGrid class.

Property	Description
<code>DataGrid.columnCount</code>	Read-only; the number of columns that are displayed.
<code>DataGrid.columnNames</code>	An array of field names within each item that are displayed as columns.
<code>DataGrid.dataProvider</code>	The data model for a data grid.
<code>DataGrid.editable</code>	A Boolean value that indicates whether the data grid is editable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.focusedCell</code>	Defines the cell that has focus.
<code>DataGrid.headerHeight</code>	The height of the column headers, in pixels.
<code>DataGrid.hScrollPolicy</code>	Indicates whether a horizontal scroll bar is present ("on"), not present ("off"), or appears when necessary ("auto").
<code>DataGrid.resizableColumns</code>	A Boolean value that indicates whether the columns are resizable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.selectable</code>	A Boolean value that indicates whether the data grid is selectable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.showHeaders</code>	A Boolean value that indicates whether the column headers are visible (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.sortableColumns</code>	A Boolean value that indicates whether the columns are sortable (<code>true</code>) or not (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the DataGrid class inherits from the UIObject class. When accessing these properties from the DataGrid object, use the form *dataGridInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the DataGrid class inherits from the UIComponent class. When accessing these properties from the DataGrid object, use the form *dataGridInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the List class

The following table lists the properties the DataGrid class inherits from the List class. When accessing these properties from the DataGrid object, use the form *dataGridInstance.propertyName*.

Property	Description
List.cellRenderer	Assigns the class or symbol to use to display each row of the list.
List.dataProvider	The source of the list items.
List.hPosition	The horizontal position of the list.
List.hScrollPolicy	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
List.iconField	A field in each item to be used to specify icons.
List.iconFunction	A function that determines which icon to use.
List.labelField	Specifies a field of each item to be used as label text.
List.labelFunction	A function that determines which fields of each item to use for the label text.
List.length	The number of items in the list. This property is read-only.
List.maxHPosition	The number of pixels the list can scroll to the right, when List.hScrollPolicy is set to "on".
List.multipleSelection	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
List.rowCount	The number of rows that are at least partially visible in the list.
List.rowHeight	The pixel height of every row in the list.
List.selectable	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).
List.selectedIndex	The index of a selection in a single-selection list.
List.selectedIndices	An array of the selected items in a multiple-selection list.
List.selectedItem	The selected item in a single-selection list. This property is read-only.
List.selectedItems	The selected item objects in a multiple-selection list. This property is read-only.
List.vPosition	Scrolls the list so the topmost visible item is the number assigned.
List.vScrollPolicy	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Event summary for the DataGrid class

The following table lists the events of the DataGrid class.

Event	Description
DataGrid.cellEdit	Broadcast when the cell value has changed.
DataGrid.cellFocusIn	Broadcast when a cell receives focus.
DataGrid.cellFocusOut	Broadcast when a cell loses focus.
DataGrid.cellPress	Broadcast when a cell is pressed (clicked).
DataGrid.change	Broadcast when an item has been selected.
DataGrid.columnStretch	Broadcast when a user resizes a column horizontally.
DataGrid.headerRelease	Broadcast when a user clicks (releases) a header.

Events inherited from the UIObject class

The following table lists the events the DataGrid class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DataGrid class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

Events inherited from the List class

The following table lists the events the DataGrid class inherits from the List class.

Event	Description
List.change	Broadcast whenever user interaction causes the selection to change.
List.itemRollOut	Broadcast when the mouse pointer rolls over and then off of list items.
List.itemRollOver	Broadcast when the mouse pointer rolls over list items.
List.scroll	Broadcast when a list is scrolled.

DataGrid.addColumn()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addColumn(dataGridColumn)
```

```
myDataGrid.addColumn(name)
```

Parameters

dataGridColumn An instance of the DataGridColumn class.

name A string that indicates the name of a new DataGridColumn object to be inserted.

Returns

A reference to the DataGridColumn object that was added, or returns the string that indicates the name of the new column.

Description

Method; adds a new column to the end of the data grid. For more information, see [“DataGridColumn class \(Flash Professional only\)” on page 300](#).

Example

This example shows three different ways of creating columns for a DataGrid component. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline (notice that it imports the `DataGridColumn` class first):

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(320, 240);

// Add columns to grid.
my_dg.addColumn("Red");

// Add another column to grid.
my_dg.addColumn(new DataGridColumn("Green"));

// Add a third column to grid.
var blue_dgc:DataGridColumn = new DataGridColumn("Blue");
blue_dgc.width = 140;
blue_dgc.headerText = "Blue Column:";
my_dg.addColumn(blue_dgc);
```

DataGrid.addColumnAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addColumnAt(index, name)
myDataGrid.addColumnAt(index, dataGridColumn)
```

Parameters

index The index position at which the `DataGridColumn` object is added. The first position is 0.

name A string that indicates the name of the `DataGridColumn` object.

dataGridColumn An instance of the `DataGridColumn` class.

Returns

A reference to the `DataGridColumn` object that was added, or returns the string that indicates the name of the new column.

Description

Method; adds a new column at the specified position. Columns are shifted to the right and their indexes are incremented. For more information, see [“DataGridColumn class \(Flash Professional only\)” on page 300](#).

Example

This example shows two ways to use `addColumnAt()` and sets the column widths. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline (notice that it imports the `DataGridColumn` class first):

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(320, 240);

// Add columns to grid.
my_dg.addColumnAt(0, "Orange");
var orange_dgc:DataGridColumn = my_dg.getColumnAt(0);
orange_dgc.width = 125;

var blue_dgc:DataGridColumn = new DataGridColumn("Blue");
blue_dgc.width = 75;
my_dg.addColumnAt(1, blue_dgc);
```

DataGrid.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addItem(item)
```

Parameters

item An instance of an object to be added to the grid.

Returns

A reference to the instance that was added.

Description

Method; adds an item to the end of the grid (after the last item index).

NOTE

This differs from the `List.addItem()` method in that an object is passed rather than a string.

Example

This example creates one column with the heading “name” and then inserts the `item_obj` value for “name”. Notice that the “age” value is ignored, because only the name column has been defined. If you don’t specify a column (remove the `addColumn` line), `DataGrid` automatically creates the appropriate columns. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Add columns to grid and add data.
my_dg.addColumn("name");

var item_obj:Object = {name:"Jim", age:30};
my_dg.addItem(item_obj);
```

DataGrid.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addItemAt(index, item)
```

Parameters

index The index position (among the child nodes) at which the node should be added. The first position is 0.

item A string that displays the node.

Returns

A reference to the object instance that was added.

Description

Method; adds an item to the grid at the position specified.

Example

This example creates one column with the heading “name”, populates the column from an array, and then adds the name “Chase” in the first row. Notice that the “age” value is ignored, because only the name column has been defined. If you don’t specify a column (remove the `addColumn` line), `DataGrid` automatically creates the appropriate columns. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
var my_dg:mx.controls.DataGrid;

// Add columns to grid and add data.
my_dg.addColumn("name");

var myDP_array:Array = new Array({name:"John", age:33}, {name:"Jose",
    age:41});
my_dg.dataProvider = myDP_array;

var item_obj:Object = {name:"Chase", age:30};
my_dg.addItemAt(0, item_obj);
```

DataGrid.cellEdit

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellEdit = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("cellEdit", listenerObject)
```

Description

Event; broadcast to all registered listeners when cell value changes.

Version 2 Macromedia Component Architecture components use a dispatcher/listener event model. The `DataGrid` component dispatches a `cellEdit` event when the value of a cell has changed, and the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellEdit` event's event object has four additional properties:

`columnIndex` A number that indicates the index of the target column.

`itemIndex` A number that indicates the index of the target row.

`oldValue` The previous value of the cell.

`type` The string "cellEdit".

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `myDataGridListener` is defined and passed to `myDataGrid.addEventListener()` as the second parameter. The event object is captured by the `cellEdit` handler in the *eventObject* parameter. When the `cellEdit` event is broadcast (after you alter a “score” value and press Enter), a trace statement is sent to the Output panel. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(320, 240);
my_dg.editable = true;

// Add columns and make the first one not editable.
my_dg.addColumn("name");
my_dg.getColumnAt(0).editable = false;
my_dg.addColumn("score");

var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

// Set data source of DataGrid.
my_dg.dataProvider = myDP_array;

// Create listener object.
var myListener_obj:Object = new Object();
myListener_obj.cellEdit = function(evt_obj:Object) {
    // Retrieve location of cell that was changed.
    var cell_obj:Object = "("+evt_obj.columnIndex+", "+evt_obj.itemIndex+")";
    // Retrieve cell value that was changed.
    var value_obj:Object = evt_obj.target.selectedItem.score;
    trace("The value of the cell at "+cell_obj+" has changed to "+value_obj);
};

// Add listener object.
my_dg.addEventListener("cellEdit", myListener_obj);
```

DataGrid.cellFocusIn

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellFocusIn = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("cellFocusIn", listenerObject)
```

Description

Event; broadcast to all registered listeners when a particular cell receives focus. This event is broadcast after any previously edited cell's `editCell` and `cellFocusOut` events are broadcast.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `cellFocusIn` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellFocusIn` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the target column.

`itemIndex` A number that indicates the index of the target row.

`type` The string "cellFocusIn".

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `dgListener` is defined and passed to `my_dg.addEventListener()` as the second parameter. When the `cellFocusIn` event is broadcast, a trace statement is sent to the Output panel. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

// Make DataGrid editable.
my_dg.editable = true;

// Create listener object.
var dgListener:Object = new Object();
dgListener.cellFocusIn = function(evt_obj:Object) {
    var cell_str:String = "(" + evt_obj.columnIndex + ", " +
        evt_obj.itemIndex + ")";
    trace("The cell at " + cell_str + " has gained focus");
};

// Add listener.
my_dg.addEventListener("cellFocusIn", dgListener);
```

NOTE

The grid must be editable for this code to work, and the event is broadcast only for editable cells. So if you have two columns and only one of them is editable (for example, "score"), then clicking in a row in the "name" column would not trigger this event.

DataGrid.cellFocusOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellFocusOut = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("cellFocusOut", listenerObject)
```

Description

Event; broadcast to all registered listeners whenever a user moves off a cell that has focus. You can use the event object properties to isolate the cell that was left. This event is broadcast after the `cellEdit` event and before any subsequent `cellFocusIn` events are broadcast by the next cell.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `cellFocusOut` event, the event is handled by a function (also called a *handler*) that is attached to a listener object that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellFocusOut` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the target column. The first position is 0.

`itemIndex` A number that indicates the index of the target row. The first position is 0.

`type` The string "cellFocusOut".

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `dgListener` is defined and passed to `my_dg.addEventListener()` as the second parameter. When the `cellFocusOut` event is broadcast, a trace statement is sent to the Output panel. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

// Make DataGrid editable.
my_dg.editable = true;
```

```

// Create listener object.
var dgListener:Object = new Object();
dgListener.cellFocusOut = function(evt_obj:Object) {
    var cell_str:String = "(" + evt_obj.columnIndex + ", " +
        evt_obj.itemIndex + ")";
    trace("The cell at " + cell_str + " has lost focus");
};

// Add listener.
my_dg.addEventListener("cellFocusOut", dgListener);

```

NOTE

The grid must be editable for this code to work, and the event is broadcast only for editable cells. If you have two columns and only one of them is editable (for example, "score"), clicking out of a row in the "name" column does not trigger this event.

DataGrid.cellPress

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```

listenerObject = new Object();
listenerObject.cellPress = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("cellPress", listenerObject)

```

Description

Event; broadcast to all registered listeners when a user presses the mouse button on a cell.

Version 2 components use a dispatcher/listener event model. When a DataGrid component broadcasts a `cellPress` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellPress` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the column that was pressed. The first position is 0.

`itemIndex` A number that indicates the index of the row that was pressed. The first position is 0.

`type` The string "cellPress".

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `dgListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `cellPress` handler in the `evt_obj` parameter. When the `cellPress` event is broadcast, a trace statement is sent to the Output panel. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Set up sample data.
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce",
    score:403}, {name:"Peter", score:25}];

// Create listener object.
var dgListener:Object = new Object();
dgListener.cellPress = function(evt_obj:Object) {
    var cell_str:String = ("+"+evt_obj.columnIndex+", "+"+evt_obj.itemIndex+"");
    trace("The cell at "+cell_str+" has been clicked");
};

// Add listener.
my_dg.addEventListener("cellPress", dgListener);
```

DataGrid.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when an item has been selected.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a change event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.change` event's event object has one additional property, `type`, whose value is "change". For more information, see ["EventDispatcher class" on page 499](#).

Example

In the following example, a handler called `dgListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `change` handler in the `evt_obj` parameter. When the change event is broadcast, a trace statement is sent to the Output panel. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Set up sample data.
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce",
    score:403}, {name:"Peter", score:25}];

// Create listener object.
var dgListener:Object = new Object();
dgListener.change = function(evt_obj:Object) {
    trace("The selection has changed to " + evt_obj.target.selectedIndex);
};

// Add listener.
my_dg.addEventListener("change", dgListener);
```

DataGrid.columnCount

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.columnCount
```

Description

Property (read-only); the number of columns displayed.

Example

The following example displays the total number of columns in the Output panel. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
// Add columns to grid and add data.  
my_dg.addColumn("a");  
my_dg.addColumn("b");  
  
my_dg.addItem({a:"one", b:"two"});  
  
// Get number of columns in grid.  
var colCount_num:Number = my_dg.columnCount;  
trace("Number of columns: "+colCount_num);
```

DataGrid.columnNames

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.columnNames
```

Description

Property; an array of field names within each item that are displayed as columns.

Example

The following example displays the column name in the Output panel when the title is clicked. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(200, 100);
my_dg.columnNames = ["Name", "Description", "Price"];

var dgListener:Object = new Object();
dgListener.headerRelease = function (evt_obj:Object) {
    trace("You clicked on the \" + my_dg.columnNames[evt_obj.columnIndex] +
        "\" column.");
}
my_dg.addEventListener("headerRelease", dgListener);
```

DataGrid.columnStretch

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.columnStretch = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("columnStretch", listenerObject)
```

Description

Event; broadcast to all registered listeners when a user resizes a column horizontally.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `columnStretch` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.columnStretch` event's event object has two additional properties:

`columnIndex` A number that indicates the index of the target column. The first position is 0.

`type` The string "columnStretch".

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example displays the column index number in the Output panel when the title is resized. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(240, 100);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({id:0, name:"Clark", score:3135});
myDP_array.push({id:1, name:"Bruce", score:403});
myDP_array.push({id:2, name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

// Create listener object.
var dgListener:Object = new Object();
dgListener.columnStretch = function(evt_obj:Object) {
    trace("column " + evt_obj.columnIndex + " was resized");
};

// Add listener.
my_dg.addEventListener("columnStretch", dgListener);
```

DataGrid.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.dataProvider
```

Description

Property; the data model for items viewed in a DataGrid component.

The data grid adds methods to the prototype of the Array class so that each Array object conforms to the DataProvider API (see `DataProvider.as` in the `Classes/mx/controls/listclasses` folder). Any array that is in the same frame or screen as a data grid automatically has all the methods (`addItem()`, `getItemAt()`, and so on) needed for it to be the data model of a data grid, and can be used to broadcast data model changes to multiple components.

In a DataGrid component, you specify fields for display in the `DataGrid.columnNames` property. If you don't define the column set (by setting the `DataGrid.columnNames` property or by calling `DataGrid.addColumn()`) for the data grid before the `DataGrid.dataProvider` property has been set, the data grid generates columns for each field in the data provider's first item, once that item arrives.

Any object that implements the DataProvider API can be used as a data provider for a data grid (including Flash Remoting recordsets, data sets, and arrays). For example, see ["DataSet.dataProvider" on page 353](#).

Use a grid's data provider to communicate with the data in the grid because the data provider remains consistent, regardless of scroll position.

Example

The following example creates an array to be used as a data provider and assigns it directly to the `dataProvider` property:

```
my_dg.dataProvider = [{name:"Chris", price:"Priceless"}, {name:"Nigel",  
    price:"cheap"}];
```

The following example creates a new Array object that is decorated with the DataProvider API. It uses a `for` loop to add 20 items to the grid:

```
var myDP:Array = new Array();  
for (var i=0; i<20; i++)  
    myDP.addItem({id:i, name:"Dave", price:"Priceless"});  
my_dg.dataProvider = myDP
```

DataGrid.editable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`myDataGrid.editable`

Description

Property; determines whether the data grid can be edited by a user (`true`) or not (`false`). This property must be `true` in order for individual columns to be editable and for any cell to receive focus. The default value is `false`.

If you want individual columns to be uneditable, use the [DataGridColumn.editable](#) property.

CAUTION

The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable. For more information, see Chapter 16, “Data Integration (Flash Professional Only),” in *Using Flash*.

Example

The following example allows users to edit all the columns of the grid except the first column. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);

// Add columns to grid and add data.
my_dg.addColumn("a");
my_dg.addColumn("b");
my_dg.addItem({a:"one", b:1});
my_dg.addItem({a:"two", b:2});

// Make DataGrid editable.
my_dg.editable = true;
// Make the first column read-only.
my_dg.getColumnAt(0).editable = false;
```

See also

[DataGridColumn.editable](#)

DataGrid.editField()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.editField(index, colName, data)
```

Parameters

index The index of the target cell. This number is zero-based.

colName A string indicating the name of the column (field) that contains the target cell.

data The value to be stored in the target cell. This parameter can be of any data type.

Returns

The data that was in the cell.

Description

Method; replaces the cell data at the specified location and refreshes the data grid with the new value. Any cell present for that value has its `setValue()` method triggered.

Example

The following example places a value in the grid in the first row of the first column (index value 0) when the button is clicked. With a `DataGrid` instance named `my_dg` and a `Button` instance named `my_btn` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);

// Set up sample data.
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce",
    score:403}, {name:"Peter", score:25}];

// Create listener object.
var btnListener:Object = new Object();
btnListener.click = function() {
    //Replace first field with new values.
    my_dg.editField(0, "name", "Arthur");
};

// Add button listener.
my_btn.addEventListener("click", btnListener);
```

DataGrid.focusedCell

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.focusedCell

Description

Property; in editable mode only, an object instance that defines the cell that has focus. The object must have the fields `columnIndex` and `itemIndex`, which are both integers that indicate the index of the column and item of the cell. The origin is (0,0). The default value is `undefined`.

Example

The following example sets the focused cell to the second column, eleventh row (numbered “10” because the first row is “0”). Because you can’t access the cells until the DataGrid has finished drawing, use `UIObject.doLater()` to delay using the `focusedCell` property:

```
// Create a data provider with three columns and 50 rows.
var myDP:Array = new Array();
for (var i=0; i<50; i++)
    myDP.addItem({id:i, name:"Dave", price:"Priceless"});

// Assign the data provider to the DataGrid instance and set it to be
    editable.
my_dg.dataProvider = myDP;
my_dg.editable = true;

// Use UIObject.doLater() in the current timeline to call the function after
    the data grid has set all of its properties.
my_dg.doLater(this, "select");

function select() {
    my_dg.focusedCell = {columnIndex:1, itemIndex:10};
}
```


DataGrid.getColumnAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index)
```

Parameters

index The index of the DataGridColumn object to be returned. This number is zero-based.

Returns

A DataGridColumn object.

Description

Method; gets a reference to the DataGridColumn object at the specified index.

Example

The following example gets the DataGridColumn object at index 0 and changes the text. With a DataGrid instance named `my_dg` and a Button instance named `my_btn` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);

// Set up sample data.
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce",
    score:403}, {name:"Peter", score:25}];

// Create listener object.
var btnListener:Object = new Object();
btnListener.click = function() {
    // Get column at location 0.
    var a_dgc = my_dg.getColumnAt(0);
    // Change header text.
    a_dgc.headerText = "c";
};

// Add button listener.
my_btn.addEventListener("click", btnListener);
```

DataGrid.getColumnIndex()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnIndex(columnName)
```

Parameters

columnName A string that is the name of a column.

Returns

A number that specifies the index of the column.

Description

Method; returns the index of the column specified by the *columnName* parameter.

Example

The following example displays the index number of the “score” column. With a DataGrid instance named *my_dg* on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(150, 100);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

var column_num:Number = my_dg.getColumnIndex("score");
trace("Column that has name of 'score': " + column_num);
```

DataGrid.headerHeight

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.headerHeight

Description

Property; the height of the header bar of the data grid, in pixels. The default value is 20.

Example

The following example sets the height of the header bar to 40. With a DataGrid instance named *my_dg* on the Stage, paste the following code in the first frame of the main timeline:

```
// Set grid attributes.
my_dg.setSize(240, 100);
my_dg.spaceColumnsEqually();

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

my_dg.headerHeight = 40;
```

DataGrid.headerRelease

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.headerRelease = function(eventObject){
    // Insert your code here.
}
myDataGridInstance.addEventListener("headerRelease", listenerObject)
```

Description

Event; broadcast to all registered listeners when a column header has been released. You can use this event with the `DataGridColumn.sortOnHeaderRelease` property to prevent automatic sorting and to let you sort as you like.

Version 2 components use a dispatcher/listener event model. When the `DataGrid` component dispatches a `headerRelease` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.headerRelease` event's event object has two additional properties:

`columnIndex` A number that indicates the index of the target column.

`type` The string "headerRelease".

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `headerRelease` handler in the *eventObject* parameter. When the `headerRelease` event is broadcast, a trace statement is sent to the Output panel.

```
var myListener = new Object();
myListener.headerRelease = function(event) {
    trace("column " + event.columnIndex + " header was pressed");
};
grid.addEventListener("headerRelease", myListener);
```

In the following example, you change the sort direction using a column. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
var my_dg:mx.controls.DataGrid;
my_dg.setSize(150, 100);
my_dg.spaceColumnsEqually();
var myListener:Object = new Object();
myListener.headerRelease = function(evt:Object) {
    trace("column "+evt.columnIndex+" header was pressed");
    trace("\t current sort order is: "+evt.target.sortDirection);
    trace("");
};
my_dg.addEventListener("headerRelease", myListener);

my_dg.addColumn("a");
my_dg.addColumn("b");
my_dg.addItem({a:'one', b:1});
my_dg.addItem({a:'two', b:2});
```

By accessing the `sortDirection` property, you can tell whether the sort order is ascending or descending. The `sortDirection` property is a string, so it traces as either `ASC` or `DESC`.

DataGrid.hScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.hScrollPolicy

Description

Property; specifies whether the data grid has a horizontal scroll bar. This property can have the value `"on"`, `"off"`, or `"auto"`. The default value is `"off"`.

If `hScrollPolicy` is set to `"off"`, columns scale proportionally to accommodate the finite width.

NOTE

This differs from the `List` component, which cannot have `hScrollPolicy` set to `"auto"`.

Example

The following example sets horizontal scroll policy to automatic, which means that the horizontal scroll bar appears if it's necessary to display all the content:

```
my_dg.setSize(150, 100);

// Add columns to grid and add data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

my_dg.hScrollPolicy = "on";
```

DataGrid.removeAllColumns()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.removeAllColumns()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all `DataGridColumn` objects from the data grid. Calling this method has no effect on the data provider.

Call this method if you are setting a new data provider that has different fields from the previous data provider, and you want to clear the fields that are displayed.

Example

The following example removes all `DataGridColumn` objects from the `DataGrid` when the button is clicked. With a `DataGrid` instance named `my_dg` and a `Button` instance named `clear_button` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

this.createClassObject(mx.controls.Button, "clear_button", 20,
    {label:"Clear"});
clear_button.move(10, 10);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

var buttonListener:Object = new Object();
buttonListener.click = function (evt_obj:Object) {
    my_dg.removeAllColumns();
}
clear_button.addEventListener("click", buttonListener);
```

DataGrid.removeColumnAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.removeColumnAt(index)
```

Parameters

index The index of the column to remove.

Returns

A reference to the `DataGridColumn` object that was removed.

Description

Method; removes the `DataGridColumn` object at the specified index.

Example

The following example removes the first `DataGridColumn` object when the button is clicked. With a `DataGrid` instance named `my_dg` and a `Button` instance named `name_button` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

name_button.setSize(140, name_button.height);
name_button.move(10, 10);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// Create listener object.
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    my_dg.removeColumnAt(my_dg.getColumnIndex("name"));
    evt_obj.target.enabled = false;
};
// Add button listener.
name_button.addEventListener("click", buttonListener);
```

DataGrid.replaceItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.replaceItemAt(index, item)
```

Parameters

index The index of the item to be replaced.

item An object that is the item value to use as a replacement.

Returns

The previous value.

Description

Method; replaces the item at a specified index and refreshes the display of the grid.

Example

The following example replaces the item at row index 2 with new entries. With a `DataGrid` instance named `my_dg` and a `Button` instance named `replace_button` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

replace_button.move(10, 10);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// Create listener object.
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    //Replace previous value
    var prevValue_obj:Object = my_dg.replaceItemAt(2, {name:"Frank",
        score:949});
    my_dg.selectedIndex = 2;
};
// Add button listener.
replace_button.addEventListener("click", buttonListener);
```

DataGrid.resizableColumns

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.resizableColumns

Description

Property; a Boolean value that determines whether the columns of the grid can be stretched by the user (`true`) or not (`false`). This property must be `true` for individual columns to be resizable by the user. The default value is `true`.

Example

The following example prevents users from resizing columns. With a `DataGrid` instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// Don't allow columns to be resizable.
my_dg.resizableColumns = false;
```

DataGrid.selectable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.selectable
```

Description

Property; a Boolean value that determines whether a user can select the data grid (`true`) or not (`false`). The default value is `true`. If `false`, an item in the grid does not remain selected when the user clicks the item and moves the pointer.

Example

The following example prevents the grid from being selected. With a DataGrid instance named `my_dg` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.setSize(140, 100);

// Set up sample data.
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

my_dg.selectable = false;
```

DataGrid.showHeaders

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.showHeaders

Description

Property; a Boolean value that indicates whether the data grid displays the column headers (`true`) or not (`false`). Column headers are shaded to differentiate them from the other rows in a grid. Users can click column headers to sort the contents of the column if [DataGrid.sortableColumns](#) is set to `true`. The default value of `showHeaders` is `true`.

Example

The following example hides the column headers:

```
my_dg.setSize(140, 100);

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't show headers.
my_dg.showHeaders = false;
```

See also

[DataGrid.sortableColumns](#)

DataGrid.sortableColumns

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.sortableColumns

Description

Property; a Boolean value that determines whether the columns of the data grid can be sorted (`true`) or not (`false`) when a user clicks the column headers. This property must be `true` for individual columns to be sortable, and for the `headerRelease` event to be broadcast. The default value is `true`.

CAUTION

The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable. For more information, see Chapter 16, “Data Integration (Flash Professional Only),” in *Using Flash*.

Example

The following example turns off sorting:

```
my_dg.setSize(140, 100);

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't allow columns to be sorted.
my_dg.sortableColumns = false;
```

See also

[DataGrid.headerRelease](#)

DataGrid.spaceColumnsEqually()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.spaceColumnsEqually()
```

Parameters

None.

Returns

Nothing.

Description

Method; respaces the columns equally.

Example

The following example respaces the columns of `my_dg` when the button is clicked. With a `DataGrid` instance named `my_dg` and a `Button` instance named `resize_button` on the Stage, paste the following code in the first frame of the main timeline:

```
my_dg.move(10, 40);
my_dg.setSize(200, 100);

resize_button.move(10, 10);
resize_button.setSize(200, resize_button.height);

my_dg.addColumn("guitar");
my_dg.addColumn("name");

// Set up sample data.
my_dg.addItem({guitar:"Flying V", name:"maggot"});
my_dg.addItem({guitar:"SG", name:"dreschie"});
my_dg.addItem({guitar:"jagstang", name:"vitapup"});

// Create listener object.
var buttonListener:Object = new Object();
buttonListener.click = function() {
    my_dg.spaceColumnsEqually();
};
// Add button listener.
resize_button.addEventListener("click", buttonListener);
```

DataGridColumn class (Flash Professional only)

ActionScript Class Name mx.controls.gridclasses.DataGridColumn

You can create and configure DataGridColumn objects to use as columns of a data grid. Many of the methods of the DataGrid class are dedicated to managing DataGridColumn objects.

DataGridColumn objects are stored in an zero-based array in the data grid; 0 is the leftmost column. After columns have been added or created, you can access them by calling

`DataGrid.getColumnAt(index)`.

There are three ways to add or create columns in a grid. If you want to configure your columns, it is best to use either the second or third way before you add data to a data grid so you don't have to create columns twice.

- Add a data provider or an item with multiple fields to a grid that has no configured DataGridColumn objects. This approach automatically generates columns for every field in the reverse order of the `for...in` loop. For example, for a DataGrid instance named

`my_dg`:

```
my_dg.dataProvider = [{guitar:"Flying V", name:"maggot"}, {guitar:"SG", name:"dreschie"}, {guitar:"jagstang", name:"vitapup"}];
```

- Use [DataGrid.columnNames](#) to create the field names of the desired item fields and generate DataGridColumn objects, in order, for each field listed. This approach lets you select and order columns quickly with a minimal amount of configuration. This approach removes any previous column information. For example, for a DataGrid instance named

`my_dg`:

```
my_dg.columnNames = ["guitar","name"];
```

- Prebuild the DataGridColumn objects and add them to the data grid by using [DataGrid.addColumn\(\)](#). This approach is useful, and the most flexible, because it lets you add columns with proper sizing and formatting before the columns ever reach the grid (which reduces processor demand). For more information, see [“Constructor for the DataGridColumn class” on page 302](#). For example, for a DataGrid instance named

`my_dg`:

```
// Create column object.
var location_dgc:DataGridColumn = new DataGridColumn("Location");
location_dgc.width = 100;
// Add column to DataGrid.
my_dg.addColumn(location_dgc);
```

Property summary for the DataGridColumn class

The following table lists the properties of the DataGridColumn class.

Property	Description
DataGridColumn.cellRenderer	The linkage identifier of a symbol to be used to display the cells in this column.
DataGridColumn.columnName	Read-only; the name of the field associated with the column.
DataGridColumn.editable	A Boolean value that indicates whether a column is editable (<code>true</code>) or not (<code>false</code>).
DataGridColumn.headerRenderer	The name of a class to be used to display the header of this column.
DataGridColumn.headerText	The text for the header of this column.
DataGridColumn.labelFunction	A function that determines which field of an item to display.
DataGridColumn.resizable	A Boolean value that indicates whether a column is resizable (<code>true</code>) or not (<code>false</code>).
DataGridColumn.sortable	A Boolean value that indicates whether a column is sortable (<code>true</code>) or not (<code>false</code>).
DataGridColumn.sortOnHeaderRelease	A Boolean value that indicates whether a column is sorted (<code>true</code>) or not (<code>false</code>) when a user clicks a column header.
DataGridColumn.width	The width of a column, in pixels.

Constructor for the DataGridColumn class

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
new DataGridColumn(name)
```

Parameters

name A string that indicates the name of the DataGridColumn object. This parameter is the field of each item to display.

Returns

Nothing.

Description

Constructor; creates a DataGridColumn object. Use this constructor to create columns to add to a DataGrid component. After you create the DataGridColumn objects, you can add them to a data grid by calling [DataGrid.addColumn\(\)](#).

Example

The following example creates a DataGridColumn object called `Location`:

```
import mx.controls.gridclasses.DataGridColumn;  
var column = new DataGridColumn("Location");
```


DataGridColumn.cellRenderer

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).cellRenderer
```

Description

Property; a linkage identifier for a symbol to be used to display cells in this column. Any class used for this property must implement the CellRenderer API (see [“CellRenderer API” on page 109.](#)) The default value is undefined.

Example

The following example uses a linkage identifier to set a new cell renderer:

```
myGrid.getColumnAt(3).cellRenderer = "MyCellRenderer";
```

DataGridColumn.columnName

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).columnName
```

Description

Property (read-only); the name of the field associated with this column. The default value is the name called in the DataGridColumn constructor.

Example

The following example displays the name of the column as index position 1:

```
import mx.controls.gridclasses.DataGridColumn;
// Set grid attributes.
my_dg.setSize(150, 100);

// Add columns to grid.
var name_dgc:DataGridColumn = my_dg.addColumn(new DataGridColumn("name"));
name_dgc.headerText = "Name:";
var score_dgc:DataGridColumn = my_dg.addColumn(new
    DataGridColumn("score"));
score_dgc.headerText = "Score:";

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Get column name.
var name_str:String = my_dg.getColumnAt(1).columnName;
trace(name_str);
```

See also

[Constructor for the DataGridColumn class](#)

DataGridColumn.editable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).editable
```

Description

Property; determines whether the column can be edited by a user (`true`) or not (`false`). The `DataGrid.editable` property must be `true` in order for individual columns to be editable, even when `DataGridColumn.editable` is set to `true`. The default value is `true`.

CAUTION

The `DataGrid` is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the `DataGrid` component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable. For more information, see Chapter 16, “Data Integration (Flash Professional Only),” in *Using Flash*.

Example

The following example prevents items in the first column in a grid from being edited:

```
// Set grid attributes.
my_dg.setSize(150, 100);
my_dg.editable = true;

// Add columns to grid.
my_dg.addColumn("name");
my_dg.addColumn("score");

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't allow first column to be editable.
my_dg.getColumnAt(0).editable = false;
```

See also

[DataGrid.editable](#)

DataGridColumn.headerRenderer

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).headerRenderer
```

Description

Property; a string that indicates a class name to be used to display the header of this column. Any class used for this property must implement the CellRenderer API (see [“CellRenderer API” on page 109](#)). The default value is `undefined`.

Example

The following example uses a linkage identifier to set a new header renderer:

```
myGrid.getColumnAt(3).headerRenderer = "MyHeaderRenderer";
```

DataGridColumn.headerText

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).headerText
```

Description

Property; the text in the column header. The default value is the column name.

This property allows you to display something other than the field name as the header.

Example

The following example sets the column header text to “Price (USD)”:

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;

var price_dgc:DataGridColumn = new DataGridColumn("price");
price_dgc.headerText = "Price (USD)";
price_dgc.width = 80;
my_dg.addColumn(price_dgc);

my_dg.addItem({price:"$14.99"});
```

DataGridColumn.labelFunction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).labelFunction
```

Description

Property; specifies a function to determine which field (or field combination) of each item to display. This function receives one parameter, *item*, which is the item being rendered, and must return a string representing the text to display. This property can be used to create virtual columns that have no equivalent field in the item.

NOTE

The specified function operates in a nondefined scope.

Example

The following example calculates a value for the “Subtotal” column:

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(300, 200);

// Set up columns.
var guitar_dgc:DataGridColumn = new DataGridColumn("guitar");
var value_dgc:DataGridColumn = new DataGridColumn("value");
var tax_dgc:DataGridColumn = new DataGridColumn("tax");
var st_dgc:DataGridColumn = new DataGridColumn("Subtotal");
//Define labelFunction for Subtotal column.
st_dgc.labelFunction = function(item:Object):String {
    if ((item.value != undefined) && (item.tax != undefined)) {
        return "$"+(item.value+item.tax);
    }
};

// Add columns to grid.
my_dg.addColumn(guitar_dgc);
my_dg.addColumn(value_dgc);
my_dg.addColumn(tax_dgc);
my_dg.addColumn(st_dgc);

// Set data model.
my_dg.addItem({guitar:"Flying V", value:10, tax:1});
my_dg.addItem({guitar:"SG", value:20, tax:2});
my_dg.addItem({guitar:"jagstang", value:30, tax:3});
```

DataGridColumn.resizable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).resizable
```

Description

Property; a Boolean value that indicates whether a column can be resized by a user (`true`) or not (`false`). The `DataGrid.resizableColumns` property must be set to `true` for this property to take effect. The default value is `true`.

Example

The following example prevents the column at index 0 from being resized:

```
// Set grid attributes.
my_dg.setSize(150, 100);
my_dg.addColumn("name");
my_dg.addColumn("score");

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't allow resize of the first column
my_dg.getColumnAt(0).resizable = false;
```

DataGridColumn.sortable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).sortable
```

Description

Property; a Boolean value that indicates whether a column can be sorted by a user (`true`) or not (`false`). The `DataGrid.sortableColumns` property must be set to `true` for this property to take effect. The default value is `true`.

CAUTION

The DataGrid is not editable or sortable if it is bound directly to a WebServiceConnector component or an XMLConnector component. You must bind the DataGrid component to the DataSet component and bind the DataSet component to the WebServiceConnector component or XMLConnector component if you want the grid to be editable or sortable. For more information, see Chapter 16, "Data Integration (Flash Professional Only)," in *Using Flash*.

Example

The following example prevents the column at index 1 from being sorted:

```
// Set grid attributes.
my_dg.setSize(150, 100);

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't allow sort of the second column.
my_dg.getColumnAt(1).sortable = false;
```

DataGridColumn.sortOnHeaderRelease

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).sortOnHeaderRelease
```

Description

Property; a Boolean value that indicates whether the column is sorted automatically (`true`) or not (`false`) when a user clicks a header. This property can be set to `true` only if `DataGridColumn.sortable` is set to `true`. If `DataGridColumn.sortOnHeaderRelease` is set to `false`, you can catch the `headerRelease` event and perform your own sort.

The default value is `true`.

CAUTION

The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable. For more information, see Chapter 16, "Data Integration (Flash Professional Only)," in *Using Flash*.

Example

The following example disables sorting of the second column:

```
// Set grid attributes.
my_dg.setSize(150, 100);

// Set up sample data.
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// Don't allow sort of the second column by clicking the header.
my_dg.getColumnAt(1).sortOnHeaderRelease = false;
```

DataGridColumn.width

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).width
```

Description

Property; a number that indicates the width of the column, in pixels. The default value is 50.

Example

The following example makes the width of the first column 50 pixels:

```
// Create new DataProvider component.
var myDP_array:Array = new Array({name:"Chris", price:"Priceless"},
    {name:"Nigel", price:"Cheap"});
//Assign DataProvider to DataGrid.
my_dg.dataProvider = myDP_array;

// Alter DataGrid dimensions.
my_dg.setSize(140, 100);
my_dg.rowHeight = 30;
my_dg.getColumnAt(0).width = 50;
```


DataHolder component (Flash Professional only)

The DataHolder component is a repository for data and a means of generating events when that data has changed. Its main purpose is to hold data and act as a connector between other components that use data binding.

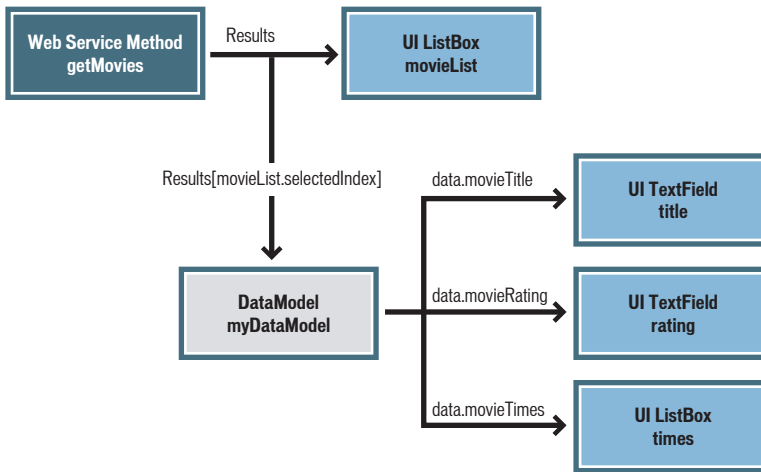
Initially, the DataHolder component has a single bindable property named `data`. You can add more properties by using the Schema tab in the Component inspector. For more information on using the Schema tab, see “Working with schemas in the Schema tab (Flash Professional only)” in *Using Flash*.

You can assign any type of data to a DataHolder property, either by creating a binding between the data and another property, or by using your own ActionScript code. Whenever the value of that data changes, the DataHolder component emits an event whose name is the same as the property, and any bindings associated with that property are executed.

In most cases, you do not use this component to build an application. It is needed only when you cannot bind external data directly to another component and you do not want to use a DataSet component. The DataHolder component is useful when you can't directly bind components (such as connectors, user interface components, or DataSet components) together. Below are some scenarios in which you might use a DataHolder component:

- If a data value is generated by ActionScript, you might want to bind it to some other components. In this case, you could have a DataHolder component that contains properties that are bound as desired. Whenever new values are assigned to those properties (by means of ActionScript, for example) those values are distributed to the data-bound object.

- You might have a data value that results from a complex indexed data binding, as shown in the following diagram.



In this case it is convenient to bind the data value to a DataHolder component (called *DataModel* in this illustration) and then use that for bindings to the user interface.

NOTE

The DataHolder component is not meant to implement the same control over your data as the DataSet component. It does not manage or track data, nor does it have the ability to update data. It is a repository for holding data and generating events when that data has changed.

Creating an application with the DataHolder component (Flash Professional only)

In this example, you add an array property to a DataHolder component's schema (an array) whose value is determined by ActionScript code that you write. You then bind that array property to the `dataProvider` property of a DataGrid component by using the Bindings tab in the Component inspector.

To use the DataHolder component in a simple application:

1. In Flash, create a new file.
2. Open the Components panel, drag a DataHolder component to the Stage, and name it `dataHolder`.

3. Drag a DataGrid component to the Stage and name it **namesGrid**.
4. Select the DataHolder component and open the Component inspector.
5. Click the Schema tab in the Component inspector.
6. Click the Add Component Property (+) button located in the top pane of the Schema tab.
7. In the bottom pane of the Schema tab, type **namesArray** in the Field Name field, and select Array from the Data Type pop-up menu.
8. Click the Bindings tab in the Component inspector, and add a binding between the `namesArray` property of the DataHolder component and the `dataProvider` property of the DataGrid component.

For more information on creating bindings with the Bindings tab, see “Working with bindings in the Bindings tab (Flash Professional only)” in *Using Flash*.

9. In the Timeline, select the first frame on Layer 1 and open the Actions panel.
10. Enter the following code in the Actions panel:

```
dataHolder.namesArray = [{name:"Tim"},{name:"Paul"},{name:"Jason"}];
```

This code populates the `namesArray` array with several objects. When this variable assignment executes, the binding that you established previously between the DataHolder component and the DataGrid component executes.

11. Test the file by selecting Control > Test Movie.

DataHolder class

Inheritance MovieClip > DataHolder

ActionScript class name mx.data.components.DataHolder

The DataHolder component is a repository for data and a means of generating events when that data has changed. Its main purpose is to hold data and act as a connector between other components that use data binding.

Initially, the DataHolder component has a single bindable property named `data`. You can add more properties by using the Schema tab in the Component inspector.

Property summary for the DataHolder class

The following table lists the properties of the DataHolder class.

Property	Description
<code>DataHolder.data</code>	Default bindable property for the DataHolder component.

DataHolder.data

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dataHolder.data

Description

Property; the default item in a DataHolder object's schema. This property is not a "permanent" member of the DataHolder component. Rather, it is the default bindable property for each instance of the component. You can add your own bindable properties, or delete the default `data` property, by using the Schema tab in the Component inspector.

For more information on using the Schema tab, see "Working with schemas in the Schema tab (Flash Professional only)" in *Using Flash*.

Example

For a step-by-step example of using this component, see ["Creating an application with the DataHolder component \(Flash Professional only\)" on page 314](#).

The following code shows a simple example of how to populate the DataHolder component with data that is a variable. To test the application, you enter a value into the text input field and click the `addDate_btn` instance, which adds the value to the DataHolder component. Click the `dumpDataHolder_btn` instance to trace the contents of the DataHolder component.

```
// Drag two Button components onto the Stage (addDate_btn and
dumpDataHolder_btn), a TextInput (myDate_txt) and a DataHolder
(myDataHolder). Add the following ActionScript to Frame 1:

var dhListener:Object = new Object();
dhListener.click = function() {
    trace("dumping DataHolder");
    trace(" " + myDataHolder.myDate);
    trace("");
};
var dateListener:Object = new Object();
dateListener.click = function() {
    myDataHolder.myDate = myDate_txt.text;
    trace("added value");
};
this.dumpDataHolder_btn.addEventListener("click", dhListener);
this.addDate_btn.addEventListener("click", dateListener);
```

The DataProvider API is a set of methods and properties that a data source needs so that a list-based class can communicate with it. Arrays, recordsets, and data sets implement this API. You can create a DataProvider-compliant class by implementing all the methods and properties described in this section. A list-based component could then use that class as a data provider.

DataProvider class

ActionScript Class Name `mx.controls.listclasses.DataProvider`

The methods of the DataProvider class let you query and modify the data in any component that displays data (also called a *view*). The DataProvider API also broadcasts `change` events when the data changes. Multiple views can use the same data provider and receive the `change` events.

A data provider is a linear collection (like an array) of items. Each item is an object composed of many fields of data. You can access these items by index (as you can with an array), using `DataProvider.getItemAt()`.

Data providers are most commonly used with arrays. Data-aware components apply all the methods of the DataProvider API to `Array.prototype` when an Array object is in the same frame or screen as a data-aware component. This lets you use any existing array as the data for views that have a `dataProvider` property.

Because of the DataProvider API, the version 2 Macromedia Component Architecture components that provide views for data (DataGrid, List, Tree, and so on) can also display Flash Remoting RecordSet objects and data from the DataSet component. The DataProvider API is the language with which data-aware components communicate with their data providers.

In the Macromedia Flash documentation, “DataProvider” is the name of the class, `dataProvider` is a property of each component that acts as a view for data, and “data provider” is the generic term for a data source.

Method summary for the DataProvider API

The following table lists the methods of the DataProvider API.

Method	Description
<code>DataProvider.addItem()</code>	Adds an item at the end of the data provider.
<code>DataProvider.addItemAt()</code>	Adds an item to the data provider at the specified position.
<code>DataProvider.editField()</code>	Changes one field of the data provider.
<code>DataProvider.getEditingData()</code>	Gets the data for editing from a data provider.
<code>DataProvider.getItemAt()</code>	Gets a reference to the item at a specified position.
<code>DataProvider.getItemID()</code>	Returns the unique ID of the item.
<code>DataProvider.removeAll()</code>	Removes all items from a data provider.
<code>DataProvider.removeItemAt()</code>	Removes an item from a data provider at a specified position.
<code>DataProvider.replaceItemAt()</code>	Replaces the item at a specified position with another item.
<code>DataProvider.sortItems()</code>	Sorts the items in the data provider according to a compare function or sort options.
<code>DataProvider.sortItemsBy()</code>	Sorts the items in the data provider alphabetically or numerically, in the specified order, using the specified field name.

Property summary for the DataProvider API

The following table lists the properties of the DataProvider API.

Property	Description
<code>DataProvider.length</code>	The number of items in a data provider.

Event summary for the DataProvider API

The following table lists the events of the DataProvider API.

Event	Description
<code>DataProvider.modelChanged</code>	Broadcast when the data provider is changed.

DataProvider.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.addItem(item)
```

Parameters

item An object that contains data. This constitutes an item in a data provider.

Returns

Nothing.

Description

Method; adds a new item at the end of the data provider. This method triggers the `modelChanged` event with the event name `addItem`.

Example

The following example adds an item to the end of the data provider `myDP`:

```
myDP.addItem({label : "this is an Item"});
```

DataProvider.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.addItemAt(index, item)
```

Parameters

index A number greater than or equal to 0. This number indicates the position at which to insert the item; it is the index of the new item.

item An object containing the data for the item.

Returns

Nothing.

Description

Method; adds a new item to the data provider at the specified index. Indices greater than the data provider's length are ignored.

This method triggers the `modelChanged` event with the event name `addItem`.

Example

The following example adds an item to the data provider `myDP` at the fourth position:

```
myDP.addItemAt(3, {label : "this is the fourth Item"});
```

DataProvider.editField()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.editField(index, fieldName, newData)
```

Parameters

index A number greater than or equal to 0; the index of the item.

fieldName A string indicating the name of the field to modify in the item.

newData The new data to put in the data provider.

Returns

Nothing.

Description

Method; changes one field of the data provider.

This method triggers the `modelChanged` event with the event name `updateField`.

Example

The following code modifies the `label` field of the third item:

```
myDP.editField(2, "label", "mynewData");
```

DataProvider.getEditingData()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.getEditingData(index, fieldName)
```

Parameters

index A number greater than or equal to 0 and less than `DataProvider.length`. This number is the index of the item to retrieve.

fieldName A string indicating the name of the field being edited.

Returns

The editable formatted data to be used.

Description

Method; retrieves data for editing from a data provider. This lets the data model provide different formats of data for editing and displaying.

Example

The following code gets an editable string for the price field:

```
trace(myDP.getEditingData(4, "price");
```

DataProvider.getItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.getItemAt(index)
```

Parameters

index A number greater than or equal to 0 and less than `DataProvider.length`. This number is the index of the item to retrieve.

Returns

A reference to the retrieved item; undefined if the index is out of range.

Description

Method; retrieves a reference to the item at a specified position.

Example

The following code displays the label of the fifth item:

```
trace(myDP.getItemAt(4).label);
```

DataProvider.getItemID()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.getItemID(index)
```

Parameters

index A number greater than or equal to 0.

Returns

A number that is the unique ID of the item.

Description

Method; returns a unique ID for the item. This method is primarily used to track selection. The ID is used in data-aware components to keep lists of what items are selected.

Example

This example gets the ID of the fourth item:

```
var ID = myDP.getItemID(3);
```

DataProvider.length

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myDP.length

Description

Property (read-only); the number of items in the data provider.

Example

This example sends the number of items in the `myArray` data provider to the Output panel:

```
trace(myArray.length);
```

DataProvider.modelChanged

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.modelChanged = function(eventObject){  
    // Insert your code here.  
}  
myMenu.addEventListener("modelChanged", listenerObject)
```

Description

Event; broadcast to all of its view listeners whenever the data provider is modified. You typically add a listener to a model by assigning its `dataProvider` property.

Version 2 components use a dispatcher/listener event model. When a data provider changes in some way, it broadcasts a `modelChanged` event, and data-aware components catch it to update their displays to reflect the changes in data.

The `Menu.modelChanged` event's event object has five additional properties:

- `eventName` The `eventName` property is used to subcategorize `modelChanged` events. Data-aware components use this information to avoid completely refreshing the component instance (view) that is using the data provider. The `eventName` property supports the following values:
 - `updateAll` The entire view needs refreshing, excluding scroll position.
 - `addItem` A series of items has been added.
 - `removeItems` A series of items has been deleted.
 - `updateItems` A series of items needs refreshing.
 - `sort` The data has been sorted.
 - `updateField` A field in an item must be changed and needs refreshing.
 - `updateColumn` An entire field's definition in the data provider needs refreshing.
 - `filterModel` The model has been filtered, and the view needs refreshing (reset the scroll position).
 - `schemaLoaded` The field's definition of the data provider has been declared.
- `firstItem` The index of the first affected item.
- `lastItem` The index of the last affected item. The value equals `firstItem` if only one item is affected.
- `removedIDs` An array of the item identifiers that were removed.
- `fieldName` A string indicating the name of the field that is affected.

For more information, see [“EventDispatcher class” on page 499](#).

Example

In the following example, a handler called `listener` is defined and passed to `addEventListener()` as the second parameter. The event object is captured by the `modelChanged` handler in the `evt` parameter. When the `modelChanged` event is broadcast, a trace statement is sent to the Output panel.

```
listener = new Object();
listener.modelChanged = function(evt){
    trace(evt.eventName);
}
myList.addEventListener("modelChanged", listener);
```

DataProvider.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.removeAll();
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items in the data provider. This method triggers the `modelChanged` event with the event name `removeItems`.

Example

This example removes all the items in the data provider:

```
myDP.removeAll();
```

DataProvider.removeItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.removeItemAt(index)
```

Parameters

index A number greater than or equal to 0. This number is the index of the item to remove.

Returns

Nothing.

Description

Method; removes the item at the specified index. The indices after the removed index collapse by one.

This method triggers the `modelChanged` event with the event name `removeItems`.

Example

This example removes the item at the fourth position:

```
myDP.removeItemAt(3);
```

DataProvider.replaceItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDP.replaceItemAt(index, item)
```

Parameters

index A number greater than or equal to 0. This number is the index of the item to change.

item An object that is the new item.

Returns

Nothing.

Description

Method; replaces the content of the item at the specified index. This method triggers the `modelChanged` event with the event name `updateItems`.

Example

This example replaces the item at index 3 with the item labeled “new label”:

```
myDP.replaceItemAt(3, {label : "new label"});
```


DataProvider.sortItems()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myDP.sortItems([compareFunc], [optionsFlag])
```

Parameters

compareFunc A reference to a function that compares two items to determine their sort order. For more information, see `sort` (`Array.sort` method) in *ActionScript 2.0 Language Reference*. This parameter is optional.

optionsFlag Lets you perform multiple, different types of sorts on a single array without having to replicate the entire array or resort it repeatedly. This parameter is optional.

The following are possible values for *optionsFlag*:

- `Array.DECENDING`, which sorts highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts case-insensitively.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the data provider according to the specified compare function or according to one or more specified sort options.

This method triggers the `modelChanged` event with the event name `sort`.

Example

This example sorts according to uppercase labels. The items `a` and `b` are passed to the function and contain `label` and `data` fields:

```
myList.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
    return a.label.toUpperCase() > b.label.toUpperCase();
}
```

DataProvider.sortItemsBy()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myDP.sortItemsBy(fieldName, optionsFlag)
```

```
myDP.sortItemsBy(fieldName, order)
```

Parameters

fieldName A string that specifies the name of the field to use for sorting. This value is usually "label" or "data".

order A string that specifies whether to sort the items in ascending order ("ASC") or descending order ("DESC").

optionsFlag Lets you perform multiple, different types of sorts on a single array without having to replicate the entire array or resort it repeatedly. This parameter is optional.

The following are possible values for *optionsFlag*:

- `Array.DECENDING`—sorts highest to lowest.
- `Array.CASEINSENSITIVE`—sorts case-insensitively.
- `Array.NUMERIC`—sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).

- `Array.UNIQUESORT`—if two objects in the array are identical or have identical sort fields, this method returns an error code (0) instead of a sorted array.
- `Array.RETURNINDEXEDARRAY`—returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the data provider in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but advanced programmers may specify any primitive value.

This method triggers the `modelChanged` event with the event name `sort`.

This is the fastest way to sort data in a component. It also maintains the component's selection state. The `sortItemsBy()` method is fast because it doesn't run any ActionScript while sorting. The `sortItems()` method needs to run an ActionScript compare function, and is therefore slower.

Example

The following code sorts the items in a list in ascending order using the labels of the list items:

```
myDP.sortItemsBy("label", "ASC");
```


DataSet component (Flash Professional only)

The DataSet component lets you work with data as collections of objects that can be indexed, sorted, searched, filtered, and modified.

The DataSet component functionality includes DataSetIterator, a set of methods for traversing and manipulating a data collection, and DeltaPacket, a set of interfaces and classes for working with updates to a data collection. In most cases, you don't use these classes and interfaces directly; you use them indirectly through methods provided by the DataSet class.

The items managed by the DataSet component are also called *transfer objects*. A transfer object exposes business data that resides on the server with public attributes or accessor methods for reading and writing data. The DataSet component allows developers to work with sophisticated client-side objects that mirror their server-side counterparts or, in its simplest form, a collection of anonymous objects with public attributes that represent the fields in a record of data. For details on transfer objects, see Core J2EE Patterns Transfer Object at <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html>.

NOTE

The DataSet component requires Flash Player 7 or later.

Using the DataSet component

You typically use the DataSet component in combination with other components to manipulate and update a data source: a connector component for connecting to an external data source, user interface components for displaying data from the data source, and a resolver component for translating updates made to the data set into the appropriate format for sending to the external data source. You can then use data binding to bind properties of these different components together.

The DataSet component uses functionality in the data binding classes. If you intend to work with the DataSet component in ActionScript only, without using the Binding and Schema tabs in the Component inspector to set properties, you'll need to import the data binding classes into your FLA file and set required properties in your code. See [“Making data binding classes available at runtime \(Flash Professional only\)” on page 207](#).

For general information on how to manage data in Flash using the DataSet component, see [“Data management \(Flash Professional only\)” in *Using Flash*](#).

DataSet parameters

You can set the following parameters for the DataSet component:

itemClassName is a string indicating the name of the transfer object class that is instantiated each time a new item is created in the DataSet component.

The DataSet component uses transfer objects to represent the data that you retrieve from an external data source. If you leave this parameter blank, the data set creates an anonymous transfer object for you. If you give this parameter a value, the data set instantiates your transfer object whenever new data is added.

NOTE

You must make a fully qualified reference to this class somewhere in your code to make sure that it gets compiled into your application (such as `private var myItem:my.package.myItem;`).

logChanges is a Boolean value that defaults to `true`. If this parameter is set to `true`, the data set logs all changes made to its data and any method calls made on the associated transfer objects.

readOnly is a Boolean value that defaults to `false`. If this parameter is set to `true`, the data set cannot be modified.

You can write ActionScript code to use the properties, methods, and events of the DataSet component to control these and additional options. For more information, see [“DataSet class \(Flash Professional only\)” on page 335](#).

Common workflow for the DataSet component

The typical workflow for the DataSet component is as follows.

To use a DataSet component:

1. Add an instance of the DataSet component to your application and give it an instance name.
2. Select the Schema tab for the DataSet component and create component properties to represent the persistent fields of the data set.
3. Load the DataSet component with data from an external data source. (For more information, see “About loading data into the DataSet component” in *Using Flash*.)
4. Use the Bindings tab of the Component inspector to bind the data set fields to user interface components in your application.

The UI controls are notified as records (transfer objects) are selected or modified within the DataSet component, and updated accordingly. In addition, the DataSet component is notified of changes made from within a UI control; those changes are tracked by the data set and can be extracted by means of a delta packet.

5. Call the methods of the DataSet component in your application to manage your data.

NOTE

In addition to these steps, you can bind the DataSet component to a connector and a resolver component to provide a complete solution for accessing, managing, and updating data from an external data source.

Creating an application with the DataSet component

Typically, you use the DataSet component with other user interface components, and often with a connector component such as XMLConnector or WebServiceConnector. The items in the data set are populated by means of the connector component or raw ActionScript data, and then bound to user interface controls (such as List or DataGrid components).

The DataSet component uses functionality in the data binding classes. If you intend to work with the DataSet component in ActionScript only, without using the Binding and Schema tabs in the Component inspector to set properties, you'll need to import the data binding classes into your FLA file and set required properties in your code. See [“Making data binding classes available at runtime \(Flash Professional only\)”](#) on page 207.

To create an application using the DataSet component:

1. In Flash Professional 8, select File > New. In the Type column, select Flash Document and click OK.
2. Open the Components panel if it's not already open.
3. Drag a DataSet component from the Components panel to the Stage. In the Property inspector, give it the instance name `user_ds`.
4. Drag a DataGrid component to the Stage and give it the instance name `user_dg`.
5. Resize the DataGrid component to be approximately 300 pixels wide and 100 pixels tall.
6. Drag a Button component to the Stage and give it the instance name `next_button`.
7. In the Timeline, select the first frame on Layer 1 and open the Actions panel.
8. Add the following code to the Actions panel:

```
var recData_array:Array = [{id:0, firstName:"Mick", lastName:"Jones"},
    {id:1, firstName:"Joe", lastName:"Strummer"},
    {id:2, firstName:"Paul", lastName:"Simonon"}];
user_ds.items = recData_array;
```

This populates the DataSet object's `items` property with an array of objects, each of which has three properties: `id`, `firstName`, and `lastName`.

9. Add the three properties and their required data types to the DataSet schema:
 - a. Select the DataSet component on the Stage, open the Component inspector, and click the Schema tab.
 - b. Click Add Component Property, and add three new properties, with field names `id`, `firstName`, and `lastName`, and data types `Number`, `String`, and `String`, respectively.

Or, if you prefer to add the properties and their required data types in code, you can add the following line of code to the Actions panel instead of following steps a and b above:

```
// Add required schema types.
var i:mx.data.types.Str;
var j:mx.data.types.Num;
```

10. To bind the contents of the DataSet component to the contents of the DataGrid component, open the Component inspector and click the Bindings tab.
11. Select the DataGrid component (`user_dg`) on the Stage, and click the Add Binding (+) button in the Component inspector.
12. In the Add Binding dialog box, select "dataProvider : Array" and click OK.
13. Double-click the Bound To field in the Component inspector.
14. In the Bound To dialog box that appears, select "DataSet <user_ds>" from the Component Path column and then select "dataProvider : Array" from the Schema Location column.

15. To bind the selected index of the DataSet component to the selected index of the DataGrid component, select the DataGrid component on the Stage and click the Add Binding (+) button again in the Component inspector.
16. In the dialog box that appears, select “selectedIndex : Number”. Click OK.
17. Double-click the Bound To field in the Component inspector to open the Bound To dialog box.
18. In the Component Path field, select “DataSet <user_ds>” from the Component Path column and then select “selectedIndex : Number” from the Schema Location column.
19. Enter the following code in the Actions panel:

```
next_button.addEventListener("click", nextBtnClick);
function nextBtnClick(evt_obj:Object):Void {
    user_ds.next();
}
```

This code uses the `DataSet.next()` method to navigate to the next item in the DataSet object’s collection of items. Since you had previously bound the `selectedIndex` property of the DataGrid object to the same property of the DataSet object, changing the current item in the DataSet object changes the current (selected) item in the DataGrid object as well.

20. Save the file, and select Control > Test Movie to test the SWF file.

The DataGrid object is populated with the specified items. Notice how clicking the button changes the selected item in the DataGrid object.

DataSet class (Flash Professional only)

Inheritance MovieClip > DataSet

ActionScript Class Name mx.data.components.DataSet

The DataSet component lets you work with data as collections of objects that can be indexed, sorted, searched, filtered, and modified.

The DataSet component functionality includes DataSetIterator, a set of methods for traversing and manipulating a data collection, and DeltaPacket, a set of interfaces and classes for working with updates to a data collection. In most cases, you don’t use these classes and interfaces directly; you use them indirectly through methods provided by the DataSet class.

Method summary for the DataSet class

The following table lists the methods of the DataSet class.

Method	Description
<code>DataSet.addItem()</code>	Adds the specified item to the collection.
<code>DataSet.addItemAt()</code>	Adds an item to the data set at the specified position.
<code>DataSet.addSort()</code>	Creates a new sorted view of the items in the collection.
<code>DataSet.applyUpdates()</code>	Signals that the <code>deltaPacket</code> property has a value that you can access using data binding or <code>ActionScript</code> .
<code>DataSet.changesPending()</code>	Indicates whether the collection has changes pending that have not yet been sent in a delta packet.
<code>DataSet.clear()</code>	Clears all items from the current view of the collection.
<code>DataSet.createItem()</code>	Returns a newly initialized collection item.
<code>DataSet.disableEvents()</code>	Stops sending DataSet events to listeners.
<code>DataSet.enableEvents()</code>	Resumes sending DataSet events to listeners.
<code>DataSet.find()</code>	Locates an item in the current view of the collection.
<code>DataSet.findFirst()</code>	Locates the first occurrence of an item in the current view of the collection.
<code>DataSet.findLast()</code>	Locates the last occurrence of an item in the current view of the collection.
<code>DataSet.first()</code>	Moves to the first item in the current view of the collection.
<code>DataSet.getItemId()</code>	Returns the unique ID for the specified item.
<code>DataSet.getIterator()</code>	Returns a clone of the current iterator.
<code>DataSet.getLength()</code>	Returns the number of items in the data set.
<code>DataSet.hasNext()</code>	Indicates whether the current iterator is at the end of its view of the collection.
<code>DataSet.hasPrevious()</code>	Indicates whether the current iterator is at the beginning of its view of the collection.
<code>DataSet.hasSort()</code>	Indicates whether the specified sort exists.
<code>DataSet.isEmpty()</code>	Indicates whether the collection contains any items.
<code>DataSet.last()</code>	Moves to the last item in the current view of the collection.
<code>DataSet.loadFromSharedObj()</code>	Loads all of the relevant data needed to restore the DataSet collection from a shared object.
<code>DataSet.locateById()</code>	Moves the current iterator to the item with the specified ID.

Method	Description
<code>DataSet.next()</code>	Moves to the next item in the current view of the collection.
<code>DataSet.previous()</code>	Moves to the previous item in the current view of the collection.
<code>DataSet.removeAll()</code>	Removes all the items from the collection.
<code>DataSet.removeItem()</code>	Removes the specified item from the collection.
<code>DataSet.removeItemAt()</code>	Removes a data set item at a specified position.
<code>DataSet.removeRange()</code>	Removes the current iterator's range settings.
<code>DataSet.removeSort()</code>	Removes the specified sort from the <code>DataSet</code> object.
<code>DataSet.saveToSharedObj()</code>	Saves the data in the <code>DataSet</code> object to a shared object.
<code>DataSet.setIterator()</code>	Sets the current iterator for the <code>DataSet</code> object.
<code>DataSet.setRange()</code>	Sets the current iterator's range settings.
<code>DataSet.skip()</code>	Moves forward or backward by a specified number of items in the current view of the collection.
<code>DataSet.useSort()</code>	Makes the specified sort the active one.

Property summary for the `DataSet` class

The following table lists the properties of the `DataSet` class.

Property	Description
<code>DataSet.currentItem</code>	Returns the current item in the collection.
<code>DataSet.dataProvider</code>	Returns the data provider.
<code>DataSet.deltaPacket</code>	Returns changes made to the collection, or assigns changes to be made to the collection.
<code>DataSet.filtered</code>	Indicates whether items are filtered.
<code>DataSet.filterFunc</code>	User-defined function for filtering items in the collection.
<code>DataSet.items</code>	Items in the collection.
<code>DataSet.itemClassName</code>	Name of the object to create when assigning items.
<code>DataSet.length</code>	Specifies the number of items in the current view of the collection.
<code>DataSet.logChanges</code>	Indicates whether changes made to the collection, or its items, are recorded.
<code>DataSet.properties</code>	Contains the properties (fields) for any transfer object in this collection.

Property	Description
<code>DataSet.readOnly</code>	Indicates whether the collection can be modified.
<code>DataSet.schema</code>	Specifies the collection's schema in XML format.
<code>DataSet.selectedIndex</code>	Contains the current item's index in the collection.

Event summary for the DataSet class

The following table lists the events of the DataSet class.

Event	Description
<code>DataSet.addItem</code>	Broadcast before an item is added to the collection.
<code>DataSet.afterLoaded</code>	Broadcast after the <code>items</code> property is assigned.
<code>DataSet.calcFields</code>	Broadcast when calculated fields should be updated.
<code>DataSet.deltaPacketChanged</code>	Broadcast when the DataSet object's delta packet has been changed and is ready to be used.
<code>DataSet.iteratorScrolled</code>	Broadcast when the iterator's position is changed.
<code>DataSet.modelChanged</code>	Broadcast when items in the collection have been modified in some way.
<code>DataSet.newItem</code>	Broadcast when a new transfer object is constructed by the DataSet object, but before it is added to the collection.
<code>DataSet.removeItem</code>	Broadcast before an item is removed.
<code>DataSet.resolveDelta</code>	Broadcast when a delta packet is assigned to the DataSet object that contains messages.

DataSet.addItem

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.addItem = function (eventObj:Object) {
```

```
    // ...
};
dataSetInstance.addEventListener("addItem", listenerObject);
```

Usage 2:

```
on (addItem) {
    // ...
}
```

Description

Event; generated just before a new record (transfer object) is inserted into this collection.

If you set the `result` property of the event object to `false`, the add operation is canceled; if you set it to `true`, the add operation is allowed.

The event object (*eventObj*) contains the following properties:

`target` The DataSet object that generated the event.

`type` The string "addItem".

`item` A reference to the item in the collection to be added.

`result` A Boolean value that specifies whether the specified item should be added. By default, this value is `true`.

Example

The following `addItem` event handler cancels the addition of the new item if a user-defined function named `userHasAdminPrivs()` returns `false`; otherwise, the item addition is allowed.

```
function userHasAdminPrivs():Boolean {
    return false; // Change this to true to allow inserts.
}

my_ds.addEventListener("addItem", addItemListener);
my_ds.addItem({name:"Bobo", occupation:"clown"});

function addItemListener(evt_obj:Object):Void {
    if (userHasAdminPrivs()) {
        // Allow the item addition.
        evt_obj.result = true;
        trace("Item added");
    } else {
        // Don't allow item addition; user doesn't have admin privileges.
        evt_obj.result = false;
        trace("Error, insufficient permissions");
    }
}
```

See also

[DataSet.removeItem](#)

DataSet.addItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.addItem([obj])
```

Parameters

obj An object to add to this collection. This parameter is optional.

Returns

A Boolean value: `true` if the item was added to the collection, `false` if it was not.

Description

Method; adds the specified record (transfer object) to the collection for management. The newly added item becomes the current item of the data set. If no *obj* parameter is specified, a new object is created automatically by means of [DataSet.createItem\(\)](#).

The location of the new item in the collection depends on whether a sort has been specified for the current iterator. If no sort is in use, the item is added to the end of the collection. If a sort is in use, the item is added to the collection according to its position in the current sort.

For more information on initialization and construction of the transfer object, see [DataSet.createItem\(\)](#).

Example

The following example uses `DataSet.addItem()` to create a new item and add it to the data set:

```
my_ds.addEventListener("addItem", addItemListener);
my_ds.addItem({name:"Bobo", occupation:"clown"});

function addItemListener(evt_obj:Object):Void {
    trace("adding item");
}
```

The following example demonstrates how you can accept or reject an item's insertion into the DataSet by setting the result to `true` or `false` within the handler for the `addItem` event. Drag a DataSet component to the Stage, and assign it an instance name of `my_ds`. Drag a DataGrid component to the Stage, and give it an instance name of `my_dg`. Drag a CheckBox component to the Stage, and give it an instance name of `my_ch`. Drag a Button component to the Stage, and give it an instance name of `submit_button`. Add two properties, `name` and `occupation`, to the DataSet component by using the Schema tab of the Component inspector. Create a binding between the `my_ds.dataProvider` property and the `my_dg.dataProvider` property by using the Bindings panel of the Component inspector. Add the following ActionScript to Frame 1 of the main timeline:

```
my_ds.addEventListener("addItem", addItemListener);
submit_button.addEventListener("click", submitListener);
function userHasAdminPrivs():Boolean {
    return my_ch.selected;
}
function addItemListener(evt_obj:Object):Void {
    if (userHasAdminPrivs()) {
        // Allow the item addition.
        evt_obj.result = true;
        trace("Item added");
    } else {
        // Don't allow the item addition; user doesn't have admin privileges.
        evt_obj.result = false;
        trace("Error, insufficient permissions");
    }
}
function submitListener(evt_obj:Object):Void {
    my_ds.addItem({name:"bobo", occupation:"clown"});
}
```

See also

[DataSet.createItem\(\)](#)

DataSet.addItemAt()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.addItemAt(index, item)
```

Parameters

index A number greater than or equal to 0. This number indicates the position at which to insert the item; it is the index of the new item.

item An object containing the data for the item.

Returns

A Boolean value indicating whether the item was added: `true` indicates that the item was added, and `false` indicates that the item already exists in the data set.

Description

Method; adds a new item to the data set at the specified index. Indices greater than the data provider's length are ignored.

This method triggers the `modelChanged` event with the event type `addItem`.

Example

The following example uses the `addItemAt()` method to add an item to the `DataSet` at the first position:

```
my_ds.addItem({name:"Milton", years:3});  
my_ds.addItem({name:"Mark", years:3});  
my_ds.addItem({name:"Sarah", years:1});  
my_ds.addItem({name:"Michael", years:2});  
my_ds.addItem({name:"Frank", years:2});  
my_ds.addItemAt(0, {name:"Bobo", years:1});
```


DataSet.addSort()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.addSort(name, fieldList, sortOptions)
```

Parameters

name A string that specifies the name of the sort.

fieldList An array of strings that specify the field names to sort on.

sortOptions One or more of the following integer (constant) values, which indicate what options are used for this sort. Separate multiple values using the bitwise OR operator (|).

Specify one or more of the following values:

- `DataSetIterator.Ascending` Sorts items in ascending order. This is the default sort option, if none is specified.
- `DataSetIterator.Descending` Sorts items in descending order based on item properties specified.
- `DataSetIterator.Unique` Prevents the sort if any fields have like values.
- `DataSetIterator.CaseInsensitive` Ignores case when comparing two strings during the sort operation. By default, sorts are case-sensitive when the property being sorted on is a string.

A `DataSetError` exception is thrown when `DataSetIterator.Unique` is specified as a sort option and the data being sorted is not unique, when the specified sort name has already been added, or when a property specified in the *fieldList* array does not exist in this data set.

Returns

Nothing.

Description

Method; creates a new ascending or descending sort for the current iterator based on the properties specified by the *fieldList* parameter. Flash automatically assigns the new sort to the current iterator after it is created, and then stores it in the sorting collection for later retrieval.

Example

The following code creates a new sort named "nameSort" that performs a descending, case-insensitive sort on the DataSet object's "name" field.

```
import mx.data.components.datasetclasses.DataSetIterator;

my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addSort("nameSort", ["name"], DataSetIterator.Descending |
    DataSetIterator.Unique | DataSetIterator.CaseInsensitive);
```

In the following example, you can dynamically add data to a DataSet component by entering first and last names into TextInput component instances on the Stage and clicking the Submit button. After you add items to the DataSet component, you can clear the data set by clicking the Clear button on the Stage. Drag a DataGrid component to the Stage, and give it an instance name of `my_dg`. Drag two Button components to the Stage, and give them instance names of `submit_button` and `clear_button`. Drag a DataSet component to the Stage, and give it an instance name of `my_ds`. Drag two TextInput components to the Stage, and give them instance names of `firstName_ti` and `lastName_ti`. Drag an Alert component into the current document's library. Add two component properties, `firstName` and `lastName`, to the DataSet component's schema by using the Schema tab of the Component inspector. Next, add a data binding from the `dataProvider` property of the DataSet component to the `dataProvider` property of the DataGrid component by using the Binding tab in the Component inspector. Finally, paste the following code in the first frame of the main timeline:

```
import mx.controls.Alert;

my_ds.addSort("lastFirst", ["lastName", "firstName"]);

my_dg.enabled = false;
clear_button.enabled = false;
submit_button.label = "Submit";
clear_button.label = "Clear";

my_ds.addEventListener("addItem", addItemListener);
my_ds.addEventListener("modelChanged", modelChangedListener);
submit_button.addEventListener("click", submitListener);
clear_button.addEventListener("click", clearListener);
```

```

function modelChangeListener(evt_obj:Object):Void {
    my_dg.enabled = (evt_obj.target.length > 0);
    clear_button.enabled = my_dg.enabled;
}
function submitListener(evt_obj:Object):Void {
    my_ds.addItem({firstName:firstName_ti.text, lastName:lastName_ti.text});
}
function addItemListener(evt_obj:Object):Void {
    if ((evt_obj.item.firstName.length == 0) || (evt_obj.item.lastName.length
    == 0)) {
        Alert.show("Error, first name or last name cannot be blank.", "Error",
        Alert.OK, _level0);
        evt_obj.result = false;
    } else {
        firstName_ti.text = "";
        lastName_ti.text = "";
    }
}
function clearListener(evt_obj:Object):Void {
    Alert.show("Are you sure you want to clear the data?", "Warning",
    Alert.OK | Alert.CANCEL, _level0, clearConfirmListener);
}
function clearConfirmListener(evt_obj:Object):Void {
    switch (evt_obj.detail) {
    case Alert.OK:
        my_ds.clear();
        break;
    case Alert.CANCEL:
        break;
    }
}
}

```

Select Control > Test Movie to test the document in the authoring environment. Enter some text into both of the TextInput instances and then click the Submit button. A new item should be added to the DataGrid instance. Clicking the Clear button should display an Alert component instance with a confirmation message asking if you want to clear the contents of the DataGrid. Clicking OK clears the `dataProvider` property of the DataSet component (and then the `dataProvider` property of the DataGrid component, because of the binding). Clicking Cancel dismisses the Alert component instance.

See also

[DataSet.removeSort\(\)](#)

DataSet.afterLoaded

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.afterLoaded = function (eventObj:Object):Void {
    // ...
};
dataSetInstance.addEventListener("afterLoaded", listenerObject);
```

Usage 2:

```
on (afterLoaded) {
    // ...
}
```

Description

Event; broadcast immediately after the `DataSet.items` property has been assigned.

The event object (*eventObj*) contains the following properties:

`target` The DataSet object that generated the event.

`type` The string "afterLoaded".

Example

In the following example, a form named `contactForm` (not shown) is made visible once the items in the data set `contact_ds` have been assigned.

```
contact_ds.addEventListener("afterLoaded", loadListener);
var loadListener:Object = new Object();
loadListener.afterLoaded = function (evt_obj:Object) {
    if (evt_obj.target == "contact_ds") {
        contactForm.visible = true;
    }
};
```

The following example uses the `afterLoaded` event of the `DataSet` component to populate the `dataProvider` property for a `List` component on the Stage. Drag a `List` component and a `DataSet` component to the Stage, and give them instance names of `my_list` and `my_ds`, respectively. Add the following ActionScript code to Frame 1 of the main timeline:

```
my_list.labelField = "name";

var itemsListener:Object = new Object();
itemsListener.afterLoaded = function (evt_obj:Object):Void {
    trace("After loaded");
    my_list.dataProvider = evt_obj.target.items;
}
my_ds.addEventListener("afterLoaded", itemsListener);

var item_array:Array = [{name:"Douglas"}, {name:"Vinnie"},
    {name:"Katherine"}, {name:"David"}];
my_ds.items = item_array;
```

DataSet.applyUpdates()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.applyUpdates()
```

Returns

Nothing.

Description

Method; signals that the `DataSet.deltaPacket` property has a value that you can access using data binding or directly by ActionScript. Before this method is called, the `DataSet.deltaPacket` property is `null`. This method has no effect if events have been disabled by means of the `DataSet.disableEvents()` method.

Calling this method also creates a transaction ID for the current `DataSet.deltaPacket` property and emits a `deltaPacketChanged` event. For more information, see [DataSet.deltaPacket](#).

Example

The following code calls the `applyUpdates()` method on the `my_ds DataSet`.

```
my_ds.applyUpdates();
```

The following example adds four items to the `my_ds DataSet` instance on the Stage and displays each item in the top-level of the `deltaPacket` property:

```
my_ds.addItem({name:"Thomas", age:35, gender:"M"});  
my_ds.addItem({name:"Orville", age:33, gender:"M"});  
my_ds.addItem({name:"Jonathan", age:48, gender:"M"});  
my_ds.addItem({name:"Carol", age:31, gender:"F"});
```

```
my_ds.applyUpdates();  
var i:String;  
for (i in my_ds.deltaPacket) {  
    trace(i + ":\t" + my_ds.deltaPacket[i]);  
}
```

See also

[DataSet.deltaPacket](#)

DataSet.calcFields

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();  
listenerObject.calcFields = function (eventObj:Object):Void {  
    // ...  
};  
dataSetInstance.addEventListener("calcFields", listenerObject);
```

Usage 2:

```
on (calcFields) {  
    // ...  
}
```

Description

Event; generated when values of calculated fields for the current item in the collection need to be determined. A calculated field is one whose `Kind` property is set to `Calculated` on the Schema tab of the Component inspector. The `calcFields` event listener that you create should perform the required calculation and set the value for the calculated field.

This event is also called when the value of a noncalculated field (that is, a field with its `Kind` property set to `Data` on the Schema tab) is updated.

For more information on the `Kind` property, see “Schema kinds” in *Using Flash*.

CAUTION

Do not change the values of any of noncalculated fields in this event, because this results in an “infinite loop.” Set only the values of calculated fields within the `calcFields` event.

DataSet.changesPending()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.changesPending()
```

Returns

A Boolean value.

Description

Method; returns `true` if the collection, or any item in the collection, has changes pending that have not yet been sent in a delta packet; otherwise, returns `false`.

Example

The following code enables a Save Changes button (not shown) if the DataSet collection, or any items with that collection, have had modifications made to them that haven't been committed to a delta packet.

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addEventListener("modelChanged", modelChangeListener);
function modelChangeListener(evt_obj:Object):Void {
    if (evt_obj.target.changesPending()) {
        trace("changes pending");
        submitChanges_button.enabled = true;
    }
}
submitChanges_button.enabled = false;
my_ds.addItem({name:"Hal", years:4});
```

DataSet.clear()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.clear()
```

Returns

Nothing.

Description

Method; removes the items in the current view of the collection. Which items are considered “viewable” depends on any current filter and range settings on the current iterator. Therefore, calling this method might not clear all of the items in the collection. To clear all of the items in the collection regardless of the current iterator's view, use [DataSet.removeAll\(\)](#).

If [DataSet.logChanges](#) is set to true when you invoke this method, “remove” entries are added to [DataSet.deltaPacket](#) for all items in the collection.

Example

The following example removes all items from the current view of the DataSet collection. Because the `logChanges` property is set to `true`, the removal of those items is logged.

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addSort("nameSort", ["name"]);
my_ds.filtered = true;
my_ds.filterFunc = function(item:Object):Boolean {
    return (item.years >= 3);
};
my_ds.logChanges = true;
my_ds.clear(); // Remove filtered items from dataset.
my_ds.removeSort("nameSort");
```

See also

[DataSet.deltaPacket](#), [DataSet.logChanges](#)

DataSet.createItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.createItem([ itemData ])
```

Parameters

itemData Data associated with the item. This parameter is optional.

Returns

The newly constructed item.

Description

Method; creates an item that isn't associated with the collection. You can specify the class of object created by using the `DataSet.itemClassName` property. If no `DataSet.itemClassName` value is specified and the `itemData` parameter is omitted, an anonymous object is constructed. This anonymous object's properties are set to the default values based on the schema currently specified by `DataSet.schema`.

When this method is invoked, any listeners for the `DataSet.newItem` event are notified and are able to manipulate the item before it is returned by this method. The optional item data is used to initialize the class specified with the `DataSet.itemClassName` property or is used as the item if `DataSet.itemClassName` is blank.

A `DataSetError` exception is thrown when the class specified with the `DataSet.itemClassName` property cannot be loaded.

Example

```
my_ds.addEventListener("newItem", newItemListener);
function newItemListener(evt_obj:Object):Void {
    trace("new item was added: {name:'" + evt_obj.item.name + "', years:" +
        evt_obj.item.years + "}");
}

my_ds.addItem(my_ds.createItem({name:"Wilson", years:3}));
my_ds.addItem({name:"Tom", years:2});

my_ds.filtered = true;
my_ds.filterFunc = function(item:Object):Boolean {
    return (item.years % 2 == 0);
};
```

See also

[DataSet.itemClassName](#), [DataSet.newItem](#), [DataSet.schema](#)

DataSet.currentItem

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`dataSetInstance.currentItem`

Description

Property (read-only); returns the current item in the `DataSet` collection, or `null` if the collection is empty or if the current iterator's view of the collection is empty.

This property provides direct access to the item in the collection. Changes made by directly accessing this object are not tracked (in the `DataSet.deltaPacket` property), nor are any of the schema settings applied to any properties of this object.

Example

The following example displays the value of the `name` property defined in the current item in the data set named `customers_ds`.

```
customers_ds.addItem({name:"Milton", years:3});
customers_ds.addItem({name:"Mark", years:3});
customers_ds.addItem({name:"Sarah", years:1});
customers_ds.addItem({name:"Michael", years:2});
customers_ds.addItem({name:"Frank", years:2});

trace(customers_ds.currentItem.name); // Frank
```

DataSet.dataProvider

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.dataProvider
```

Description

Property; the data provider for this data set. This property provides data to user interface controls, such as the `List` and `DataGrid` components.

For more information about the `DataProvider` API, see [“DataProvider API” on page 317](#).

Example

The following code assigns the `dataProvider` property of a `DataSet` object to the corresponding property of a `DataGrid` component.

```
my_dg.dataProvider = my_ds.dataProvider;
```

DataSet.deltaPacket

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.deltaPacket

Description

Property; returns a delta packet that contains all of the change operations made to the *dataSet* collection and its items. This property is null until [DataSet.applyUpdates\(\)](#) is called on *dataSet*.

When [DataSet.applyUpdates\(\)](#) is called, a transaction ID is assigned to the delta packet. This transaction ID is used to identify the delta packet on an update round trip from the server and back to the client. Any subsequent assignment to the `deltaPacket` property by a delta packet with a matching transaction ID is assumed to be the server's response to the changes previously sent. A delta packet with a matching ID is used to update the collection and report errors specified within the packet.

Errors or server messages are reported to listeners of the [DataSet.resolveDelta](#) event. Note that the [DataSet.logChanges](#) settings are ignored when a delta packet with a matching ID is assigned to [DataSet.deltaPacket](#). A delta packet without a matching transaction ID updates the collection, as if the DataSet API were used directly. This may create additional delta entries, depending on the current [DataSet.logChanges](#) setting of *dataSet* and the delta packet.

A `DataSetError` exception is thrown if a delta packet is assigned with a matching transaction ID and one of the items in the newly assigned delta packet cannot be found in the original delta packet.

See also

[DataSet.applyUpdates\(\)](#), [DataSet.logChanges](#), [DataSet.resolveDelta](#)

DataSet.deltaPacketChanged

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.deltaPacketChanged = function (eventObj:Object):Void {
    // ...
};
dataSetInstance.addEventListener("deltaPacketChanged", listenerObject);
```

Usage 2:

```
on (deltaPacketChanged) {
    // ...
}
```

Description

Event; broadcast when the specified DataSet object's `deltaPacket` property has been changed and is ready to be used.

See also

[DataSet.deltaPacket](#)

DataSet.disableEvents()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.disableEvents()
```

Returns

Nothing.

Description

Method; disables events for the DataSet object. While events are disabled, no user interface controls (such as a DataGrid component) are updated when changes are made to items in the collection, or when the DataSet object is scrolled to another item in the collection.

To reenable events, you must call [DataSet.enableEvents\(\)](#). The `disableEvents()` method can be called multiple times, and `enableEvents()` must be called an equal number of times to reenable the dispatching of events.

Example

In the following example, events are disabled before changes are made to items in the collection, so that the DataSet object won't affect performance by trying to refresh controls:

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
    trace("model changed, DataSet now has " + evt_obj.target.items.length + "
        items");
}
// Disable events for the data set.
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
    my_ds.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}
```

```
trace("After:");
traceItems();

// Tell the dataset it's time to update the controls now.
my_ds.enableEvents();

function traceItems():Void {
    for (var i:Number = 0; i < my_ds.items.length; i++) {
        trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
    }
    trace("");
}
```

See also

[DataSet.enableEvents\(\)](#)

DataSet.enableEvents()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.enableEvents()
```

Returns

Nothing.

Description

Method; reenables events for the DataSet objects after events have been disabled by a call to [DataSet.disableEvents\(\)](#). To reenable events for the DataSet object, the `enableEvents()` method must be called an equal or greater number of times than `disableEvents()` was called.

Example

In the following example, events are disabled before changes are made to items in the collection, so that the DataSet object won't affect performance by trying to refresh controls.

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
    trace("model changed, DataSet now has " + evt_obj.target.items.length + "
        items");
}
// Disable events for the data set.
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
    my_ds.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}

trace("After:");
traceItems();

// Tell the data set it's time to update the controls now.
my_ds.enableEvents();

function traceItems():Void {
    for (var i:Number = 0; i < my_ds.items.length; i++) {
        trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
    }
    trace("");
}
}
```

See also

[DataSet.disableEvents\(\)](#)

DataSet.filtered

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.filtered

Description

Property; a Boolean value that indicates whether the data in the current iterator is filtered. The default value is `false`. When this property is `true`, the filter function specified by `DataSet.filterFunc` is called for each item in the collection.

Example

In the following example, filtering is enabled on the `DataSet` object named `employee_ds`. Suppose that each record in the `DataSet` collection contains a field named `empType`. The following filter function returns `true` if the `empType` field in the current item is set to "management"; otherwise, it returns `false`.

```
employee_ds.filtered = true;
employee_ds.filterFunc = function (item:Object) {
    // Filter out employees who are managers.
    return(item.empType != "management");
};
```

The following example populates a `DataGrid` component from content dynamically loaded using the `XMLConnector` component. When a user clicks a `CheckBox` instance on the Stage, the contents of the `DataSet` component are filtered and are updated automatically in the `DataGrid` component.

Drag the following components to the Stage, and give them the following instance names:

- `CheckBox` (`editorsChoice_ch`)
- `DataGrid` (`reviews_dg`)
- `DataSet` (`reviews_ds`)
- `XMLConnector` (`reviews_xmlconn`)

Download a copy of the following XML document and save it to your local hard disk: <http://www.helpexamples.com/flash/xml/reviews.xml>. This XML document will be loaded dynamically using the XMLConnector component, but you'll use the local copy to import the XML schema into the DataSet component. Select the XMLConnector instance on the Stage and select the Schema tab from the Component inspector. Select the `results` property and click the "Import a schema from a sample XML file" button. Select the XML document that you downloaded to your local hard disk and click Open. The schema of the XML document should be imported into the DataSet component. With the `reviews_xmlconn` XMLConnector instance still selected on the Stage, add a binding from the `reviews_xmlconn.results.reviews.review` array to the `dataProvider` property of the `reviews_ds` DataSet instance. Set the direction of the data binding to "out" in the Bindings tab. Select the `reviews_ds` DataSet instance on the Stage and add another binding for the `dataProvider` property. With the `reviews_ds` instance selected, select the Bindings tab of the Component inspector. Click the Add binding button and add a binding from the `dataProvider` property of the DataSet component to the `dataProvider` property of the `reviews_dg` DataGrid instance.

Add the following code to Frame 1 of the main timeline:

```
editorsChoice_ch.label = "Editor's Choice";

reviews_xmlconn.direction = "receive";
reviews_xmlconn.multipleSimultaneousAllowed = false;
reviews_xmlconn.URL = "http://www.helpexamples.com/flash/xml/reviews.xml";
reviews_xmlconn.trigger();

reviews_dg.setSize(320, 240);
reviews_dg.addColumn("name");
reviews_dg.addColumn("rating");
reviews_dg.addColumn("reviewDate");
reviews_dg.getColumnAt(0).width = 100;
reviews_dg.getColumnAt(1).width = 100;
reviews_dg.getColumnAt(2).width = 100;
reviews_dg.setStyle("alternatingRowColors", [0xFFFFFFFF, 0xF6F6F6]);

editorsChoice_ch.addEventListener("click", editorsChoiceListener);
function editorsChoiceListener(evt_obj:Object):Void {
    reviews_ds.filtered = evt_obj.target.selected;
    reviews_ds.filterFunc = function(item:Object):Boolean {
        return (item.editorsChoice == 1);
    };
}
```

Select Control > Test Movie. By default, the DataGrid component should display each of the reviews from the external XML document. Clicking on the Editor's Choice CheckBox component causes the DataSet component to be filtered, so that only the specific reviews flagged as an editor's choice appear.

See also

[DataSet.filterFunc](#)

DataSet.filterFunc

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.filterFunc = function (item:Object):Boolean {  
    // return true|false;  
};
```

Description

Property; specifies a function that determines which items are included in the current view of the collection. When [DataSet.filtered](#) is set to `true`, the function assigned to this property is called for each record (transfer object) in the collection. For each item that is passed to the function, it should return `true` if the item should be included in the current view, or `false` if the item should not be included in the current view.

When changing the filter function on a data set, you must set the `filtered` property to `false` and then `true` again in order for the proper `modelChanged` event to be generated. Changing the `filterFunc` property won't generate the event.

Also, if a filter is already in place when the data loads in (`modelChanged` or `updateAll`), the filter isn't applied until `filtered` is set to `false` and then back to `true` again.

Example

In the following example, filtering is enabled on the DataSet object named `employee_ds`. The specified filter function returns `true` if the `empType` field in each item is set to "management"; otherwise, it returns `false`.

```
employee_ds.filtered = true;
employee_ds.filterFunc = function (item:Object):Boolean {
    // Filter out employees who are managers.
    return(item.empType != "management");
};
```

In the following example, you filter the contents of a DataSet component based on the selected item in a ComboBox component. The ComboBox component lets you select all people, males only, or females only.

Drag the a DataSet, DataGrid, and ComboBox components to the Stage, and give them instance names of `my_ds`, `data_dg`, and `filter_cb` respectively. Select the `my_ds` DataSet instance on the Stage and add two new properties: `name` and `gender`. Create a data binding between the `dataProvider` property of the DataSet and the `dataProvider` of the DataGrid component by using the Bindings panel of the Component inspector. Add the following ActionScript code to Frame 1 of the main timeline:

```
my_ds.dataProvider = new Array({name:"Charles", gender:"M"}, {name:"Buddy",
    gender:"M"}, {name:"Walter", gender:"M"}, {name:"Ellen", gender:"F"},
    {name:"Jamie", gender:"F"}, {name:"Sarah", gender:"F"}, {name:"Adam",
    gender:"M"});
my_ds.addSort("nameSort", ["name"]);
my_ds.addSort("genderSort", ["gender", "name"]);
my_ds.useSort("genderSort");

filter_cb.dataProvider = [{label:"All", value:""} , {label:"Male only",
    value:"M"}, {label:"Female only", value:"F"}];
filter_cb.addEventListener("change", filterListener);
function filterListener(evt_obj:Object):Void {
    var selItem:Object = evt_obj.target.selectedItem;
    if (selItem.value.length == 0) {
        my_ds.filtered = false;
    } else {
        my_ds.filtered = true;
        my_ds.filterFunc = function(item:Object):Boolean {
            return (item.gender == selItem.value);
        }
    }
}
```

The following example populates a DataGrid component from content dynamically loaded using the XMLConnector component. When a user clicks a CheckBox instance on the Stage, the contents of the DataSet component are filtered and are updated automatically in the DataGrid component.

Drag the following components to the Stage, and give them the following instance names:

- CheckBox (editorsChoice_ch)
- DataGrid (reviews_dg)
- DataSet (reviews_ds)
- XMLConnector (reviews_xmlconn)

Download a copy of the following XML document, and save it to your local hard disk: www.helpexamples.com/flash/xml/reviews.xml. This XML document loads dynamically using the XMLConnector component. You'll use the local copy to import the XML schema into the DataSet component. Select the XMLConnector instance on the Stage, and select the Schema tab from the Component inspector. Select the results property, and click Import a Schema From a Sample XML File. Select the XML document that you downloaded to your local hard disk, and click Open. The schema of the XML document should import into the DataSet component. With the reviews_xmlconn XMLConnector instance still selected on the Stage, add a binding from the reviews_xmlconn.results.reviews.review array to the dataProvider property of the reviews_ds DataSet instance. Set the direction of the data binding to Out in the Bindings tab. Select the reviews_ds DataSet instance on the Stage, and add another binding for the dataProvider property. With the reviews_ds instance selected, select the Bindings tab of the Component inspector. Click the Add Binding button and add a binding from the dataProvider property of the DataSet component to the dataProvider property of the reviews_dg DataGrid instance.

Add the following code to Frame 1 of the main timeline:

```
editorsChoice_ch.label = "Editor's Choice";

reviews_xmlconn.direction = "receive";
reviews_xmlconn.multipleSimultaneousAllowed = false;
reviews_xmlconn.URL = "http://www.helpexamples.com/flash/xml/reviews.xml";
reviews_xmlconn.trigger();

reviews_dg.setSize(320, 240);
reviews_dg.addColumn("name");
reviews_dg.addColumn("rating");
reviews_dg.addColumn("reviewDate");
reviews_dg.getColumnAt(0).width = 100;
reviews_dg.getColumnAt(1).width = 100;
reviews_dg.getColumnAt(2).width = 100;
reviews_dg.setStyle("alternatingRowColors", [0xFFFFFFFF, 0xF6F6F6]);

editorsChoice_ch.addEventListener("click", editorsChoiceListener);
function editorsChoiceListener(evt_obj:Object):Void {
    reviews_ds.filtered = evt_obj.target.selected;
    reviews_ds.filterFunc = function(item:Object):Boolean {
        return (item.editorsChoice == 1);
    };
}
```

Select Control > Test Movie. By default, the DataGrid component should display each of the reviews from the external XML document. Clicking on the Editor's Choice CheckBox component causes the DataSet component to be filtered, so that only the specific reviews flagged as an editor's choice appear.

See also

[DataSet.filtered](#)

DataSet.find()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.find(searchValues)
```

Parameters

searchValues An array that contains one or more field values to be found within the current sort.

Returns

Returns `true` if the values are found; otherwise, returns `false`.

Description

Method; searches the current view of the collection for an item with the field values specified by *searchValues*. Which items are in the current view depends on any current filter and range settings. If an item is found, it becomes the current item in the `DataSet` object.

The values specified by *searchValues* must be in the same order as the field list specified by the current sort (see the example below).

If the current sort is not unique, the record (transfer object) found is nondeterministic. If you want to find the first or last occurrence of a transfer object in a nonunique sort, use `DataSet.findFirst()` or `DataSet.findLast()`.

Conversion of the data specified is based on the underlying field's type. For example, if you specify `["05-02-02"]` as a search value, the underlying date field is used to convert the value using the date's `DataType.setAsString()` method. If you specify `[new Date().getTime()]`, the date's `DataType.setAsNumber()` method is used.

Example

The following example searches for an item in the current collection whose `name` and `id` fields contain the values "Bobby" and 105, respectively. If found, `DataSet.getItemId()` is used to get the unique identifier for the item in the collection, and `DataSet.locateById()` is used to position the current iterator on that item.

```
var studentID:String = null;
student_ds.addSort("id", ["name","id"]);
// Locate the transfer object identified by "Bobby" and 105.
// Note that the order of the search fields matches those
// specified in addSort().
if (student_ds.find(["Bobby", 105])) {
    studentID = student_ds.getItemId();
}
// Now use locateById() to position the current iterator
// on the item in the collection whose ID matches studentID.
if (studentID != null) {
    student_ds.locateById(studentID);
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.getItemId\(\)](#), [DataSet.locateById\(\)](#)

DataSet.findFirst()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.findFirst(searchValues)
```

Parameters

searchValues An array that contains one or more field values to be found within the current sort.

Returns

Returns `true` if the items are found; otherwise, returns `false`.

Description

Method; searches the current view of the collection for the first item with the field values specified by *searchValues*. Which items are in the current view depends on any current filter and range settings.

The values specified by *searchValues* must be in the same order as the field list specified by the current sort (see the example below).

Conversion of the data specified is based on the underlying field's type. For example, if the search value specified is ["05-02-02"], the underlying date field is used to convert the value with the date's `setAsString()` method. If the value specified is [`new Date().getTime()`], the date's `setAsNumber()` method is used.

Example

The following example uses `DataSet.find()` to search for an item in the current collection whose name and `id` fields contain the values "Bobby" and 105, respectively. If found, `DataSet.getItemId()` is used to get the unique identifier for that item, and `DataSet.locateById()` is used to position the current iterator at that item.

To test this example, drag a `DataSet` component to the Stage, and give it an instance name of `student_ds`. Add two properties, `name` (data type: `String`) and `id` (data type: `Number`) to the `DataSet` by using the Schema tab of the Component inspector. If you don't already have a copy of the `DataBindingClasses` compiled clip in your library, drag a copy of the compiled clip from the Classes library (Window > Common Libraries > Classes). Add the following `ActionScript` to Frame 1 of the main timeline:

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
    studentID = student_ds.getItemId();
    student_ds.locateById(studentID);
    trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
    trace("We lost Billy!");
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.getItemId\(\)](#), [DataSet.locateById\(\)](#)

DataSet.findLast()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.findLast(searchValues)
```

Parameters

searchValues An array that contains one or more field values to be found within the current sort.

Returns

Returns `true` if the items are found; otherwise, returns `false`.

Description

Method; searches the current view of the collection for the last item with the field values specified by *searchValues*. Which items are in the current view depends on any current filter and range settings.

The values specified by *searchValues* must be in the same order as the field list specified by the current sort (see the example below).

Conversion of the data specified is based on the underlying field's type. For example, if the search value specified is ["05-02-02"], the underlying date field is used to convert the value with the date's `setAsString()` method. If the value specified is [`new Date().getTime()`], the date's `setAsNumber()` method is used.

Example

The following example searches for the last item in the current collection whose name and age fields contain "Bobby" and "13". If found, `DataSet.getItemId()` is used to get the unique identifier for the item in the collection, and `DataSet.locateById()` is used to position the current iterator on that item.

```
var studentID:String = null;
student_ds.addSort("nameAndAge", ["name", "age"]);
// Locate the last transfer object with the specified values.
// Note that the order of the search fields matches those
// specified in addSort().
if (student_ds.findLast(["Bobby", "13"])) {
    studentID = student_ds.getItemId();
}

// Now use locateById() to position the current iterator
// on the item in the collection whose ID matches studentID.
if (studentID != null) {
    student_ds.locateById(studentID);
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.getItemId\(\)](#), [DataSet.locateById\(\)](#)

DataSet.first()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.first()
```

Returns

Nothing.

Description

Method; makes the first item in the current view of the collection the current item. Which items are in the current view depends on any current filter and range settings.

Example

The following code positions the data set `inventory_ds` at the first item in its collection, and then displays the value of the `price` property contained by that item using the `DataSet.currentItem` property.

```
inventory_ds.first();  
trace("The price of the first item is:" + inventory_ds.currentItem.price);
```

The following example iterates over all of the items in the current view of the collection (starting at its beginning) and performs a calculation on the `price` property of each item.

```
my_ds.addItem({name:"item a", price:16});  
my_ds.addItem({name:"item b", price:9});  
  
my_ds.first();  
while (my_ds.hasNext()) {  
    my_ds.currentItem.price *= 0.5; // Everything's 50% off!  
    my_ds.next();  
}  
  
for (var i in my_ds.items) {  
    trace(my_ds.items[i].name + ": " + my_ds.items[i].price);  
}
```

See also

[DataSet.last\(\)](#)

DataSet.getItemId()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.getItemId([index])
```

Parameters

index A number specifying the item in the current view for which to get the ID. This parameter is optional.

Returns

A string.

Description

Method; returns the identifier of the current item in the collection, or that of the item specified by *index*. This identifier is unique only in this collection and is assigned automatically by [DataSet.addItem\(\)](#).

Example

The following code gets the unique ID for the current item in the collection and then displays it in the Output panel.

```
var itemNo:String = my_ds.getItemId();  
trace("Employee id(" + itemNo + ")");
```

The following example uses [DataSet.find\(\)](#) to search for an item in the current collection whose name and id fields contain the values "Bobby" and 105, respectively. If found, [DataSet.getItemId\(\)](#) is used to get the unique identifier for that item, and [DataSet.locateById\(\)](#) is used to position the current iterator at that item.

To test this example, drag a `DataSet` component to the Stage, and give it an instance name of `student_ds`. Add two properties, `name` (data type: `String`) and `id` (data type: `Number`) to the `DataSet` by using the Schema tab of the Component inspector. If you don't already have a copy of the `DataBindingClasses` compiled clip in your library, drag a copy of the compiled clip from the `Classes` library (`Window > Common Libraries > Classes`). Add the following `ActionScript` to Frame 1 of the main timeline:

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
    studentID = student_ds.getItemId();
    student_ds.locateById(studentID);
    trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
    trace("We lost Billy!");
}
```

See also

[DataSet.addItem\(\)](#)

DataSet.getIterator()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.getIterator()
```

Returns

A `ValueListIterator` object.

Description

Method; returns a new iterator for this collection; this iterator is a clone of the current iterator in use, including its current position in the collection. This method is mainly for advanced users who want access to multiple, simultaneous views of the same collection.

Example

The following example uses `DataSet.find()` to search for an item in the current collection whose name field contain the value "Bobby". Even though the `myIterator` iterator is pointing to Bobby's record, the main iterator of `student_ds` still points to the last record, Billy.

To test this example, drag a `DataSet` component to the Stage, and give it an instance name of `student_ds`. Add two properties, `name` (data type: `String`) and `id` (data type: `Number`) to the `DataSet` component by using the Schema tab of the Component inspector. If you don't already have a copy of the `DataBindingClasses` compiled clip in your library, drag a copy of the compiled clip from the Classes library (Window > Common Libraries > Classes). Add the following `ActionScript` to Frame 1 of the main timeline:

```
import mx.data.to.ValueListIterator;

student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

var myIterator:ValueListIterator = student_ds.getIterator();
myIterator.sortOn(["name"]);
myIterator.find({name:"Bobby"}).id = "999";

trace(student_ds.currentItem.name + " [" + student_ds.currentItem.id +
    "]);
    // Billy [107]

student_ds.addSort("id", ["name", "id"]);
if (student_ds.find({name:"Bobby", id:999})) {
    student_ds.locateById(student_ds.getItemId());
    trace(student_ds.currentItem.name + " [" + student_ds.currentItem.id +
        "]);
        // Bobby [999]
} else {
    trace("We lost Billy!");
}
```

DataSet.getLength()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.getLength()
```

Returns

The number of items in the data set.

Description

Method; returns the number of items in the data set.

Example

The following example calls `getLength()`:

```
//...  
var my_ds:mx.data.components.DataSet;  
my_ds = _parent.thisShelf.compactDiscs_ds;  
trace ("Data set size is: " + my_ds.getLength());  
//...
```

DataSet.hasNext()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.hasNext()
```

Returns

A Boolean value.

Description

Method; returns `false` if the current iterator is at the end of its view of the collection; otherwise, returns `true`.

Example

The following example iterates over all of the items in the current view of the collection (starting at its beginning) and performs a calculation on the `price` property of each item.

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.first();
while (my_ds.hasNext()) {
    my_ds.currentItem.price *= 0.5; // Everything's 50% off!
    my_ds.next();
}

for (var i in my_ds.items) {
    trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

See also

[DataSet.currentItem](#), [DataSet.first\(\)](#), [DataSet.next\(\)](#)

DataSet.hasPrevious()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.hasPrevious()
```

Returns

A Boolean value.

Description

Method; returns `false` if the current iterator is at the beginning of its view of the collection; otherwise, returns `true`.

Example

The following example iterates over all the items in the current view of the collection (starting from the its last item) and performs a calculation on the `price` property of each item:

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.last();
while (my_ds.hasPrevious()) {
    my_ds.currentItem.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}

for (var i in my_ds.items) {
    trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

See also

[DataSet.currentItem](#), [DataSet.skip\(\)](#), [DataSet.previous\(\)](#)

DataSet.hasSort()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.hasSort(sortName)
```

Parameters

sortName A string that contains the name of a sort created with [DataSet.addSort\(\)](#).

Returns

A Boolean value.

Description

Method; returns `true` if the sort specified by *sortName* exists; otherwise, returns `false`.

Example

The following code tests whether a sort named “nameSort” exists. If the sort already exists, it is made the current sort by means of `DataSet.useSort()`. If a sort by that name doesn’t exist, one is created by means of `DataSet.addSort()`. To test this example, drag a `DataSet` component and a `List` component to the Stage, and give them instance names of `my_ds` and `my_list` respectively. Add a binding between the `dataProvider` property of the `DataSet` component and the `dataProvider` property of the `List` component by using the Bindings tab of the Component inspector. Create two properties in the schema of `my_ds` `DataSet` by using the Schema tab of the Component inspector: `name` (data type: `String`) and `years` (data type: `Number`). Add the following code on Frame 1 of the main timeline:

```
import mx.data.components.datasetclasses.DataSetIterator;
my_list.labelField = "name";

my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

if (my_ds.hasSort("nameSort")) {
    my_ds.useSort("nameSort");
} else {
    my_ds.addSort("nameSort", ["name"], DataSetIterator.Descending);
}
```

See also

[DataSet.addSort\(\)](#), [DataSet.applyUpdates\(\)](#), [DataSet.useSort\(\)](#)

DataSet.isEmpty()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.isEmpty()
```

Returns

A Boolean value.

Description

Method; returns `true` if the specified `DataSet` object doesn't contain any items (that is, if `dataSet.length == 0`).

Example

The following code disables a Delete Record button (not shown) if the `DataSet` object it applies to is empty:

```
if (my_ds.isEmpty()) {
    delete_button.enabled = false;
}
```

See also

[DataSet.length](#)

DataSet.items

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`dataSetInstance.items`

Description

Property; an array of items managed by `my_ds`.

Example

The following example assigns an array of objects to the `items` property of a `DataSet` object:

```
var recData:Array = [{id:0, firstName:"Mick", lastName:"Jones"},
                    {id:1, firstName:"Joe", lastName:"Strummer"},
                    {id:2, firstName:"Paul", lastName:"Simonon"}];
my_ds.items = recData;
```

DataSet.itemClassName

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.itemClassName
```

Description

Property; a string indicating the name of the class that should be created when items are added to the collection. The class you specify must implement the `TransferObject` interface, shown below.

```
interface mx.data.to.TransferObject {  
    function clone():Object;  
    function getPropertyData():Object;  
    function setPropertyData(propData:Object):Void;  
}
```

You can also set this property in the Property inspector.

To make the specified class available at runtime, you must also make a fully qualified reference to this class somewhere in your SWF file's code, as in the following code snippet:

```
var myItem:my.package.myItem;
```

A `DataSetError` exception is thrown if you try to modify the value of this property after the `DataSet.items` array has been loaded.

For more information, see [“TransferObject interface” on page 1233](#).

DataSet.iteratorScrolled

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.iteratorScrolled = function (eventObj:Object) {
    // ...
};
dataSetInstance.addEventListener("iteratorScrolled", listenerObject);
```

Usage 2:

```
on (iteratorScrolled) {
    // ...
}
```

Description

Event; generated immediately after the current iterator has scrolled to a new item in the collection.

The event object (*eventObj*) contains the following properties:

target The DataSet object that generated the event.

type The string "iteratorScrolled".

scrolled A number that specifies how many items the iterator scrolled; positive values indicate that the iterator moved forward in the collection; negative values indicate that it moved backward in the collection.

Example

In the following example, the status bar of an application (not shown) is updated when the position of the current iterator changes:

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addEventListener("iteratorScrolled", iteratorScrolledListener);

my_ds.first(); // Trigger the iteratorScrolled event.

function iteratorScrolledListener(evt_obj:Object):Void {
    trace("The iterator was scrolled.");
}
```

DataSet.last()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.last()
```

Returns

Nothing.

Description

Method; makes the last item in the current view of the collection the current item.

Example

The following code, attached to a Button component, goes to the last item in the DataSet collection:

```
function goLast(evt_obj:obj):Void {  
    inventory_ds.last();  
}  
goLast_button.addEventListener("click", goLast);
```

The following example iterates over all the items in the current view of the collection (starting from the its last item) and performs a calculation on the price property of each item:

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.last();
while (my_ds.hasPrevious()) {
    my_ds.currentItem.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}

for (var i in my_ds.items) {
    trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

See also

[DataSet.first\(\)](#)

DataSet.length

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.length

Description

Property (read-only); specifies the number of items in the current view of the collection. The viewable number of items is based on the current filter and range settings.

Example

In the following example, events are disabled before changes are made to items in the collection, so that the DataSet object won't affect performance by trying to refresh controls:

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
    trace("model changed, DataSet now has " + evt_obj.target.length + "
        items");
}
// Disable events for the data set.
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
    my_ds.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}

trace("After:");
traceItems();

// Tell the dataset it's time to update the controls now.
my_ds.enableEvents();

function traceItems(label:String):Void {
    for (var i:Number = 0; i < my_ds.length; i++) {
        trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
    }
    trace("");
}
```

DataSet.loadFromSharedObj()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`dataSetInstance.loadFromSharedObj(objName, [localPath])`

Parameters

objName A string specifying the name of the shared object to retrieve. The name can include forward slashes (for example, “work/addresses”). Spaces and the following characters are not allowed in the specified name:

~ % & \ ; : " ' , < > ? #

localPath An optional string parameter that specifies the full or partial path to the SWF file that created the shared object. This string is used to determine where the object is stored on the user’s computer. The default value is the SWF file’s full path.

Returns

Nothing.

Description

Method; loads all of the relevant data needed to restore this DataSet collection from a shared object. To save a DataSet collection to a shared object, use [DataSet.saveToSharedObj\(\)](#). The `DataSet.loadFromSharedObject()` method overwrites any data or pending changes that might exist in this DataSet collection. Note that the instance name of the DataSet collection is used to identify the data in the specified shared object.

This method throws a `DataSetError` exception if the specified shared object isn’t found or if there is a problem retrieving the data from it.

Example

The following example attempts to load a shared object named `webapp/customerInfo` associated with the data set named `my_ds`. The method is called within a `try...catch` code block.

```
import mx.data.components.datasetclasses.DataSetError;
try {
    my_ds.loadFromSharedObj("webapp/customerInfo");
} catch(e:DataSetError) {
    trace("Unable to load shared object.");
}
```

See also

[DataSet.saveToSharedObj\(\)](#)

DataSet.locateById()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.locateById(id)
```

Parameters

id A string identifier for the item in the collection to be located.

Returns

A Boolean value.

Description

Method; positions the current iterator on the collection item whose ID matches *id*. This method returns `true` if the specified ID can be matched to an item in the collection; otherwise, it returns `false`.

Example

The following example uses `DataSet.find()` to search for an item in the current collection whose `name` and `id` fields contain the values "Bobby" and 105, respectively. If found, `DataSet.getItemId()` is used to get the unique identifier for that item, and `DataSet.locateById()` is used to position the current iterator at that item.

To test this example, drag a `DataSet` component to the Stage, and give it an instance name of `student_ds`. Add two properties, `name` (String) and `id` (Number) to the `DataSet` by using the **Schema** tab of the Component inspector. If you don't already have a copy of the `DataBindingClasses` compiled clip in your library, drag a copy of the compiled clip from the **Classes** library (Window > Common Libraries > Classes). Add the following ActionScript to Frame 1 of the main timeline:

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
    studentID = student_ds.getItemId();
    student_ds.locateById(studentID);
    trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
    trace("We lost Billy!");
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.find\(\)](#), [DataSet.getItemId\(\)](#)

DataSet.logChanges

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.logChanges

Description

Property; a Boolean value that specifies whether changes made to the data set, or to its items, should (`true`) or should not (`false`) be recorded in `DataSet.deltaPacket`.

When this property is set to `true`, operations performed at the collection level and item level are logged. Collection-level changes include the addition and removal of items from the collection. Item-level changes include property changes made to items and method calls made on items by means of the `DataSet` component.

Example

The following example disables logging for the `DataSet` object named `userData`.

```
user_ds.logChanges = false;
```

See also

[DataSet.deltaPacket](#)

DataSet.modelChanged

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Description

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.modelChanged = function (eventObj:Object):Void {
    // ...
};
dataSetInstance.addEventListener("modelChanged", listenerObject);
```

Usage 2:

```
on (modelChanged) {
    // ...
}
```

Description

Event; broadcast when the collection changes in some way—for example, when items are removed or added to the collection, when the value of an item's property changes, or when the collection is filtered or sorted.

The event object (*eventObj*) contains the following properties:

`target` The `DataSet` object that generated the event.

`type` The string "iteratorScrolled".

`firstItem` The index (number) of the first item in the collection that was affected by the change.

`lastItem` The index (number) of the last item in the collection that was affected by the change (equals `firstItem` if only one item was affected).

`fieldName` A string that contains the name of the field being affected. This property is undefined unless the change was made to a property of the `DataSet` object.

`eventName` A string that describes the change that took place. This can be one of the following values:

String value	Description
"addItem"	A series of items has been added.
"filterModel"	The model has been filtered, and the view needs refreshing (reset scroll position).
"removeItems"	A series of items has been deleted.
"schemaLoaded"	The fields definition of the data provider has been declared.
"sort"	The data has been sorted.
"updateAll"	The entire view needs refreshing, excluding scroll position.
"updateColumn"	An entire field's definition in the data provider needs refreshing.
"updateField"	A field in an item has been changed and needs refreshing.
"updateItems"	A series of items needs refreshing.

Example

In the following example, the `modelChanged` event gets dispatched whenever an item is added or removed from the data set:

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
    trace("[event = " + evt_obj.eventName + "] the data set now has " +
        evt_obj.target.items.length + " items.");
}
my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});
my_ds.removeItemAt(0);
```

In the following example, a Delete Item button is disabled if the items have been removed from the collection and the target DataSet object has no more items:

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
    trace("model changed, DataSet now has " + evt_obj.target.items.length + "
        items");
}
// Disable events for the data set.
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while (my_ds.hasPrevious()) {
    my_ds.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}

trace("After:");
traceItems();

// Tell the dataset it's time to update the controls now.
my_ds.enableEvents();

function traceItems(label:String):Void {
    for (var i:Number = 0; i < my_ds.items.length; i++) {
        trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
    }
    trace("");
}
```

See also

[DataSet.isEmpty\(\)](#)

DataSet.newItem

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.newItem = function (eventObj:Object) {
    // ...
};
dataSetInstance.addEventListener("newItem", listenerObject);
```

Usage 2:

```
on (newItem) {
    // ...
}
```

Description

Event; broadcast when a new transfer object is constructed by means of [DataSet.createItem\(\)](#). A listener for this event can make modifications to the item before it is added to the collection.

The event object (*eventObj*) contains the following properties:

target The DataSet object that generated the event.

type The string "iteratorScrolled".

item A referenece to the item that was created.

Example

The following example makes modifications to a newly created item before it's added to the collection:

```
function newItemEvent(evt_obj:Object):Void {
    var employee:Object = evt_obj.item;
    employee.name = "newGuy";
    // Property data happens to be XML.
    employee.zip =
        employee.getPropertyData().firstChild.childNodes[1].attributes.zip;
}
employees_ds.addEventListener("newItem", newItemEvent);
```

DataSet.next()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.next()

Returns

Nothing.

Description

Method; makes the next item in the current view of the collection the current item. Which items are in the current view depends on any current filter and range settings.

Example

The following example iterates over all the items in the current view of the collection (starting at its beginning) and performs a calculation on the price property of each item:

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.first();
while (my_ds.hasNext()) {
    my_ds.currentItem.price *= 0.5; // Everything's 50% off!
    my_ds.next();
}

for (var i in my_ds.items) {
    trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

See also

[DataSet.first\(\)](#), [DataSet.hasNext\(\)](#)

DataSet.previous()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.previous()
```

Returns

Nothing.

Description

Method; makes the previous item in the current view of the collection the current item. Which items are in the current view depends on any current filter and range settings.

Example

The following example loops over each item in a data set and traces each item's price:

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});
my_ds.last();
while (my_ds.hasPrevious()) {
    trace(my_ds.currentItem.price);
    my_ds.previous();
}
```

The following example loops over all the items in the current view of the collection, starting from the last item, and performs a calculation on a field in each item:

```
my_ds.last();
while (my_ds.hasPrevious()) {
    my_ds.price *= 0.5; // Everything's 50% off!
    my_ds.previous();
}
```

See also

[DataSet.first\(\)](#), [DataSet.hasNext\(\)](#)

DataSet.properties

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.properties

Description

Property (read-only); returns an object that contains all of the exposed properties (fields) for any transfer object within this collection.

Example

The following example displays all the names of the properties in the DataSet object named `my_ds`:

```
var i:String;
for (i in my_ds.properties) {
    trace("field '" + i + "' has value " + my_ds.properties[i]);
}
```

DataSet.readOnly

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.readOnly

Description

Property; a Boolean value that specifies whether this collection can be modified (`false`) or is read-only (`true`). Setting this property to `true` prevents updates to the collection. The default value is `false`.

You can also set this property in the Property inspector.

Example

The following example makes the `DataSet` object named `my_ds` read-only, and then attempts to change the value of a property that belongs to the current item in the collection. This attempt throws a `DataSetError` exception.

```
import mx.data.components.datasetclasses.DataSetError;
my_ds.readOnly = true;
try {
    // This throws an exception.
    my_ds.addItem({name:'Joe'});
} catch (e:DataSetError) {
    // Sort specified 'name' doesn't exist for DataSet 'my_ds'.
    trace("DataSetError >> " + e.message);
}
```

See also

[DataSet.currentItem](#)

DataSet.removeAll()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.removeAll()
```

Returns

Nothing.

Description

Method; removes all items in the `DataSet` collection.

Example

The following example removes all the items in the `DataSet` collection `contact_ds`:

```
contact_ds.removeAll();
```

DataSet.removeItem

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.removeItem = function (eventObj:Object):Void {
    // ...
};
dataSetInstance.addEventListener("removeItem", listenerObject);
```

Usage 2:

```
on (removeItem) {
    // ...
}
```

Description

Event; generated just before a new item is deleted from this collection.

If you set the `result` property of the event object to `false`, the delete operation is canceled; if you set it to `true`, the delete operation is allowed.

The event object (*eventObj*) contains the following properties:

`target` The `DataSet` object that generated the event.

`type` The string "removeItem".

`item` A reference to the item in the collection to be removed.

`result` A Boolean value that specifies whether the item should be removed. By default, this value is `true`.

Example

In the following example, an `on(removeItem)` event handler cancels the deletion of the new item if a user-defined function named `userHasAdminPrivs()` returns `false`; otherwise, the deletion is allowed:

```
on (removeItem) {
  if (globalObj.userHasAdminPrivs()) {
    // Allow the item deletion.
    eventObj.result = true;
  } else {
    // Don't allow the item deletion; user doesn't have admin privileges.
    eventObj.result = false;
  }
}
```

The following `removeItem` event handler cancels the removal of the existing item if a user-defined function named `userHasAdminPrivs()` returns `false`; otherwise, the item removal is allowed:

```
function userHasAdminPrivs():Boolean {
  return false; // change this to true to allow inserts
}
function removeItemListener(evt_obj:Object):Void {
  if (userHasAdminPrivs()) {
    // Allow the item removal.
    evt_obj.result = true;
    trace("Item removed");
  } else {
    // Don't allow item removal; user doesn't have admin privileges.
    evt_obj.result = false;
    trace("Error, insufficient permissions");
  }
}
my_ds.addEventListener("removeItem", removeItemListener);
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});
my_ds.removeItemAt(0);
```

See also

[DataSet.addItem](#)

DataSet.removeItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.removeItem([item])
```

Parameters

item The item to be removed. This parameter is optional.

Returns

A Boolean value. Returns `true` if the item was successfully removed; otherwise, returns `false`.

Description

Method; removes the specified item from the collection, or removes the current item if the *item* parameter is omitted. This operation is logged to `DataSet.deltaPacket` if `DataSet.logChanges` is `true`.

Example

The following code removes the item at the current iterator position. To test this example, add a `DataSet` component to the Stage, and give it an instance name of `my_ds`. Add the following code to Frame 1 of the main timeline:

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

trace(my_ds.getLength()); // 5
trace(my_ds.currentItem.name); // Frank
my_ds.removeItem();
trace(my_ds.getLength()); // 4
trace(my_ds.currentItem.name); // Michael
```

See also

[DataSet.deltaPacket](#), [DataSet.logChanges](#)

DataSet.removeItemAt()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.removeItemAt(index)
```

Parameters

index A number greater than or equal to 0. This number is the index of the item to remove.

Returns

A Boolean value indicating whether the item was removed.

Description

Method; removes the item at the specified index. The indices after the removed index collapse by one.

This method triggers the `modelChanged` event with the event name `removeItems`.

In addition, it triggers the [DataSet.removeItem](#) event, which contains the `result` and `item` properties. The `result` property is used to determine if the item (referenced by the `item` property of the event) can be removed. By default, the `result` property is set to `true`. If no event listener is specified for the `removeItem` event, the item is removed by default.

An event listener can stop the item from being removed by listening for the `removeItem` event and setting the `result` property of the event to `false`, as shown in the following example:

```
function removeItem(evt_obj:Object):Void {  
    // Don't allow anyone to remove the item with customerId == 0.  
    evt_obj.result = (evt_obj.item.customerId != 0);  
}
```

Example

The following example removes an item from the data set at the first position:

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

trace(my_ds.getLength()); // 5
trace(my_ds.currentItem.name); // Frank
my_ds.removeItemAt(0);
trace(my_ds.getLength()); // 4
trace(my_ds.currentItem.name); // Frank
```

DataSet.removeRange()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.removeRange()
```

Returns

Nothing.

Description

Method; removes the current end point settings specified by [DataSet.setRange\(\)](#) for the current iterator.

Example

```
my_ds.addSort("name_id", ["name", "id"]);
my_ds.setRange(["Bobby", 105],["Cathy", 110]);
while (my_ds.hasNext()) {
    my_ds.gradeLevel = "5"; // Change all of the grades in this range.
    my_ds.next();
}
my_ds.removeRange();
my_ds.removeSort("name_id");
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.hasNext\(\)](#), [DataSet.next\(\)](#),
[DataSet.removeSort\(\)](#), [DataSet.setRange\(\)](#)

DataSet.removeSort()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.removeSort(sortName)
```

Parameters

sortName A string that specifies the name of the sort to remove.

Returns

Nothing.

Description

Method; removes the specified sort from this DataSet object if the sort exists. If the specified sort does not exist, this method throws a `DataSetError` exception.

Example

The following example creates a range of items in the DataSet component and modifies the `gradeLevel` property of each item. To test this example, drag a DataSet component to the Stage, and give it an instance name of `my_ds`. With the DataSet component selected, create three new properties in the schema of the DataSet component by using the Schema tab in the Component inspector. Name the new properties `name`, `id`, and `gradeLevel`, and give them the data types of String, Number, and Number respectively. Add a copy of the `DataBindingClasses` compiled clip from the Classes common library (Window > Common Libraries > Classes) and add the following ActionScript to Frame 1 of the main timeline:

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});

my_ds.addSort("name_id", ["name", "id"]);
my_ds.setRange(["Bobby", 105], ["Cathy", 110]);
while (my_ds.hasNext()) {
    my_ds.gradeLevel = "5"; // Change all of the grades in this range.
    my_ds.next();
}
my_ds.removeRange();
my_ds.removeSort("name_id");

for (var i=0; i<my_ds.length; i++) {
    trace(my_ds.items[i].name + " > " + my_ds.items[i].gradeLevel);
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.hasNext\(\)](#), [DataSet.next\(\)](#),
[DataSet.removeRange\(\)](#), [DataSet.setRange\(\)](#)

DataSet.resolveDelta

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.resolveDelta = function (eventObj:Object):Void {
    // ...
};
dataSetInstance.addEventListener("resolveDelta", listenerObject);
```

Usage 2:

```
on (resolveDelta) {
    // ...
}
```

Description

Event; broadcast when `DataSet.deltaPacket` is assigned a delta packet whose transaction ID matches that of a delta packet previously retrieved from the `DataSet` object, and that has messages associated with any of the deltas or `DeltaItem` objects contained by that delta packet.

This event gives you the chance to reconcile any error returned from the server while attempting to apply changes previously submitted. Typically, you use this event to display a “reconcile dialog box” with the conflicting values, allowing the user to make appropriate modifications to the data so that it can be re-sent.

The event object (*eventObj*) contains the following properties:

target The `DataSet` object that generated the event.

type The string "resolveDelta".

data An array of deltas and associated `DeltaItem` objects that have nonzero length messages.

Example

The following example displays a form called `reconcileForm` (not shown) and calls a method on that form object (`setReconcileData()`) that allows the user to reconcile any conflicting values returned by the server:

```
import mx.data.components.datasetclasses.*;
my_ds.addEventListener("resolveDelta", onResolveDelta);
function onResolveDelta(eventObj:Object) {
    reconcileForm.visible = true;
    reconcileForm.setReconcileData(eventObj.data);
}
// in the reconcileForm code
function setReconcileData(data:Array):Void {
    var di:DeltaItem;
    var ops:Array = ["property", "method"];
    var cl:Array;
    // change list
    var msg:String;
    for (var i = 0; i<data.length; i++) {
        cl = data[i].getChangeList();
        for (var j = 0; j<cl.length; j++) {
            di = cl[j];
            msg = di.message;
            if (msg.length>0) {
                trace("The following problem occurred '"+msg+"' while performing a
''+ops[di.kind]+' modification on/with '"+di.name+"' current server
value ['"+di.curValue+"], value sent ['"+di.newValue+"'] Please fix!");
            }
        }
    }
}
```

DataSet.saveToSharedObj()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.saveToSharedObj(objName, [localPath])
```

Parameters

objName A string that specifies the name of the shared object to create. The name can include forward slashes (for example, “work/addresses”). Spaces and the following characters are not allowed in the specified name:

~ % & \ ; : " ' , < > ? #

localPath An optional string parameter that specifies the full or partial path to the SWF file that created the shared object. This string is used to determine where the object is stored on the user’s computer. The default value is the SWF file’s full path.

Returns

Nothing.

Description

Method; saves all of the relevant data needed to restore this DataSet collection to a shared object. This allows users to work when disconnected from the source data, if it is a network resource. This method overwrites any data that might exist within the specified shared object for this DataSet collection. To restore a DataSet collection from a shared object, use [DataSet.loadFromSharedObj\(\)](#). Note that the instance name of the DataSet collection is used to identify the data within the specified shared object.

If the shared object can’t be created or there is a problem flushing the data to it, this method throws a `DataSetError` exception.

Example

The following example calls `saveToSharedObj()` in a `try..catch` block and displays an error if there is a problem saving the data to the shared object.

```
import mx.data.components.datasetclasses.DataSetError;
try {
    my_ds.saveToSharedObj("webapp/customerInfo");
} catch(e:DataSetError) {
    trace("Unable to create shared object");
}
```

See also

[DataSet.loadFromSharedObj\(\)](#)

DataSet.schema

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.schema

Description

Property; provides the XML representation of the schema for this DataSet object. The XML assigned to this property must have the following format:

```
<?xml version="1.0"?>
<properties>
  <property name="propertyName">
    <type name="dataType" />
    <encoder name="dataType">
      <options>
        <dataFormat>format options</dataFormat/>
      </options/>
    </encoder/>
    <kind name="dataKind">
      </kind>
    </property>
  <property> ... </property>
  ...
</properties>
```

A `DataSetError` exception is thrown if the XML specified does not follow the above format.

Example

The following example sets the schema of the data set `my_ds` to a new XML object containing appropriately formatted XML:

```
my_ds.schema = new XML("<properties><property name='billable'> ..etc.. </properties>");
```

DataSet.selectedIndex

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.selectedIndex

Description

Property; specifies the selected index in the collection. You can bind this property to the selected item in a DataGrid or List component, and vice versa. For a complete example that demonstrates this, see [“Creating an application with the DataSet component” on page 333](#).

Example

The following example sets the selected index of a DataSet object (*user_ds*) to the selected index in a DataGrid component (*user_dg*).

```
user_ds.selectedIndex = user_dg.selectedIndex;
```

DataSet.setIterator()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

dataSetInstance.setIterator(*iterator*)

Parameters

iterator An iterator object returned by a call to `DataSet.getIterator()`.

Returns

Nothing.

Description

Method; assigns the specified iterator to this DataSet object and makes it the current iterator. The specified iterator must come from a previous call to [DataSet.getIterator\(\)](#) on the DataSet object to which it is being assigned; otherwise, a `DataSetError` exception is thrown.

Example

```
import mx.data.to.ValueListIterator;
myIterator:ValueListIterator = my_ds.getIterator();
myIterator.sortOn(["name"]);
my_ds.setIterator(myIterator);
```

See also

[DataSet.getIterator\(\)](#)

DataSet.setRange()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.setRange(startValues, endValues)
```

Parameters

startValues An array of key values of the properties of the first transfer object in the range.

endValues An array of key values of the properties of the last transfer object in the range.

Returns

Nothing.

Description

Method; sets the end points for the current iterator. The end points define a range in which the iterator operates. This is only valid if a valid sort has been set for the current iterator by means of [DataSet.addSort\(\)](#).

Setting a range for the current iterator is more efficient than using a filter function if you want a grouping of values (see [DataSet.filterFunc](#)).

Example

The following example selects a range of students and traces each of their names to the Output panel:

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});
my_ds.addSort("name_id",["name", "id"]);
my_ds.setRange(["Bobby", 105],["Cathy", 110]);
while (my_ds.hasNext()) {
    trace(my_ds.name); // Bobby..Cathy
    my_ds.next();
}
```

See also

[DataSet.addSort\(\)](#), [DataSet.hasNext\(\)](#), [DataSet.next\(\)](#), [DataSet.removeRange\(\)](#), [DataSet.removeSort\(\)](#)

DataSet.skip()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.skip(offset)
```

Parameters

offset An integer specifying the number of records by which to move the iterator position.

Returns

Nothing.

Description

Method; moves the current iterator's position forward or backward in the collection by the amount specified by *offset*. Positive *offset* values move the iterator's position forward; negative values move it backward.

If the specified offset is beyond the beginning (or end) of the collection, the iterator is positioned at the beginning (or end) of the collection.

Example

The following example positions the current iterator at the first item in the collection, moves to the next-to-last item, and performs a calculation on a field belonging to that item:

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.first();
var itemsToSkip:Number = 3;
trace(my_ds.currentItem.name); // Billy
my_ds.skip(itemsToSkip);
trace(my_ds.currentItem.name); // Mally
```

DataSet.useSort()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
dataSetInstance.useSort(sortName, order)
```

Parameters

sortName A string that contains the name of the sort to use.

order An integer value that indicates the sort order for the sort; the value must be `DataSetIterator.Ascending` or `DataSetIterator.Descending`.

Returns

Nothing.

Description

Method; switches the sort for the current iterator to the one specified by *sortName*, if it exists. If the specified sort does not exist, a `DataSetError` exception is thrown.

To create a sort, use [DataSet.addSort\(\)](#).

Example

The following example uses `DataSet.hasSort()` to determine if a sort named "customer" exists. If it does, the code calls `DataSet.useSort()` to make "customer" the current sort. Otherwise, the code creates a sort by that name using `DataSet.addSort()`.

```
if (my_ds.hasSort("customer")) {
    my_ds.useSort("customer");
} else {
    my_ds.addSort("customer", ["customer"], DataSetIterator.Descending);
}
```

See also

[DataSet.applyUpdates\(\)](#), [DataSet.hasSort\(\)](#)

DateChooser component (Flash Professional only)

The DateChooser component is a calendar that allows users to select a date. It has buttons that allow users to scroll through months and click a date to select it. You can set parameters that indicate the month and day names, the first day of the week, and disabled dates, as well as highlighting the current date.

A live preview of each DateChooser instance reflects the values indicated by the Property inspector or Component inspector during authoring.

Using the DateChooser component (Flash Professional only)

The DateChooser can be used anywhere you want a user to select a date. For example, you could use a DateChooser component in a hotel reservation system with certain dates selectable and others disabled. You could also use the DateChooser component in an application that displays current events, such as performances or meetings, when a user chooses a date.

DateChooser parameters

You can set the following authoring parameters for each DateChooser component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

dayNames sets the names of the days of the week. The value is an array and the default value is ["S", "M", "T", "W", "T", "F", "S"].

disabledDays indicates the disabled days of the week. This parameter is an array that can have up to seven values. The default value is [] (an empty array).

firstDayOfWeek indicates which day of the week (0-6, with 0 being the first element of the dayNames array) is displayed in the first column of the date chooser. This property changes the display order of the day columns.

monthNames sets the month names that are displayed in the heading row of the calendar. The value is an array and the default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

showToday indicates whether to highlight today's date. The default value is `true`.

You can set the following additional parameters for each `DateChooser` component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector will have no visible effect.

You can write ActionScript to control these and additional options for the `DateChooser` component using its properties, methods, and events. For more information, see [“DateChooser class \(Flash Professional only\)” on page 417](#).

Creating an application with the `DateChooser` component

The following procedure explains how to add a `DateChooser` component to an application while authoring. In this example, the date chooser allows a user to pick a date for an airline reservation system. All dates before October 15th must be disabled. Also, a range in December must be disabled to create a holiday black-out period, and Mondays must be disabled.

To create an application with the `DateChooser` component:

1. Double-click the `DateChooser` component in the Components panel to add it to the Stage.
2. In the Property inspector, enter the instance name `flightCalendar`.
3. In the Actions panel, enter the following code on Frame 1 of the timeline to set the range of selectable dates:

```
flightCalendar.selectableRange = {rangeStart:new Date(2003, 9, 15),  
rangeEnd:new Date(2003, 11, 31)}
```

This code assigns a value to the `selectableRange` property in an `ActionScript` object that contains two `Date` objects with the variable names `rangeStart` and `rangeEnd`. This defines an upper and lower end of a range in which the user can select a date.

4. In the Actions panel, enter the following code on Frame 1 of the timeline to set a range of holiday disabled dates:

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),  
    rangeEnd: new Date(2003, 11, 26)}];
```

5. In the Actions panel, enter the following code on Frame 1 of the timeline to disable Mondays:

```
flightCalendar.disabledDays=[1];
```

6. Select Control > Test Movie.

To create a `DateChooser` component instance using `ActionScript`:

1. Drag the `DateChooser` component from the Components panel to the current document's library.
2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.DateChooser, "my_dc", 1);
```

This script uses the method “[UIObject.createClassObject\(\)](#)” on page 1362 to create the `DateChooser` instance, and then sizes and positions the grid.

3. Select Control > Test Movie.

Customizing the `DateChooser` component (Flash Professional only)

You can transform a `DateChooser` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

Using styles with the `DateChooser` component

You can set style properties to change the appearance of a `DateChooser` instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A DateChooser component supports the following styles:

Style	Theme	Description
themeColor	Halo	The glow color for the rollover and selected dates. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
backgroundColor	Both	The background color. The default value is OxEFEBEF (light gray).
borderColor	Both	The border color. The default value is Ox919999. The DateChooser component uses a solid single-pixel line as its border. This border cannot be modified through styles or skinning.
headerColor	Both	The background color for the component heading. The default color is white.
rolloverColor	Both	The background color of a rolled-over date. The default value is OxE3FFD6 (bright green) with the Halo theme and OxAAAAAA (light gray) with the Sample theme.
selectionColor	Both	The background color of the selected date. The default value is OxCDFFC1 (light green) with the Halo theme and OxEEEEEE (very light gray) with the Sample theme.
todayColor	Both	The background color for the today's date. The default value is Ox666666 (dark gray).
color	Both	The text color. The default value is Ox0B333C with the Halo theme and blank with the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is Ox848384 (dark gray).
embedFonts	Both	A Boolean value that indicates whether the font specified in fontFamily is an embedded font. This style must be set to true if fontFamily refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to true and fontFamily does not refer to an embedded font, no text is displayed. The default value is false.
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".

Style	Theme	Description
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> returns "none".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".

The DateChooser component uses four categories of text to display the month name, the days of the week, today's date, and regular dates. The text style properties set on the DateChooser component itself control the regular date text and provide defaults for the other text. To set text styles for specific categories of text, use the following class-level style declarations.

Declaration name	Description
HeaderDateText	The month name.
WeekDayStyle	The days of the week.
TodayStyle	Today's date.

The following example demonstrates how to set the month name and days of the week to a deep red color.

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```

Using skins with the DateChooser component

The DateChooser component uses skins to represent the forward and back month buttons and the today indicator. To skin the DateChooser component while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/DateChooser Assets/States folder in the library of one of the themes' FLA files. For more information, see "About skinning components" in *Using Components*.

Only the month scrolling buttons can be dynamically skinned in this component. A DateChooser component uses the following skin properties:

Property	Description
backMonthButtonUpSymbolName	The month back button up state. The default value is <code>backMonthUp</code> .
backMonthButtonDownSymbolName	The month back button pressed state. The default value is <code>backMonthDown</code> .
backMonthButtonDisabledSymbolName	The month back button disabled state. The default value is <code>backMonthDisabled</code> .
fwdMonthButtonUpSymbolName	The month forward button up state. The default value is <code>fwdMonthUp</code> .
fwdMonthButtonDownSymbolName	The month forward button pressed state. The default value is <code>fwdMonthDown</code> .
fwdMonthButtonDisabledSymbolName	The month forward button disabled state. The default value is <code>fwdMonthDisabled</code> .

The button symbols are used exactly as is, without applying colors or resizing. The size is determined by the symbol during authoring.

To create movie clip symbols for DateChooser skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library and then select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the DateChooser Assets folder to the library for your document.
4. Expand the DateChooser Assets/States folder in the library of your document.
5. Open the symbols that you want to customize for editing.
For example, open the `backMonthDown` symbol.
6. Customize the symbol as desired.
For example, change the tint of the arrow to red.
7. Repeat steps 5-6 for all symbols that you want to customize.
For example, change the tint of the forward arrow down symbol to match the back arrow.

8. Click the Back button to return to the main timeline.
9. Drag a DateChooser component to the Stage.
10. Select Control > Test Movie.

NOTE

The DateChooser Assets/States folder also contains a Day Skins folder with a single skin element, `cal_todayIndicator`. This element can be modified during authoring to customize the today indicator. However, it cannot be changed dynamically on a particular DateChooser instance to use a different symbol. In addition, the `cal_todayIndicator` symbol must be a solid single-color graphic, because the DateChooser component applies the `todayColor` color to the graphic as a whole. The graphic may have cut-outs, but keep in mind that the default text color for today's date is white and the default background for the DateChooser is white, so a cut-out in the middle of the today indicator skin element would make today's date unreadable unless either the background color or the today text color is also changed.

DateChooser class (Flash Professional only)

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > DateChooser

ActionScript Class Name `mx.controls.DateChooser`

The properties of the DateChooser class let you access the selected date and the displayed month and year. You can also set the names of the days and months, indicate disabled dates and selectable dates, set the first day of the week, and indicate whether the current date should be highlighted.

Setting a property of the DateChooser class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DateChooser.version);
```

NOTE

The code `trace(myDC.version);` returns `undefined`.

Method summary for the DateChooser class

There are no methods exclusive to the DateChooser class.

Methods inherited from the UIObject class

The following table lists the methods the DateChooser class inherits from the UIObject class. When calling these methods from the DateChooser object, use the form *dateChooserInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the DateChooser class inherits from the UIComponent class. When calling these methods from the DateChooser object, use the form *dateChooserInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the DateChooser class

The following table lists the properties that are exclusive to the DateChooser class.

Property	Description
<code>DateChooser.dayNames</code>	An array indicating the names of the days of the week.
<code>DateChooser.disabledDays</code>	An array indicating the days of the week that are disabled for all applicable dates in the date chooser.
<code>DateChooser.disabledRanges</code>	A range of disabled dates or a single disabled date.
<code>DateChooser.displayedMonth</code>	A number indicating an element in the <code>monthNames</code> array to display in the date chooser.
<code>DateChooser.displayedYear</code>	A number indicating the year to display.
<code>DateChooser.firstDayOfWeek</code>	A number indicating an element in the <code>dayNames</code> array to display in the first column of the date chooser.
<code>DateChooser.monthNames</code>	An array of strings indicating the month names.
<code>DateChooser.selectableRange</code>	A single selectable date or a range of selectable dates.
<code>DateChooser.selectedDate</code>	A Date object indicating the selected date.
<code>DateChooser.showToday</code>	A Boolean value indicating whether the current date is highlighted.

Properties inherited from the UIObject class

The following table lists the properties the DateChooser class inherits from the UIObject class. When accessing these properties from the DateChooser object, use the form `dateChooserInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.

Property	Description
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the DateChooser class inherits from the UIComponent class. When accessing these properties from the DateChooser object, use the form `dateChooserInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the DateChooser class

The following table lists the events that are exclusive to the DateChooser class.

Event	Description
<code>DateChooser.change</code>	Broadcast when a date is selected.
<code>DateChooser.scroll</code>	Broadcast when the month buttons are clicked.

Events inherited from the UIObject class

The following table lists the events the DateChooser class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.

Event	Description
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DateChooser class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

DateChooser.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // ...
};
dateChooserInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    //...
}
```

Description

Event; broadcast to all registered listeners when a date is selected.

The first usage example uses a dispatcher/listener event model. A component instance (*dateChooserInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateChooser` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date chooser `my_dc`, sends “_level0.my_dc” to the Output panel:

```
on (change) {
    trace(this);
}
```

Example

This example, written on a frame of the timeline, sends a message to the Output panel when a `DateChooser` instance called `my_dc` is changed. The first line of code creates a listener object called `form`. The second line defines a function for the `change` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event (in this example, `my_dc`).

```
// Create listener object.
var dcListener:Object = new Object();
dcListener.change = function(evt_obj:Object) {
    var thisDate:Date = evt_obj.target.selectedDate;
    trace("date selected: " + thisDate);
};

// Add listener object to date chooser.
my_dc.addEventListener("change", dcListener);
```


DateChooser.dayNames

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.dayNames

Description

Property; an array containing the names of the days of the week. Sunday is the first day (at index position 0) and the rest of the day names follow in order. The default value is ["S", "M", "T", "W", "T", "F", "S"].

Example

The following example changes the value of the days of the week:

```
my_dc.dayNames = new Array("Su", "Mo", "Tu", "We", "Th", "Fr", "Sa");
```

DateChooser.disabledDays

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.disabledDays

Description

Property; an array indicating the disabled days of the week. All the dates in a month that fall on the specified day are disabled. The elements of this array can have values from 0 (Sunday) to 6 (Saturday). The default value is [] (an empty array).

Example

The following example disables Sundays and Saturdays so that users can select only weekdays:

```
my_dc.disabledDays = [0, 6];
```

DateChooser.disabledRanges

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.disabledRanges

Description

Property; disables a single day or a range of days. This property is an array of objects. Each object in the array must be either a Date object that specifies a single day to disable, or an object that contains either or both of the properties `rangeStart` and `rangeEnd`, each of whose value must be a Date object. The `rangeStart` and `rangeEnd` properties describe the boundaries of the date range. If either property is omitted, the range is unbounded in that direction.

The default value of `disabledRanges` is undefined.

Specify a full date when you define dates for the `disabledRanges` property. For example, specify `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the Date object returns the current date and time. If you don't specify a time, the time is returned as 00:00:00.

Example

The following example defines an array with `rangeStart` and `rangeEnd` Date objects that disable the dates between May 7 and June 7:

```
my_dc.disabledRanges = [{rangeStart: new Date(2003, 4, 7), rangeEnd: new Date(2003, 5, 7)}];
```

The following example disables all dates after November 7:

```
my_dc.disabledRanges = [ {rangeStart: new Date(2003, 10, 7)} ];
```

The following example disables all dates before October 7:

```
my_dc.disabledRanges = [ {rangeEnd: new Date(2002, 9, 7)} ];
```

The following example disables only December 7:

```
my_dc.disabledRanges = [ new Date(2003, 11, 7) ];
```

The following example disables April 7 and April 21:

```
my_dc.disabledRanges = [ new Date(2003, 3, 7), new Date(2003, 3, 21) ];
```

DateChooser.displayedMonth

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.displayedMonth

Description

Property; a number indicating which month is displayed. The number indicates an element in the `monthNames` array, with 0 being the first month. The default value is the month of the current date.

Example

The following example sets the displayed month to December:

```
my_dc.displayedMonth = 11;
```

See also

[DateChooser.displayedYear](#)

DateChooser.displayedYear

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.displayedYear

Description

Property; a four-digit number indicating which year is displayed. The default value is the current year.

Example

The following example sets the displayed year to 2010:

```
my_dc.displayedYear = 2010;
```

See also

[DateChooser.displayedMonth](#)

DateChooser.firstDayOfWeek

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateChooserInstance.firstDayOfWeek
```

Description

Property; a number indicating which day of the week (0-6, 0 being the first element of the `dayNames` array) is displayed in the first column of the `DateChooser` component. Changing this property changes the order of the day columns but has no effect on the order of the `dayNames` property. The default value is 0 (Sunday).

Example

The following example sets the first day of the week to Monday:

```
// Sets the first day of the week to Monday in the calendar.  
my_dc.firstDayOfWeek = 1;
```

```
// Disables day 0 (Sunday). Even though Monday is now the first day in the  
// DateChooser, Sunday is still array index 0.  
my_dc.disabledDays = [0];
```

See also

[DateChooser.dayNames](#)

DateChooser.monthNames

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.monthNames

Description

Property; an array of strings indicating the month names at the top of the DateChooser component. The default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

Example

The following example sets the month names for the instance my_dc:

```
my_dc.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
    "Sept", "Oct", "Nov", "Dec"];
```

DateChooser.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();  
listenerObject.scroll = function(eventObject:Object) {  
    //...  
}  
dateChooserInstance.addEventListener("scroll", listenerObject)
```

Usage 2:

```
on (scroll) {  
    //...  
}
```

Description

Event; broadcast to all registered listeners when a month button is clicked.

The first usage example uses a dispatcher/listener event model. A component instance (*myDC*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The scroll event's event object has an additional property, `detail`, that can have one of the following values: `nextMonth`, `previousMonth`, `nextYear`, `previousYear`.

Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateChooser` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date chooser `myDC`, sends “_level0.myDC” to the Output panel:

```
on (scroll) {  
    trace(this);  
}
```

Example

This example, written on a frame of the timeline, sends a message to the Output panel when a month button is clicked on a `DateChooser` instance called `my_dc`. The first line of code creates a listener object called `form`. The second line defines a function for the `scroll` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `evt_obj`, to generate a message.

```
// Create listener object.  
var dcListener:Object = new Object();  
dcListener.scroll = function(evt_obj:Object) {  
    trace(evt_obj.detail);  
};  
  
// Add listener object to date chooser.  
my_dc.addEventListener("scroll", dcListener);
```

DateChooser.selectableRange

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.selectableRange

Description

Property; sets a single selectable date or a range of selectable dates. The user cannot scroll beyond the selectable range. The value of this property is an object that consists of two `Date` objects named `rangeStart` and `rangeEnd`. The `rangeStart` and `rangeEnd` properties designate the boundaries of the selectable date range. If only `rangeStart` is defined, all the dates after `rangeStart` are enabled. If only `rangeEnd` is defined, all the dates before `rangeEnd` are enabled. The default value is `undefined`.

If you want to enable only a single day, you can use a single `Date` object as the value of `selectableRange`.

Specify a full date when you define dates—for example, `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the `Date` object returns the current date and time. If you don't specify a time, the time is returned as `00:00:00`.

The value of `DateChooser.selectedDate` is set to `undefined` if it falls outside the selectable range.

The values of `DateChooser.displayedMonth` and `DateChooser.displayedYear` are set to the nearest last month in the selectable range if the current month falls outside the selectable range. For example, if the current displayed month is August, and the selectable range is from June 2003 to July, 2003, the displayed month changes to July 2003.

Example

The following example defines the selectable range as the dates between and including May 7 and June 7:

```
my_dc.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable range as the dates after and including May 7:

```
my_dc.selectableRange = {rangeStart: new Date(2005, 4, 7)};
```

The following example defines the selectable range as the dates before and including June 7:

```
my_dc.selectableRange = {rangeEnd: new Date(2005, 5, 7)};
```

The following example defines the selectable date as June 7 only:

```
my_dc.selectableRange = new Date(2005, 5, 7);
```

DateChooser.selectedDate

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.selectedDate

Description

Property; a Date object that indicates the selected date if that value falls within the value of the selectableRange property. The default value is undefined.

You cannot set the selectedDate property within a disabled range, outside a selectable range, or on a day that has been disabled. If this property is set to one of these dates, the value is undefined.

Example

The following example sets the selected date to June 7:

```
my_dc.selectedDate = new Date(2005, 5, 7);
```


DateChooser.showToday

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateChooserInstance.showToday

Description

Property; a Boolean value that determines whether the current date is highlighted. The default value is `true`.

Example

The following example turns off the highlighting on today's date:

```
my_dc.showToday = false;
```


DateField component (Flash Professional only)

The DateField component is a nonselectable text field that displays the date with a calendar icon on its right side. If no date has been selected, the text field is blank and the month of today's date is displayed in the date chooser. When a user clicks anywhere inside the bounding box of the date field, a date chooser pops up and displays the dates in the month of the selected date. When the date chooser is open, users can use the month scroll buttons to scroll through months and years and select a date. When a date is selected, the date chooser closes and the selection is entered in the date field.

NOTE

The date field is cleared when a user selects the same date twice. To prevent users from accidentally deselecting their desired date, see the example for [DateField.selectedDate on page 459](#).

The live preview of the DateField component does not reflect the values indicated by the Property inspector or Component inspector during authoring, because it is a pop-up component that is not visible during authoring.

Using the DateField component (Flash Professional only)

The DateField component can be used anywhere you want a user to select a date. For example, you could use a DateField component in a hotel reservation system with certain dates selectable and others disabled. You could also use the DateField component in an application that displays current events, such as performances or meetings, when a user chooses a date.

DateField parameters

You can set the following authoring parameters for each DateField component instance in the Property inspector or in the Component inspector:

dayNames sets the names of the days of the week. The value is an array and the default value is ["S", "M", "T", "W", "T", "F", "S"].

disabledDays indicates the disabled days of the week. This parameter is an array that can have up to seven values. The default value is [] (an empty array).

firstDayOfWeek indicates which day of the week (0-6, with 0 being the first element of dayNames array) is displayed in the first column of the date chooser. This property changes the display order of the day columns.

The default value is 0, which is "S" for Sunday.

monthNames sets the month names that are displayed in the heading row of the calendar. The value is an array and the default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

showToday indicates whether to highlight today's date. The default value is true.

You can write ActionScript to control these and additional options for the DateField component using its properties, methods, and events. For more information, see [“DateField class \(Flash Professional only\)” on page 439](#).

Creating an application with the DateField component

The following procedure explains how to add a DateField component to an application while authoring. In this example, the DateField component allows a user to pick a date for an airline reservation system. All dates before today's date must be disabled. Also, a 15-day range in December must be disabled to create a holiday black-out period. Also, some flights are not available on Mondays, so all Mondays must be disabled for those flights.

To create an application with the DateField component:

1. Double-click the DateField component in the Components panel to add it to the Stage.
2. In the Property inspector, enter the instance name **flightCalendar**.
3. In the Actions panel, enter the following code on Frame 1 of the timeline to set the range of selectable dates:

```
flightCalendar.selectableRange = {rangeStart:new Date(2001, 9, 1),  
rangeEnd:new Date(2003, 11, 1)};
```

This code assigns a value to the `selectableRange` property in an ActionScript object that contains two `Date` objects with the variable names `rangeStart` and `rangeEnd`. This defines an upper and lower end of a range within which the user can select a date.

4. In the Actions panel, enter the following code on Frame 1 of the timeline to set the ranges of disabled dates, one during December, and one for all dates before the current date:

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),  
    rangeEnd: new Date(2003, 11, 31)}, {rangeEnd: new Date(2003, 6, 16)}];
```

5. In the Actions panel, enter the following code on Frame 1 of the timeline to disable Mondays:

```
flightCalendar.disabledDays=[1];
```

6. Control > Test Movie.

To create a `DateField` component instance using ActionScript:

1. Drag the `DateField` component from the Components panel to the current document's library.

This adds the component to the library, but doesn't make it visible in the application.

2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.DateField, "my_df", 1);
```

This script uses the method `UIObject.createClassObject()` to create the `DateField` instance.

3. Select Control > Test Movie.

Customizing the `DateField` component (Flash Professional only)

You can transform a `DateField` component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see `UIObject.setSize()`). Setting the width does not change the dimensions of the date chooser in the `DateField` component. However, you can use the `pullDown` property to access the `DateChooser` component and set its dimensions.

Using styles with the DateField component

You can set style properties to change the appearance of a date field instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

The DateField component supports the following styles:

Style	Theme	Description
themeColor	Halo	The glow color for the rollover and selected dates. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen"
backgroundColor	Both	The background color. The default value is OxEFEFEBF (light gray).
borderColor	Both	The border color. The default value is Ox919999. The DateField component's drop-down list uses a solid single-pixel line as its border. This border cannot be modified through styles or skinning.
headerColor	Both	The background color for the drop-down heading. The default color is white.
rolloverColor	Both	The background color of a rolled-over date. The default value is OxE3FFD6 (bright green) with the Halo theme and OxAAAAAA (light gray) with the Sample theme.
selectionColor	Both	The background color of the selected date. The default value is a OxCDFFC1 (light green) with the Halo theme and OxEEEEEE (very light gray) with the Sample theme.
todayColor	Both	The background color for the today's date. The default value is Ox666666 (dark gray).
color	Both	The text color. The default value is Ox0B333C with the Halo theme and blank with the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is Ox848384 (dark gray).

Style	Theme	Description
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `DateField` component uses four categories of text to display the month name, the days of the week, today's date, and regular dates. The text style properties set on the `DateField` component itself control the regular date text and the text displayed in the collapsed state, and provide defaults for the other text. To set text styles for specific categories of text, use the following class-level style declarations.

Declaration name	Description
<code>HeaderDateText</code>	The month name.
<code>WeekDayStyle</code>	The days of the week.
<code>TodayStyle</code>	Today's date.

The following example demonstrates how to set the month name and days of the week to a deep red color.

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```

Using skins with the DateField component

The DateField component uses skins to represent the visual states of the pop-up icon, a RectBorder instance for the border around the text input, and a DateChooser instance for the pop-up. To skin the pop-up icon while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/DateField Assets/States folder in the library of one of the themes' FLA files. For more information, see “About skinning components” in *Using Components*. For information about skinning the RectBorder and DateChooser instances, see “RectBorder class” on page 1063 and “Using skins with the DateChooser component” on page 415.

In addition to the skins used by the subcomponents mentioned earlier, a DateField component uses the following skin properties to dynamically skin the pop-up icon:

Property	Description
openDateUp	The up state of the pop-up icon.
openDateDown	The down state of the pop-up icon.
openDateOver	The over state of the pop-up icon.
openDateDisabled	The disabled state of the pop-up icon.

To create movie clip symbols for DateField skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library and then select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme's Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the DateField Assets folder to the library of your document.
4. Expand the DateField Assets folder in the library of your document.
5. Make sure that the DateFieldAssets symbol is selected for Export in First Frame.
6. Expand the DateField Assets/States folder in the library of your document.
7. Open the symbols that you want to customize for editing.
For example, open the openIconUp symbol.
8. Customize the symbol as desired.
For example, draw a down arrow over the calendar image.
9. Repeat steps 7-8 for all symbols that you want to customize.
For example, draw a down arrow over all of the symbols.
10. Click the Back button to return to the main timeline.

11. Drag a DateField component to the Stage.
12. Select Control > Test Movie.

DateField class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > ComboBase > DateField

ActionScript Class Name mx.controls.DateField

The properties of the DateField class let you access the selected date and the displayed month and year. You can also set the names of the days and months, indicate disabled dates and selectable dates, set the first day of the week, and indicate whether the current date should be highlighted.

Setting a property of the DateField class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DateField.version);
```

NOTE

The code `trace(myDateFieldInstance.version);` returns `undefined`.

Method summary for the DateField class

The following table lists methods of the DateField class.

Method	Description
DateField.close()	Closes the pop-up DateChooser subcomponent.
DateField.open()	Opens the pop-up DateChooser subcomponent.

Methods inherited from the UIObject class

The following table lists the methods the DateField class inherits from the UIObject class. When calling these methods from the DateField object, use the form *dateFieldInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the DateField class inherits from the UIComponent class. When calling these methods from the DateField object, use the form *dateFieldInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the DateField class

The following table lists properties of the DateField class.

Property	Description
<code>DateField.dateFormatter</code>	A function that formats the date to be displayed in the text field.
<code>DateField.dayNames</code>	An array indicating the names of the days of the week.
<code>DateField.disabledDays</code>	An array indicating the disabled days of the week.
<code>DateField.disabledRanges</code>	A range of disabled dates or a single disabled date.
<code>DateField.displayedMonth</code>	A number indicating which element in the <code>monthNames</code> array to display.
<code>DateField.displayedYear</code>	A number indicating the year to display.
<code>DateField.firstDayOfWeek</code>	A number indicating an element in the <code>dayNames</code> array to display in the first column of the DateField component.
<code>DateField.monthNames</code>	An array of strings indicating the month names.
<code>DateField.pullDown</code>	A reference to the DateChooser subcomponent. This property is read-only.
<code>DateField.selectableRange</code>	A single selectable date or a range of selectable dates.
<code>DateField.selectedDate</code>	A Date object indicating the selected date.
<code>DateField.showToday</code>	A Boolean value indicating whether the current date is highlighted.

Properties inherited from the UIObject class

The following table lists the properties the DateField class inherits from the UIObject class.

When accessing these properties from the DateField object, use the form

dateFieldInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the DateField class inherits from the UIComponent class. When accessing these properties from the DateField object, use the form *dateFieldInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the DateField class

The following table lists events of the DateField class.

Event	Description
<code>DateField.change</code>	Broadcast when a date is selected.
<code>DateField.close</code>	Broadcast when the DateChooser subcomponent closes.
<code>DateField.open</code>	Broadcast when the DateChooser subcomponent opens.
<code>DateField.scroll</code>	Broadcast when the month buttons are clicked.

Events inherited from the UIObject class

The following table lists the events the DateField class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DateField class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

DateField.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // ...
};
dateFieldInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when a date is selected.

The first usage example uses a dispatcher/listener event model. A component instance (*dateFieldInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `my_df`, sends “_level0.my_df” to the Output panel:

```
on (change) {
    trace(this);
}
```

Example

The following example, written on a frame of the timeline, sends a message to the Output panel when a date field called `my_df` is changed. The first line of code creates a listener object called `dfListener`. The second line defines a function for the `change` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `evt_obj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `my_df`. The `DateField.selectedDate` property is accessed from the event object's `target` property. The last line calls `EventDispatcher.addEventListener()` from `my_df` and passes it the `change` event and the `dfListener` listener object as parameters.

```
// Create listener object.
var dfListener:Object = new Object();
dfListener.change = function(evt_obj:Object){
    var thisDate:Date = evt_obj.target.selectedDate;
    trace("date selected: " + thisDate);
}

// Add listener object to DateField.
my_df.addEventListener("change", dfListener);
```

DateField.close()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.close()
```

Returns

Nothing.

Description

Method; closes the pop-up menu.

Example

The following code closes the date chooser pop-up of the `my_df` date field instance when the button `my_btn` is clicked:

```
//Create listener object.
var btnListener:Object = new Object();
btnListener.click = function() {
    my_df.close();
};

//Add Button listener.
my_btn.addEventListener("click", btnListener);
```

DateField.close

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object) {
    // ...
};
dateFieldInstance.addEventListener("close", listenerObject);
```

Usage 2:

```
on (close) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the DateChooser subcomponent closes after a user clicks outside the icon or selects a date.

The first usage example uses a dispatcher/listener event model. A component instance (*dateFieldInstance*) dispatches an event (in this case, `close`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `my_df`, sends “_level0.my_df” to the Output panel:

```
on (close) {
    trace(this);
}
```

Example

The following example, written on a frame of the timeline, sends a message to the Output panel when the date chooser in `my_df` closes. The first line of code creates a listener object called `dfListener`. The second line defines a function for the `close` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `evt_obj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `my_df`. The `selectedDate` property is accessed from the event object’s `target` property. The last line calls `EventDispatcher.addEventListener()` from `my_df` and passes it the `close` event and the `dfListener` listener object as parameters.

```
//Create listener object.
var dfListener:Object = new Object();
dfListener.close = function(evt_obj:Object){
    trace("PullDown Closed" + evt_obj.target.selectedDate);
}
//Add listener object to DateField.
my_df.addEventListener("close", dfListener);
```

DateField.dateFormatter

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.dateFormatter

Description

Property; a function that formats the date to be displayed in the text field. The function must receive a Date object as parameter, and return a string in the format to be displayed.

Example

The following example sets the function to return the format of the date to be displayed:

```
my_df.dateFormatter = function(d:Date){
    return d.getFullYear()+"/ "+(d.getMonth()+1)+"/ "+d.getDate();
};
```

DateField.dayNames

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.dayNames

Description

Property; an array containing the names of the days of the week. Sunday is the first day (at index position 0) and the other day names follow in order. The default value is ["S", "M", "T", "W", "T", "F", "S"].

Example

The following example changes the value of the fifth day of the week (Thursday) from “T” to “R”:

```
my_df.dayNames[4] = "R";
```

The following example changes the value of all the days, accordingly:

```
my_df.dayNames = new Array("Su", "Mo", "Tu", "We", "Th", "Fr", "Sa");
```

DateField.disabledDays

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.disabledDays
```

Description

Property; an array indicating the disabled days of the week. All the dates in a month that fall on the specified day are disabled. The elements of this array can have values between 0 (Sunday) and 6 (Saturday). The default value is [] (an empty array).

Example

The following example disables Sundays and Saturdays so that users can select only weekdays:

```
my_df.disabledDays = [0, 6];
```

DateField.disabledRanges

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.disabledRanges
```

Description

Property; disables a single day or a range of days. This property is an array of objects. Each object in the array must be either a `Date` object specifying a single day to disable, or an object containing either or both of the properties `rangeStart` and `rangeEnd`, each of whose value must be a `Date` object. The `rangeStart` and `rangeEnd` properties describe the boundaries of the date range. If either property is omitted, the range is unbounded in that direction.

The default value of `disabledRanges` is undefined.

Specify a full date when you define dates for the `disabledRanges` property—for example, `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

Example

The following example defines an array with `rangeStart` and `rangeEnd` `Date` objects that disable the dates between May 7 and June 7:

```
my_df.disabledRanges = [ {rangeStart: new Date(2003, 4, 7), rangeEnd: new Date(2003, 5, 7)}];
```

The following example disables all dates after November 7:

```
my_df.disabledRanges = [ {rangeStart: new Date(2003, 10, 7)} ];
```

The following example disables all dates before October 7:

```
my_df.disabledRanges = [ {rangeEnd: new Date(2002, 9, 7)} ];
```

The following example disables only December 7:

```
my_df.disabledRanges = [ new Date(2003, 11, 7) ];
```

The following example disables December 7 and December 20:

```
my_df.disabledRanges = [ new Date(2003, 11, 7), new Date(2003, 11, 20)];
```

DateField.displayedMonth

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.displayedMonth

Description

Property; a number indicating which month is displayed. The number indicates an element in the `monthNames` array, with 0 being the first month. The default value is the month of the current date.

Example

The following example sets the displayed month to December:

```
my_df.displayedMonth = 11;
```

See also

[DateField.displayedYear](#)

DateField.displayedYear

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.displayedYear
```

Description

Property; a number indicating which year is displayed. The default value is the current year.

Example

The following example sets the displayed year to 2010:

```
my_df.displayedYear = 2010;
```

See also

[DateField.displayedMonth](#)

DateField.firstDayOfWeek

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.firstDayOfWeek

Description

Property; a number indicating which day of the week (0-6, 0 being the first element of the `dayNames` array) is displayed in the first column of the DateField component. Changing this property changes the order of the day columns but has no effect on the order of the `dayNames` property. The default value is 0 (Sunday).

Example

The following example sets the first day of the week to Monday:

```
my_df.firstDayOfWeek = 1;
```

See also

[DateField.dayNames](#)

DateField.monthNames

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.monthNames

Description

Property; an array of strings indicating the month names at the top of the DateField component. The default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

Example

The following example sets the month names for the instance `my_df`:

```
my_df.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
    "Sept", "Oct", "Nov", "Dec"];
```

DateField.open()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.open()
```

Returns

Nothing.

Description

Method; opens the pop-up DateChooser subcomponent.

Example

The following code opens the date chooser pop-up of the `my_df` date field instance when the button `my_btn` is clicked:

```
//Create listener object.  
var btnListener:Object = new Object();  
btnListener.click = function() {  
    my_df.open();  
};  
  
//Add Button listener.  
my_btn.addEventListener("click", btnListener);
```

DateField.open

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.open = function(eventObject:Object) {
    // ...
};
dateFieldInstance.addEventListener("open", listenerObject);
```

Usage 2:

```
on (open) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when a DateChooser subcomponent opens after a user clicks the icon.

The first usage example uses a dispatcher/listener event model. A component instance (*dateFieldInstance*) dispatches an event (in this case, *open*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `my_df`, sends “_level0.my_df” to the Output panel:

```
on (open) {
    trace(this);
}
```

Example

The following example, written on a frame of the timeline, sends a message to the Output panel when a date field called `my_df` is opened. The message ends with the index number for the current month. The first line of code creates a listener object called `dfListener`. The second line defines a function for the `open` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `evt_obj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `my_df`. The `DateField.selectedDate` property is accessed from the event object's `target` property. The last line calls `EventDispatcher.addListener()` from `my_df` and passes it the `open` event and the `dfListener` listener object as parameters.

```
// Create listener object.
var dfListener:Object = new Object();
dfListener.open = function(evt_obj:Object){
    trace("PullDown Opened" + evt_obj.target.displayedMonth);
}
// Add listener object to DateField.
my_df.addListener("open", dfListener);
```

DateField.pullDown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.pullDown

Description

Property (read-only); a reference to the DateChooser component contained by the DateField component. The DateChooser subcomponent is instantiated when a user clicks on the DateField component. However, if the `pullDown` property is referenced before the user clicks on the component, the DateChooser is instantiated and then hidden.

Example

The following example sets the visibility of the DateChooser subcomponent to `false` and then sets the size of the DateChooser subcomponent to 300 pixels high and 300 pixels wide:

```
my_df.pullDown._visible = false;
my_df.pullDown.setSize(300, 300);
```

DateField.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
    // ...
};
dateFieldInstance.addEventListener("scroll", listenerObject);
```

Usage 2:

```
on (scroll) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when a month button is clicked.

The first usage example uses a dispatcher/listener event model. A component instance (*dateFieldInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `scroll` event's event object has an additional property, `detail`, that can have one of the following values: `nextMonth`, `previousMonth`, `nextYear`, `previousYear`.

Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `my_df`, sends “_level0.my_df” to the Output panel:

```
on (scroll) {
    trace(this);
}
```

Example

The following example, written on a frame of the timeline, sends a message to the Output panel when a user clicks a month button on a `DateField` instance called `my_df`. The first line of code creates a listener object called `dfListener`. The second line defines a function for the `scroll` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `evt_obj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `my_df`. The last line calls `EventDispatcher.addEventListener()` from `my_df` and passes it the `scroll` event and the `dfListener` listener object as parameters.

```
// Create listener object.
var dfListener:Object = new Object();
dfListener.scroll = function(evt_obj:Object) {
    trace(evt_obj.detail);
};

// Add listener object to DateField.
my_df.addEventListener("scroll", dfListener);
```

DateField.selectableRange

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.selectableRange

Description

Property; sets a single selectable date or a range of selectable dates. The value of this property is an object that consists of two `Date` objects named `rangeStart` and `rangeEnd`. The `rangeStart` and `rangeEnd` properties designate the boundaries of the selectable date range. If only `rangeStart` is defined, all the dates after `rangeStart` are enabled. If only `rangeEnd` is defined, all the dates before `rangeEnd` are enabled. The default value is `undefined`.

If you want to enable only a single day, you can use a single `Date` object as the value of `selectableRange`.

Specify a full date when you define dates—for example, `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

The value of `DateField.selectedDate` is set to `undefined` if it falls outside the selectable range.

The values of `DateField.displayedMonth` and `DateField.displayedYear` are set to the nearest last month in the selectable range if the current month falls outside the selectable range. For example, if the current displayed month is August, and the selectable range is from June 2003 to July 2003, the displayed month changes to July 2003.

Example

The following example defines the selectable range as the dates between and including May 7 and June 7:

```
my_df.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable range as the dates after and including May 7:

```
my_df.selectableRange = {rangeStart: new Date(2003, 4, 7)};
```

The following example defines the selectable range as the dates before and including June 7:

```
my_df.selectableRange = {rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable date as June 7 only:

```
my_df.selectableRange = new Date(2003, 5, 7);
```

DateField.selectedDate

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
dateFieldInstance.selectedDate
```

Description

Property; a Date object that indicates the selected date if that value falls within the value of the selectableRange property. The default value is undefined.

Example

The following example sets the selected date to June 7:

```
my_df.selectedDate = new Date(2003, 5, 7);
```

The following example uses a DateField instance named my_df on the Stage to show how to disable an already selected date (otherwise, the user can click it again to clear the date field entry):

```
function dfListener(evt_obj:Object):Void {  
    my_df.disabledRanges = [my_df.selectedDate];  
}  
my_df.addEventListener("change", dfListener);
```

DateField.showToday

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

dateFieldInstance.showToday

Description

Property; a Boolean value that determines whether the current date is highlighted. The default value is `true`.

Example

The following example turns off the highlighting on today's date:

```
my_df.showToday = false;
```

Inheritance Object > Delegate

ActionScript Class Name mx.utils.Delegate

The Delegate class lets you run a function in a specific scope. This class is provided so that you can dispatch the same event to two different functions (see “Delegating events to functions” in *Using Components*), and so that you can call functions within the scope of the containing class.

When you pass a function as a parameter to `EventDispatcher.addEventListener()`, the function is invoked in the scope of the broadcaster component instance, not the object in which it is declared (see “Delegating the scope of a function” in *Using Components*). You can call `Delegate.create()` to call the function within the scope of the declaring object.

Method summary for the Delegate class

The following table lists the method of the Delegate class.

Method	Description
<code>Delegate.create()</code>	A static method that allows you to run a function in a specific scope.

Delegate.create()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Delegate.create(scopeObject, function)
```

Parameters

scopeObject A reference to an object. This is the scope in which to run the function.

function A reference to a function.

Description

Method (static); allows you to delegate events to specific scopes and functions. Use the following syntax:

```
import mx.utils.Delegate;  
compInstance.addEventListener("eventName", Delegate.create(scopeObject,  
    function));
```

The *scopeObject* parameter specifies the scope in which the specified function is called.

Example

For examples of `Delegate.create()`, see “Delegating events” in *Using Components*.

See also

[EventDispatcher.addEventListener\(\)](#)

DeltaItem class (Flash Professional only)

ActionScript Class Name mx.data.components.datasetclasses.DeltaItem

The `DeltaItem` class provides information about an individual operation performed on a transfer object. It indicates whether a change was made directly to a property of the transfer object or whether the change was made by a method call. It also provides the original state of properties on a transfer object. For example, if the source of the delta packet was a data set, the `DeltaItem` object contains information about any field that was edited.

In addition to the above, a `DeltaItem` object can contain server response information such as current value and a message.

Use the `DeltaItem` class when accessing the changes in a delta packet. To access these changes, use `DeltaPacket.getIterator()`, which returns an iterator of deltas. Each delta contains zero or more `DeltaItem` instances, which you can access through `Delta.getItemByName()` or `Delta.getChangeList()`.

Property summary for the DeltaItem class

The following table lists the properties of the `DeltaItem` class.

Property	Description
<code>DeltaItem.argList</code>	If a change is made through a method call, this is the array of values that were passed to the method. This property is read-only.
<code>DeltaItem.curValue</code>	If a change is made to a property, this is the current server value of the property. This property is read-only.
<code>DeltaItem.delta</code>	The associated delta for the <code>DeltaItem</code> object. This property is read-only.
<code>DeltaItem.kind</code>	The type of change.
<code>DeltaItem.message</code>	The server message associated with the <code>DeltaItem</code> object.

Property	Description
<code>DeltaItem.name</code>	The name of the property or method that changed. This property is read-only.
<code>DeltaItem.newValue</code>	If a change was made to a property, this is the new value of the property. This property is read-only.
<code>DeltaItem.oldValue</code>	If a change was made to a property, this is the old value of the property. This property is read-only.

DeltaItem.argList

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`deltaitem.argList`

Description

Property (read-only); an array of values passed to the change method. This property applies only if the change's kind is `DeltaItem.Method`.

DeltaItem.curValue

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`deltaitem.curValue`

Description

Property (read-only); an object containing the current property value on the server's copy of the transfer object. This property applies only if the change's kind is `DeltaItem.Property`, and the property is relevant only in a delta that has been returned from a server and is being applied to the data set for user resolution.

DeltaItem.delta

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaitem.delta

Description

Property (read-only); a delta associated with the `DeltaItem` object. When a `DeltaItem` object is created, it is associated with a delta and adds itself to the delta's list of changes. This property provides a reference to the delta that this item belongs to.

DeltaItem.kind

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaitem.kind

Description

Property; a number that indicates the type of change. Use the following constants to evaluate this property:

- `DeltaItem.Property` The change was made to a property on the transfer object.
- `DeltaItem.Method` The change was made through a method call on the transfer object.

DeltaItem.message

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaItem.message

Description

Property; a string containing a server message associated with this DeltaItem object. This can be any message for the property or method call change attempted in the delta packet. This message is usually relevant only in a delta that has been returned from a server and is being applied to the DataSet for resolution.

DeltaItem.name

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaItem.name

Description

Property (read-only); a string containing the name of the changed property (if the change's kind is `DeltaItem.Property`) or the name of the method that made the change (if the change's kind is `DeltaItem.Method`).

DeltaItem.newValue

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaItem.newValue

Description

Property (read-only); an object containing the new value of the property. This property applies only if the change's kind is `DeltaItem.Property`.

DeltaItem.oldValue

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

deltaItem.oldValue

Description

Property (read-only); an object containing the old value of the property. This property applies only if the change's kind is `DeltaItem.Property`.

Delta interface (Flash Professional only)

ActionScript Interface Name mx.data.components.datasetclasses.Delta

The Delta interface provides access to the transfer object, collection, and transfer object-level changes. With this interface you can access the new and previous values in a transfer object. For example, if the delta packet was obtained from a data set, each delta would represent an added, edited, or deleted row.

The Delta interface also provides access to messages returned by the associated server-side process. For more information on client-server interactions, see [“RDBMSResolver component \(Flash Professional only\)” on page 1047](#).

Use the Delta interface to examine the delta packet before sending changes to the server and to review messages returned from server-side processing.

Method summary for the Delta interface

The following table lists the methods of the Delta interface.

Method	Description
<code>Delta.addDeltaItem()</code>	Adds the specified <code>DeltaItem</code> instance.
<code>Delta.getChangeList()</code>	Returns an array of changes made to the current item.
<code>Delta.getDeltaPacket()</code>	Returns the delta packet that contains the delta.
<code>Delta.getId()</code>	Returns the unique ID of the current item within the <code>DeltaPacket</code> collection.
<code>Delta.getItemByName()</code>	Returns the specified <code>DeltaItem</code> object.
<code>Delta.getMessage()</code>	Returns the message associated with the current item.
<code>Delta.getOperation()</code>	Returns the operation that was performed on the current item within the original collection.
<code>Delta.getSource()</code>	Returns the transfer object on which the changes were performed.

Delta.addDeltaItem()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.addDeltaItem(deltaitem)
```

Parameters

deltaitem DeltaItem instance to add to this delta.

Returns

Nothing.

Description

Method; adds the specified DeltaItem instance. If the specified DeltaItem instance already exists, this method replaces it.

Example

The following example calls the addDeltaItem() method:

```
//...  
var d:Delta = new DeltaImpl("ID1345678", curItem, DeltaPacketConsts.Added,  
    "", false);  
d.addDeltaItem(new DeltaItem(DeltaItem.Property, "ID", {oldValue:15,  
    newValue:16}));  
//...
```

Delta.getChangeList()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getChangeList()
```


Parameters

None.

Returns

An array of associated `DeltaItem` instances.

Description

Method; returns an array of associated `DeltaItem` instances. Each `DeltaItem` instance in the array describes a change made to the item.

Example

The following example calls the `getChangeList()` method.:

```
//...
case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
    // dpDelta is a variable of type Delta.
    var changes:Array = dpDelta.getChangeList();
    for(var i:Number = 0; i<changes.length; i++) {
        // getChangeMessage is a user-defined method.
        changeMsg = _parent.getChangeMessage(changes[i]);
        trace(changeMsg);
    }
//...
```

Delta.getDeltaPacket()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getDeltaPacket()
```

Parameters

None.

Returns

The delta packet that contains this delta.

Description

Method; returns the delta packet that contains this delta. This method lets you write code that can handle delta packets generically at the delta level.

Example

The following example uses the `getDeltaPacket()` method to access the delta packet's data source:

```
while(dpCursor.hasNext()) {
    dpDelta = Delta(dpCursor.next());
    trace("DeltaPacket source is: " + dpDelta.getDeltaPacket().getSource());
}
```

Delta.getId()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getId()
```

Parameters

None.

Returns

An object; returns the unique ID of this item within the `DeltaPacket` collection.

Description

Method; returns a unique identifier for this item within the `DeltaPacket` collection. Use this ID in the source component for the delta packet to receive updates and make changes to items that the delta packet was generated from. For example, assuming that the `DataSet` component sends updates to a server and the server returns new key field values, this method allows the `DataSet` component to examine the resulting delta packet, find the original transfer object, and make the appropriate updates to it.

Example

The following example calls the `getId()` method:

```
while(dpCursor.hasNext()) {
    dpDelta = Delta(dpCursor.next());
    trace("id ["+dpDelta.getId()+"]");
}
```

Delta.getItemByName()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getItemByName(name)
```

Parameters

name A string that specifies the name of the property or method for the associated `DeltaItem` object.

Returns

The `DeltaItem` object specified by *name*. If no `DeltaItem` object is found that matches *name*, this method returns `null`.

Description

Method; returns the `DeltaItem` object specified by *name*. When method calls or property changes on a transfer object are needed by name, this method provides the most efficient access.

Example

The following example calls the `getItemByName()` method:

```
private function buildFieldTag(deltaObj:Delta, field:Object,
    isKey:Boolean):String {
    var chgItem:DeltaItem = deltaObj.getItemByName(field.name);
    var result:String= "<field name=\"" + field.name + "\" type=\"" +
        field.type.name + "\"";
    var oldValue:String;
    var newValue:String;
    if (deltaObj.getOperation() != DeltaPacketConsts.Added) {
        oldValue = (chgItem != null ? (chgItem.oldValue != null ?
            encodeFieldValue(field.name, chgItem.oldValue) : __nullValue) :
            encodeFieldValue(field.name, deltaObj.getSource()[field.name]));
        newValue = (chgItem.newValue != null ? encodeFieldValue(field.name,
            chgItem.newValue) : __nullValue);
        result+= " oldValue=\"" + oldValue + "\"";
        result+= chgItem != null ? " newValue=\"" + newValue + "\" : ";
        result+= " key=\"" + isKey.toString() + "\" />";
    }
    else {
        result+= " newValue=\"" +encodeFieldValue(field.name,
            deltaObj.getSource()[field.name]) + "\"";
        result+= " key=\"" + isKey.toString() + "\" />";
    }
    return result;
}
```

Delta.getMessage()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getMessage()
```

Parameters

None.

Returns

A string; returns the message associated with *delta*.

Description

Method; returns the associated message for this delta. Typically this message is only populated if the delta packet has been returned from a server in response to attempted updates. For more information, see [“RDBMSResolver component \(Flash Professional only\)”](#) on page 1047.

Example

The following example calls the `getMessage()` method:

```
//...
var dpi:Iterator = dp.getIterator();
var d:Delta;
while(dpi.hasNext()) {
    d= dpi.next();
    trace(d.getMessage());
}
//...
```

Delta.getOperation()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getOperation()
```

Parameters

None.

Returns

A number; returns the operation that was performed on the item within the original collection.

Description

Method; returns the operation that was performed on this item within the original collection. Valid values for this are `DeltaPacketConsts.Added`, `DeltaPacketConsts.Removed`, and `DeltaPacketConsts.Modified`.

You must either import `mx.data.components.datasetclasses.DeltaPacketConsts` or fully qualify each constant.

Example

The following example calls the `getOperation()` method:

```
while(dpCursor.hasNext()) {
    dpDelta = Delta(dpCursor.next());
    op=dpDelta.getOperation();
    trace("DeltaPacket source is: " + dpDelta.getDeltaPacket().getSource());
    switch(op) {
        case mx.data.components.datasetclasses.DeltaPacketConsts.Added:
            trace("***In case DeltaPacketConsts.Added ***");
        case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
            trace("***In case DeltaPacketConsts.Modified ***");
        }
    }
}
```

Delta.getSource()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
delta.getSource()
```

Parameters

None.

Returns

The transfer object on which the changes were performed.

Description

Method; returns the transfer object on which the changes were performed.

Example

The following example calls the `getSource()` method:

```
while(dpCursor.hasNext()) {
    dpDelta = Delta(dpCursor.next());
    op=dpDelta.getOperation();
    switch(op) {
        case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
            // the original values are
            trace("Unmodified source is: ");
            var src = dpDelta.getDeltaPacket().getSource();
            for(var i in src){
                if(typeof(src[i]) != "function"){
                    trace(i+"="+src[i]);
                }
            }
        }
    }
}
```


DeltaPacket interface (Flash Professional only)

ActionScript Interface Name mx.data.components.datasetclasses.DeltaPacket

The DeltaPacket interface is provided by the `deltaPacket` property of the DataSet component, which is part of the data management functionality in Flash MX Professional 2004. (For more information, see Chapter 16, “Data Integration (Flash Professional Only),” in *Using Flash*). Typically the delta packet is used internally by resolver components. The DeltaPacket interface and the related Delta interface and DeltaItem class let you manage changes made to the data. These components have no visual appearance at runtime.

A delta packet is an optimized set of instructions that describe all changes that have been made to the data in a data set. When the `DataSet.applyUpdates()` method is called, the DataSet component populates the `DataSet.deltaPacket` property. Typically, this property is connected (by data binding) to a resolver component such as RDBMSResolver. The resolver converts the delta packet into an update packet in the appropriate format.

NOTE

Unless you are writing your own custom resolver, it is unlikely you will ever need to know about or write code that accesses methods or properties of a delta packet.

A delta packet contains one or more deltas (see “[Delta interface \(Flash Professional only\)](#)” on page 469), and each delta contains zero or more delta items (see “[DeltaItem class \(Flash Professional only\)](#)” on page 463).

Method summary for the DeltaPacket interface

The following table lists the methods of the DeltaPacket interface.

Method	Description
<code>DeltaPacket.getConfigInfo()</code>	Returns configuration information that is specific to the implementation of the DeltaPacket interface.
<code>DeltaPacket.getIterator()</code>	Returns the iterator for the delta packet that iterates through the delta packet's list of deltas.
<code>DeltaPacket.getSource()</code>	Returns the source of the delta packet. This is the component that has exposed this delta packet.
<code>DeltaPacket.getTimestamp()</code>	Returns the date and time at which the delta packet was instantiated.
<code>DeltaPacket.getTransactionId()</code>	Returns the transaction ID for this delta packet.
<code>DeltaPacket.logChanges()</code>	Indicates whether the consumer of the delta packet should log the changes it specifies.

DeltaPacket.getConfigInfo()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.getConfigInfo(info)
```

Properties

info Object; contains information specific to the implementation.

Returns

An object that contains information required for the specific DeltaPacket implementation.

Description

Method; returns configuration information that is specific to the implementation of the `DeltaPacket` interface. This method allows implementations of the `DeltaPacket` interface to access custom information.

Example

The following example calls the `getConfigInfo()` method:

```
// ...
new DeltaPacketImpl(source, getTransactionId(), null, logChanges(),
    getConfigInfo());
// ...
```

DeltaPacket.getIterator()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.getIterator()
```

Parameters

None.

Returns

An interface to the iterator for the `DeltaPacket` collection that iterates through the delta packet's list of deltas.

Description

Method; returns the iterator for the `DeltaPacket` collection. This provides a mechanism for looping through the changes in the delta packet. For a complete description of this interface, see [“Iterator interface \(Flash Professional only\)” on page 749](#).

Example

The following example uses the `getIterator()` method to access the iterator for the deltas in a delta packet and uses a `while` statement to loop through the deltas:

```
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
trace("*** Test deltapacket. Trans ID is: " + deltapkt.getTransactionId() +
      " ***");
var OPS:Array = new Array("added", "removed", "modified");
var dpCursor:Iterator = deltapkt.getIterator();
var dpDelta:Delta;
var op:Number;
var changeMsg:String;
while(dpCursor.hasNext()) {
    dpDelta = Delta(dpCursor.next());
    op=dpDelta.getOperation();
}
```

DeltaPacket.getSource()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.getSource()
```

Parameters

None.

Returns

An object; the source of the DeltaPacket collection. This object is typically a descendant of MovieClip, but this is not required. For example, if the source is a data set, this object might be `_level0.myDataSet`.

Description

Method; returns the source of the DeltaPacket collection.

Example

The following example calls the `getSource()` method:

```
// ...
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
var dpSourceText:String = "Source: " + deltapkt.getSource();
trace(dpSourceText);
// ...
```

DeltaPacket.getTimestamp()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.getTimestamp()
```

Parameters

None.

Returns

A Date object containing the date and time at which the delta packet was created.

Description

Method; returns the date and time at which the delta packet was created.

Example

The following example calls the `getTimestamp()` method:

```
// ...
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
var dpTime:String = " Time: " + deltapkt.getTimestamp();
trace(dpTime);
// ...
```

DeltaPacket.getTransactionId()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.getTransactionId()
```

Parameters

None.

Returns

A string; the unique transaction ID for a single transaction grouping of delta packets.

Description

Method; returns the transaction ID for the delta packet. This unique identifier is used to group a send/receive transaction for a delta packet. The data set uses this to determine if the delta packet is part of the same transaction it originated with the [DataSet.applyUpdates\(\)](#) call.

Example

The following example calls the `getTransactionId()` method:

```
// ...  
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;  
trace("*** Trans ID is: " + deltapkt.getTransactionId() + " ***");  
// ...
```

DeltaPacket.logChanges()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
deltaPacket.logChanges()
```

Parameters

None.

Returns

A Boolean value; `true` if the consumer of the delta packet should log changes found in the delta packet.

Description

Method; returns `true` if the consumer of this delta packet should log the changes it specifies. This value is used mainly for communication of changes between data sets by means of shared objects or from a server to a local data set. In both cases, the data set should not record the changes specified.

Example

The following example calls the `logChanges()` method:

```
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
if(deltapkt.logChanges()) {
    trace("*** We need to log changes. ***");
}
else {
    trace("*** We do not need to log changes");
}
```


ActionScript Class Name `mx.managers.DepthManager`

The `DepthManager` class allows you to manage the relative depth assignments of any component or movie clip, including `_root`. It also lets you manage reserved depths in a special highest-depth clip on `_root` for system-level services such as the pointer and tooltips.

In general, `DepthManager` manages components automatically, using its own “shuffling” algorithm. You do not need to use its APIs unless you are an advanced Flash developer.

NOTE

To use the `DepthManager` class for movie clip instances, you need to have a component in the library or on the Stage, and use “import `mx.managers.DepthManager`” at the beginning of your ActionScript.

The following methods constitute the relative depth-ordering API:

- `DepthManager.createChildAtDepth()`
- `DepthManager.createClassChildAtDepth()`
- `DepthManager.setDepthAbove()`
- `DepthManager.setDepthBelow()`
- `DepthManager.setDepthTo()`

The following methods constitute the reserved depth space API:

- `DepthManager.createClassObjectAtDepth()`
- `DepthManager.createObjectAtDepth()`

Method summary for the DepthManager class

The following table lists the methods of the DepthManager class.

Method	Description
<code>DepthManager.createChildAtDepth()</code>	Creates a child of the specified symbol at the specified depth.
<code>DepthManager.createClassChildAtDepth()</code>	Creates an object of the specified class at the specified depth.
<code>DepthManager.createClassObjectAtDepth()</code>	Creates an instance of the specified class at a specified depth in the special highest-depth clip.
<code>DepthManager.createObjectAtDepth()</code>	Creates an object at a specified depth in the highest-depth clip.
<code>DepthManager.setDepthAbove()</code>	Sets the depth above the specified instance.
<code>DepthManager.setDepthBelow()</code>	Sets the depth below the specified instance.
<code>DepthManager.setDepthTo()</code>	Sets the depth to the specified instance in the highest-depth clip.

Property summary for the DepthManager class

The following table lists the properties of the DepthManager class. The constant values shown are the default values that the DepthManager algorithm uses to arrange depth. If you trace the following properties, you will see those constant values in the Output panel.

However, after you implement a DepthManager method, such as `DepthManager.setDepthTo()`, using one of the following properties, and then trace the component or movie clip depth, you see that DepthManager sets the depths in increments of 20. The algorithm increments depths in case Flash needs to insert something else in the middle, based on other scripts, components, and so on.

Property	Description
<code>DepthManager.kBottom</code>	A static property with the constant value 202.
<code>DepthManager.kCursor</code>	A static property with the constant value 101. This is the cursor depth.
<code>DepthManager.kNotopmost</code>	A static property with the constant value 204.
<code>DepthManager.kTooltip</code>	A static property with the constant value 102. This is the tooltip depth.

Property	Description
DepthManager.kTop	A static property with the constant value 201.
DepthManager.kTopmost	A static property with the constant value 203.

DepthManager.createChildAtDepth()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
movieClipInstance.createChildAtDepth(linkageName, depthFlag[, initObj])
```

Parameters

linkageName A linkage identifier. This parameter is a string.

depthFlag One of the following values: [DepthManager.kTop](#), [DepthManager.kBottom](#), [DepthManager.kTopmost](#), [DepthManager.kNotopmost](#). All depth flags are static properties of the DepthManger class. You must either reference the DepthManager package (for example, `mx.managers.DepthManager.kTopmost`), or use the `import` statement to import the DepthManager package.

initObj An initialization object. This parameter is optional.

Returns

A reference to the object created. The return type is `MovieClip`.

Description

Method; creates a child instance of the symbol specified by *linkageName* at the depth specified by *depthFlag*.

Example

The following example creates a `minuteHand` instance of the `MinuteSymbol` movie clip and places it in front of the clock:

```
import mx.managers.DepthManager;
minuteHand = clock.createChildAtDepth("MinuteSymbol", DepthManager.kTop);
```

DepthManager.createClassChildAtDepth()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
movieClipInstance.createClassChildAtDepth(className, depthFlag[, initObj])
```

Parameters

className A class name. This parameter is a of type Function.

depthFlag One of the following values: `DepthManager.kTop`, `DepthManager.kBottom`, `DepthManager.kTopmost`, `DepthManager.kNotopmost`. All depth flags are static properties of the DepthManger class. You must either reference the DepthManager package (for example, `mx.managers.DepthManager.kTopmost`), or use the `import` statement to import the DepthManager package.

initObj An initialization object. This parameter is optional.

Returns

A reference to the created child. The return type is UIObject.

Description

Method; creates a child of the class specified by *className* at the depth specified by *depthFlag*.

Example

The following code draws a focus rectangle in front of all NoTopmost objects:

```
import mx.managers.DepthManager
this.ring = createClassChildAtDepth(mx.skins.RectBorder,
    DepthManager.kTop);
```

The following code creates an instance of the Button class and passes it a value for its `label` property as an *initObj* parameter:

```
import mx.managers.DepthManager
button1 = createClassChildAtDepth(mx.controls.Button, DepthManager.kTop,
    {label: "Top Button"});
```

DepthManager.createClassObjectAtDepth()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
DepthManager.createClassObjectAtDepth(className, depthSpace[, initObj])
```

Parameters

className A class name. This parameter is of type Function.

depthSpace One of the following values: [DepthManager.kCursor](#), [DepthManager.kTooltip](#). All depth flags are static properties of the DepthManger class. You must either reference the DepthManager package (for example, `mx.managers.DepthManager.kCursor`), or use the import statement to import the DepthManager package.

initObj An initialization object. This parameter is optional.

Returns

A reference to the created object. The return type is UIObject.

Description

Method; creates an object of the class specified by *className* at the depth specified by *depthSpace*. This method is used for accessing the reserved depth spaces in the special highest-depth clip.

Example

The following example creates an object from the Button class:

```
import mx.managers.DepthManager
myCursorButton = Detph.createClassObjectAtDepth(mx.controls.Button,
    DepthManager.kCursor, {label: "Cursor"});
```

DepthManager.createObjectAtDepth()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
DepthManager.createObjectAtDepth(linkageName, depthSpace[, initObj])
```

Parameters

linkageName A linkage identifier. This parameter is of type String.

depthSpace One of the following values: [DepthManager.kCursor](#), [DepthManager.kTooltip](#). All depth flags are static properties of the DepthManger class. You must either reference the DepthManager package (for example, `mx.managers.DepthManager.kCursor`), or use the `import` statement to import the DepthManager package.

initObj An optional initialization object.

Returns

A reference to the created object. The return type is MovieClip.

Description

Method; creates an object at the specified depth. This method is used for accessing the reserved depth spaces in the special highest-depth clip.

Example

The following example creates an instance of the `TooltipSymbol` symbol and places it at the reserved depth for tooltips:

```
import mx.managers.DepthManager
myCursorTooltip = DepthManager.createObjectAtDepth("TooltipSymbol",
    DepthManager.kTooltip);
```

DepthManager.kBottom

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kBottom`

Description

Property (static); a property with the constant value 202. This property is passed as a parameter in calls to [DepthManager.createClassChildAtDepth\(\)](#) and [DepthManager.createChildAtDepth\(\)](#) to place content behind other content.

DepthManager.kCursor

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kCursor`

Description

Property (static); a property with the constant value 101. This property is passed as a parameter in calls to [DepthManager.createClassObjectAtDepth\(\)](#) and [DepthManager.createObjectAtDepth\(\)](#) to request placement at cursor depth.

DepthManager.kNotopmost

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kNotopmost`

Description

Property (static); a property with the constant value 204. This property is passed as a parameter in calls to `DepthManager.createClassChildAtDepth()` and `DepthManager.createChildAtDepth()` to request removal from the topmost layer.

DepthManager.kTooltip

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kTooltip`

Description

Property (static); a property with the constant value 102. This property is passed as a parameter in calls to `DepthManager.createClassObjectAtDepth()` and `DepthManager.createObjectAtDepth()` to place an object at the tooltip depth.

DepthManager.kTop

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kTop`

Description

Property (static); a property with the constant value 201. This property is passed as a parameter in calls to `DepthManager.createClassChildAtDepth()` and `DepthManager.createChildAtDepth()` to request placement on top of other content but below `DepthManager.kTopmost` content.

DepthManager.kTopmost

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`DepthManager.kTopmost`

Description

Property (static); a property with the constant value 203. This property is used in calls to [DepthManager.createClassChildAtDepth\(\)](#) and [DepthManager.createChildAtDepth\(\)](#) to request placement on top of other content, including [DepthManager.kTop](#) objects.

DepthManager.setDepthAbove()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`movieClipInstance.setDepthAbove(instance)`

Parameters

instance An instance name. This parameter is of type `MovieClip`.

Returns

Nothing.

Description

Method; sets the depth of a movie clip or component instance above the depth of the instance specified by the *instance* parameter and moves other objects if necessary.

DepthManager.setDepthBelow()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

movieClipInstance.setDepthBelow(instance)

Parameters

instance An instance name. This parameter is of type MovieClip.

Returns

Nothing.

Description

Method; sets the depth of a movie clip or component instance below the depth of the specified instance and moves other objects if necessary.

Example

The following code sets the depth of the `textInput` instance below the depth of `button`:

```
textInput.setDepthBelow(button);
```

DepthManager.setDepthTo()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

movieClipInstance.setDepthTo(depthFlag)

Parameters

depthFlag One of the following values: `DepthManager.kTop`, `DepthManager.kBottom`, `DepthManager.kTopmost`, `DepthManager.kNotopmost`. All depth flags are static properties of the `DepthManger` class. You must either reference the `DepthManager` package (for example, `mx.managers.DepthManager.kTopmost`) or use the `import` statement to import the `DepthManager` package.

Returns

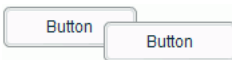
Nothing.

Description

Method; sets the depth of *movieClipInstance* to the value specified by *depthFlag*. This method moves an instance to another depth to make room for another object. `DepthManager` uses a “shuffling” algorithm to set the depths in increments of 20. The algorithm increments depths in case Flash needs to insert something else in the middle, based on other scripts, components, and so on.

Example

The following example uses two components (or movie clips) to raise their depth alternately in increments of 20 as each one is clicked. First add a `Button` component to the Stage and give it instance name `a_btn`. Then add another `Button` component to the Stage and give it instance name `b_btn`. Make sure the buttons overlap as follows:



```
import mx.managers.DepthManager;

a_btn.onRelease = function() {
    b_btn.setDepthTo(DepthManager.kTop);
    var b_depth:Number = b_btn.getDepth();
    trace(b_depth);
}

b_btn.onRelease = function() {
    a_btn.setDepthTo(DepthManager.kTop);
    var a_depth:Number = a_btn.getDepth();
    trace(a_depth);
}
```

Test the SWF file. When you click the top button, the other button changes depth and moves to the front, and the Output panel displays that button's depth. The values are 20, then 40, then 60, incremented by 20 each time you click.

NOTE

If you use `DepthManager` with movie clip instances instead of component instances, you may need to add a UI component to your library (if one isn't already there) for `DepthManager` to operate properly. `DepthManager` requires a component on the Stage or in the library to function.

For more information about depth and stacking order, see “Determining the next highest available depth” in *Learning ActionScript 2.0 in Flash*.

Events let your application know when the user has interacted with a component, and when important changes have occurred in the appearance or life cycle of a component—such as its creation, destruction, or resizing.

The methods of the `EventDispatcher` class let you add and remove event listeners so that your code can react to events appropriately. For example, you use the `EventDispatcher.addEventListener()` method to register a listener with a component instance. The listener is invoked when a component's event is triggered.

If you want to write a custom object that emits events that aren't related to the user interface, `EventDispatcher` is smaller and faster to use as a mix-in for `UIComponent` than `UIEventDispatcher`.

Event objects

An event object is passed to a listener as a parameter. The event object is an `ActionScript` object that has properties that contain information about the event that occurred. You can use the event object inside the listener callback function to access the name of the event that was broadcast, or the instance name of the component that broadcast the event. For example, the following code uses the `target` property of the `evtObj` event object to access the `label` property of the `myButton` instance and send the value to the Output panel:

```
listener = new Object();
listener.click = function(evtObj){
    trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);
```

Some event object properties are defined in the W3C specification (www.w3.org/TR/DOM-Level-3-Events/events.html) but aren't implemented in version 2 of the Macromedia Component Architecture. Every version 2 event object has the properties listed in the table below. Some events have additional properties defined, and if so, the properties are listed in the event's entry.

Property	Description
type	A string indicating the name of the event.
target	A reference to the component instance broadcasting the event.

EventDispatcher class (API)

ActionScript Class Name `mx.events.EventDispatcher`

Method summary for the EventDispatcher class

The following table lists the methods of the EventDispatcher class.

Method	Description
<code>EventDispatcher.addEventListener()</code>	Registers a listener with a component instance.
<code>EventDispatcher.dispatchEvent()</code>	Dispatches an event programmatically.
<code>EventDispatcher.removeEventListener()</code>	Removes an event listener from a component instance.

EventDispatcher.addEventListener()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
componentInstance.addEventListener(event, listener)
```

Parameters

event A string that is the name of the event.

listener A reference to a listener object or function.

Returns

Nothing.

Description

Method; registers a listener object with a component instance that is broadcasting an event. When the event occurs, the listener object or function is notified. You can call this method from any component instance. For example, the following code registers a listener to the component instance `myButton`:

```
myButton.addEventListener("click", myListener);
```

You must define the listener as either an object or a function before you call `addEventListener()` to register the listener with the component instance. If the listener is an object, it must have a callback function defined that is invoked when the event occurs. Usually, that callback function has the same name as the event with which the listener is registered. If the listener is a function, the function is invoked when the event occurs. For more information, see “Using listeners to handle events” in *Using Components*.

You can register multiple listeners to a single component instance, but you must use a separate call to `addEventListener()` for each listener. Also, you can register one listener to multiple component instances, but you must use a separate call to `addEventListener()` for each instance. For example, the following code defines one listener object and assigns it to two `Button` component instances, whose `label` properties are `button1` and `button2`, respectively:

```
lo = new Object();
lo.click = function(evt){
    trace(evt.target.label + " clicked");
}
button1.addEventListener("click", lo);
button2.addEventListener("click", lo);
```

Execution order is not guaranteed. You cannot expect one listener to be called before another.

An event object is passed to the listener as a parameter. The event object has properties that contain information about the event that occurred. You can use the event object inside the listener callback function to access information about the type of event that occurred and which instance broadcast the event. In the example above, the event object is `evt` (you can use any identifier as the event object name), and it is used in the `if` statements to determine which button instance was clicked. For more information, see “About the event object” in *Using Components*.

Example

The following example defines a listener object, `myListener`, and defines the callback function for the `click` event. It then calls `addEventListener()` to register the `myListener` listener object with the component instance `myButton`.

```
myListener = new Object();
myListener.click = function(evt){
    trace(evt.type + " triggered");
}
myButton.addEventListener("click", myListener);
```

To test this code, place a `Button` component on the Stage with the instance name `myButton`, and place this code in Frame 1.

EventDispatcher.dispatchEvent()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
dispatchEvent(eventObject)
```

Parameters

eventObject A reference to an event object. The event object must have a `type` property that is a string indicating the name of the event. Generally, the event object also has a `target` property that is the name of the instance broadcasting the event. You can define other properties on the event object that help a user capture information about the event when it is dispatched.

Returns

Nothing.

Description

Method; dispatches an event to any listener registered with an instance of the class. This method is usually called from within a component's class file. For example, the `SimpleButton.as` class file dispatches the `click` event.

Example

The following example dispatches a `click` event:

```
dispatchEvent({type:"click"});
```

EventDispatcher.removeEventListener()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
componentInstance.removeEventListener(event, listener)
```

Parameters

event A string that is the name of the event.

listener A reference to a listener object or function.

Returns

Nothing.

Description

Method; unregisters a listener object from a component instance that is broadcasting an event.

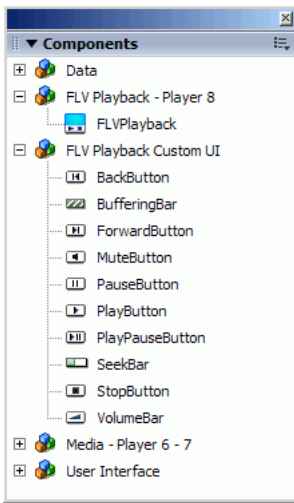
FLVPlayback Component (Flash Professional Only)

The FLVPlayback component lets you easily include a video player in your Flash application to play progressively downloaded Flash video (FLV) files over HTTP or play streaming FLV files from a Flash Communication Server (FCS) or from a Flash Video Streaming Service (FVSS).

The easy-to-use FLVPlayback component has the following characteristics and benefits:

- Can be dragged to the Stage and implemented quickly and successfully
- Provides a collection of predesigned *skins* that allow you to customize the appearance of its playback controls
- Allows advanced users to create their own skins
- Provides cue points that allow you to synchronize your video with text, graphics, and animation
- Provides live preview of customizations
- Maintains a reasonably sized SWF file

The FLVPlayback component includes the FLV Playback Custom UI components. The FLVPlayback component is a combination of the display area, or video player, in which you view the FLV file and the controls that allow you to operate it. The FLV Playback Custom UI components provide control buttons and mechanisms that you can use to play, stop, pause, and otherwise control the FLV file. These controls include the BackButton, BufferingBar, ForwardButton, MuteButton, PauseButton, PlayButton, PlayPauseButton, SeekBar, StopButton, and VolumeBar. The FLVPlayback component and the FLV Playback Custom UI controls appear in the Components panel, as shown in the following figure:



The process of adding playback controls to the FLVPlayback component is called *skinning*. The FLVPlayback component has an initial default skin, `ClearOverPlaySeekMute.swf`, that provides transparent controls for the play, seek, and mute functions. To change this skin, you have the following choices:

- Select from an collection of predesigned skins
- Select individual controls from the FLV Playback Custom UI components and customize them
- Create a custom skin and add it to the collection of predesigned skins

After you select a different skin, the selected skin becomes the new default skin.

For more information about selecting or creating a skin for the FLVPlayback component, see [“Customizing the FLVPlayback component” on page 524](#).

The FLVPlayback component also includes an ActionScript application programming interface (API). The API includes the FLVPlayback, VideoError, and VideoPlayer classes. For more information on these classes, see [“FLVPlayback class” on page 539](#), the [“VideoPlayer class” on page 706](#), and the [“VideoError class” on page 698](#).

Using the FLVPlayback component

Using the FLVPlayback component basically consists of putting it on the Stage and specifying an FLV file for it to play. In addition, however, you can also set various parameters that govern its behavior and describe the FLV file.

Creating an application with the FLVPlayback component

You can include the FLVPlayback component in your application in the following ways:

- Drag the FLVPlayback component from the Components panel to the Stage, and specify a value for the `contentPath` parameter.
- Use the Video Import wizard to create the component on the Stage, and customize it by selecting a skin.
- Use the MovieClip `attachMovie()` method to dynamically create an FLVPlayback instance on the Stage, assuming the component is in the library.

To drag the FLVPlayback component from the Components panel:

1. In the Components panel, click the Plus (+) button to open the FLV Playback - Player 8 entry.
2. Drag the FLVPlayback component to the Stage.
3. With the FLVPlayback component selected on the Stage, locate the Value cell for the `contentPath` parameter in the Parameters tab of the Component inspector, and enter a string that specifies one of the following:
 - A local path to an FLV file
 - A URL to an FLV file
 - A URL to an XML file that describes how to play an FLV file

For information on how to create an XML file to describe one or more FLV files, see [“Using a SMIL file” on page 712](#).

4. On the Parameters tab in the Component inspector, with the FLVPlayback component selected on the Stage, click the Value cell for the `skin` parameter.

5. Click the magnifying-glass icon to open the Select Skin dialog box.
6. Select one of the following options:
 - From the drop-down Skin list, select one of the predesigned skins to attach a set of playback controls to the component.
 - If you created a custom skin, select Custom Skin URL from the pop-up menu, and enter, in the URL text box, the URL for the SWF file that contains the skin.
 - Select None, and drag individual FLV Playback Custom UI components to the Stage to add playback controls.

NOTE

In the first two cases, a preview of the skin appears in the viewing pane above the pop-up menu.

7. Click OK to close the Select Skin dialog box.
8. Select Test Movie from the Control menu to execute the SWF file and start the video.

To use the Video Import wizard:

1. Select File > Import > Import Video.
2. Indicate the location of the video file by selecting one of the following options:
 - On my local computer
 - Already deployed to a web, FCS, or FVSS server
3. Depending on your choice, enter either the path or the URL that specifies the location of the video file; then click Next.
4. If you selected a file path, you'll see a Deployment dialog box next where you can select one of the options listed to specify how you would like to deploy your video:
 - Progressive download from a standard web server
 - Stream from Flash Video Streaming Service
 - Stream from Flash Communication Server
 - Embed video in SWF and play in timeline

WARNING

Do not select the Embed Video option. The FLVPlayback component plays only external streaming video. This option will not place an FLVPlayback component on the Stage.

5. Click Next.

6. Select one of the following options:

- From the drop-down Skin list, select one of the predesigned skins to attach a set of playback controls to the component.
- If you created a custom skin for the component, select Custom Skin URL from the pop-up menu, and enter the URL for the SWF file that contains the skin in the URL text box.
- Select None, and drag individual FLV Playback Custom UI components to the Stage to add playback controls.

NOTE

In the first two cases, a preview of the skin appears in the viewing pane above the pop-up menu.

7. Click OK to close the Select Skin dialog box.

8. Read the Finish Video Import dialog box to note what happens next, and then click Finish.

9. If you have not saved your FLA file, a Save As dialog box appears.

10. Select Test Movie from the Control menu to execute the SWF file, and start the video.

To create an instance dynamically using ActionScript:

1. Drag the FLVPlayback component from the Components panel to the Library (Window > Library).

2. Add the following code to the Actions panel on Frame 1 of the Timeline. Change *install_drive* to the drive on which you installed Flash 8 and modify the path to reflect the location of the Skins folder for your installation:

```
import mx.video.*;
this.attachMovie("FLVPlayback", "my_FLVPlaybk", 10, {width:320,
    height:240, x:100, y:100});
my_FLVPlaybk.skin = "file:///install_drive|/Program Files/Macromedia/
    Flash 8/en/Configuration/Skins/ClearOverPlaySeekMute.swf"
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
```

The `attachMovie()` method belongs to the `MovieClip` class. You can use it create an instance of the FLVPlayback component because the FLVPlayback class extends the `MovieClip` class.

NOTE

Without setting the `contentPath` and `skin` properties, the generated movie clip will appear to be empty.

3. Select Test Movie from the Control menu to execute the SWF file and start the FLV file.

FLVPlayback component parameters

For each instance of the FLVPlayback component, you can set the following parameters in the Component inspector or the Property inspector:

autoPlay A Boolean value that determines how to play the FLV file. If `true`, the component plays the FLV file immediately when it is loaded. If `false`, the component loads the first frame and pauses. The default value is `true` for the default video player (0) and `false` for others. For more information about using multiple video players in a single FLVPlayback instance, see [“Playing multiple FLV files” on page 521](#).

autoRewind A Boolean value that determines whether the FLV file will rewind automatically when it finishes playing. If `true`, the FLVPlayback component automatically rewinds the FLV file to the beginning when the playhead reaches the end or when the user clicks the Stop button. If `false`, the component stops play on the last frame of the FLV file and does not rewind automatically. The default value is `true`.

autoSize A Boolean value that, if `true`, resizes the component at runtime to use the source FLV file dimensions. These dimensions are encoded in the FLV file and are different than the default dimensions of the FLVPlayback component. The default value is `false`. For more information, see [FLVPlayback.autoSize on page 558](#).

bufferTime The number of seconds to buffer the FLV file in memory before beginning playback. This parameter affects streaming FLV files, which are buffered in memory but not downloaded. For an FLV file that is progressively downloaded over HTTP, there is little advantage to increasing this value, although it can improve viewing a high-quality video on an older, slower computer. The default value is 0.1. For more information, see [FLVPlayback.bufferTime on page 571](#).

NOTE

Setting this parameter does not guarantee that a certain amount of the FLV file will download before playback begins.

contentPath A string that specifies the URL to an FLV file or an XML file that describes how to play one or more FLV files. You can specify a path on your local computer, an HTTP path, or a Real-Time Messaging Protocol (RTMP) path. Double-click the value cell for this parameter to open the Content Path dialog box. The default is an empty string.

If you do not specify a value for the `contentPath` parameter, nothing happens when Flash executes the FLVPlayback instance. For more information, see [“Specifying the contentPath parameter” on page 512](#).

cuePoints A string that describes the cue points for the FLV file. Cue points allow you to synchronize specific points in the FLV file with Flash animation, graphics, or text. The default value is an empty string. For more information, see [“Using cue points” on page 513](#).

isLive A Boolean value that, if `true`, specifies that the FLV file is streaming live from Flash Communication Server. One example of a live stream is a video of news events as they are taking place. The default value is `false`. For more information, see [FLVPlayback.isLive on page 601](#).

maintainAspectRatio A Boolean value that, if `true`, resizes the video player within the FLVPlayback component to retain the aspect ratio of the source FLV file; the source FLV file is scaled to the dimensions of the FLVPlayback component on the Stage. The `autoSize` parameter takes precedence over this parameter. The default value is `true`. For more information, see [FLVPlayback.maintainAspectRatio on page 605](#).

skin A parameter that opens the Select Skin dialog box from which you can select a skin for the component. The default value is initially a predesigned skin, but it subsequently becomes the last selected skin. If you select `None`, the FLVPlayback instance does not have control elements to operate the FLV file. If the `autoPlay` parameter is set to `true`, the FLV file plays automatically. For more information, see “[Customizing the FLVPlayback component](#)” on page 524.

skinAutoHide A Boolean value that, if `true`, hides the skin when the mouse is not over the FLV file or the skin region, if it is an external skin that is not on the FLV file viewing area. The default value is `false`. For more information, see [FLVPlayback.skin on page 670](#).

totalTime The total number of seconds, to a precision of milliseconds, in the source FLV file. The default value is 0.

If you use FCS or FVSS, the component always takes the total time from the server.

If you use progressively download over HTTP, the component uses this number if it is set to a value greater than zero. Otherwise, it tries to take the time from the FLV file metadata. For more information, see [FLVPlayback.totalTime on page 683](#).

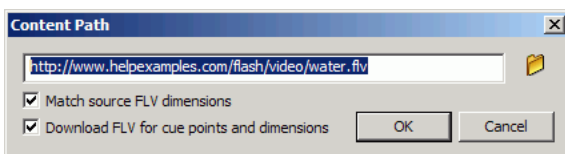
volume A number from 0 to 100 that represents a percentage of the maximum volume (100). For more information, see [FLVPlayback.volume on page 690](#).

Each of these parameters has an equivalent property in the FLVPlayback class. Setting the property overrides the parameter setting in the Component inspector or the Property inspector.

Specifying the contentPath parameter

The `contentPath` parameter lets you specify the name and location of the FLV file, both of which inform Flash how to play the file.

Open the Content Path dialog box by double-clicking the Value cell for the `contentPath` parameter in the Component inspector. The dialog box looks like the following figure:



The dialog box provides two check boxes that can determine the dimensions of the FLVPlayback instance and specify whether to acquire the dimensions and cue point information from the FLV file. For more information, see [“The FLV file options” on page 513](#).

The content path

Enter the URL or local path for either the FLV file or an XML file that describes how to play the FLV file. If you do not know the exact location of an FLV file, click the folder icon to open a Browser dialog box to help you find the correct location. When browsing for an FLV file, if it is at or below the location of the target SWF file, Flash automatically makes the path relative to that location so you can serve it from a web server. Otherwise, the path is an absolute Windows or Macintosh path. To enter the name of a local XML file, you must type the path and name.

If you specify an HTTP URL, the FLV file plays as a progressive download. If you specify a URL that is an RTMP URL, the FLV file streams from a FCS or a FVSS. A URL to an XML file could also be a streaming FLV file from a FCS or a FVSS.

CAUTION

When you click OK in the Content Path dialog box, the component updates the value of the `cuePoints` parameter because it might no longer apply if the content path changed. As a result, you *could* lose any disabled cue points, but not ActionScript cue points. (You will not lose disabled cue points if the new FLV file contains the same cue points, which can happen if you simply change the path.) For this reason, you might want to disable non-ActionScript cue points through ActionScript rather than through the Cue Points dialog box.

You can also specify the location of an XML file that describes how to play multiple FLV file streams for multiple bandwidths. The XML file uses the Synchronized Multimedia Integration Language (SMIL) to describe the FLV files. For a description of the XML SMIL file, see [“Using a SMIL file” on page 712](#).

You can also specify the name and location of the FLV file using the `ActionScript.FLVPlayback.contentPath` property and the `FLVPlayback.play()` and `FLVPlayback.load()` methods. These three alternatives take precedence over the `contentPath` parameter in the Component inspector. For more information, see [FLVPlayback.contentPath on page 579](#), [FLVPlayback.play\(\) on page 620](#) and [FLVPlayback.load\(\) on page 603](#).

The FLV file options

The Content Path dialog box also has two options. The first option, Match Source FLV Dimensions, specifies whether the FLVPlayback instance on the Stage should match the dimensions of the source FLV file. The source FLV file contains preferred height and width dimensions for playing. If you select the first option, the dimensions of the FLVPlayback instance are resized to match these preferred dimensions. However, this option is available only if the second option is also checked.

The second option, Download FLV for Cue Points and Dimensions, is enabled only if the content path is an HTTP or RTMP URL, which means the FLV file is not local. Any path that does not end in `.flv` is also considered a network path because it must be an XML file and could point to FLV files anywhere. This option specifies whether to download or stream a portion of the FLV file to acquire the FLV file dimensions and any cue point definitions that are embedded within it. Flash uses the dimensions to resize the FLVPlayback instance, and it loads the cue point definitions into the `cuePoints` parameter in the Component inspector. If this option is not selected, the first option is disabled.

Using cue points

A cue point is a point at which the video player dispatches a `cuePoint` event while an FLV file plays. You can add cue points to an FLV file at times that you want to interact with another element on the web page. You might want to display text or a graphic, for example, or synchronize with a Flash animation or affect the playing of the FLV file by pausing it, seeking to a different point in time, or switching to a different FLV file. Cue points let you receive control in your ActionScript code and synchronize those points in your FLV file with other actions on the web page.

There are three types of cue points: navigation, event, and ActionScript. The navigation and event cue points are also known as *embedded* cue points because they are embedded in the FLV file stream and in the FLV file's metadata packet.

A *navigation cue point* allows you to seek to a particular frame in the FLV file because it creates a *keyframe* within the FLV file as near as possible to the time that you specify. A keyframe is a data segment that occurs between image frames in the FLV file stream. When you seek to a navigation cue point, the component seeks to the keyframe and starts the `cuePoint` event.

An *event cue point* enables you to synchronize a point in time within the FLV file with an external event on the web page. The `cuePoint` event occurs precisely at the specified time. You can embed navigation and event cue points in an FLV file using either the Video Import wizard or the Flash Video encoder. For more information on the Video Import wizard and the Flash Video encoder, see Chapter 11, “Working with Video,” in *Using Flash*.

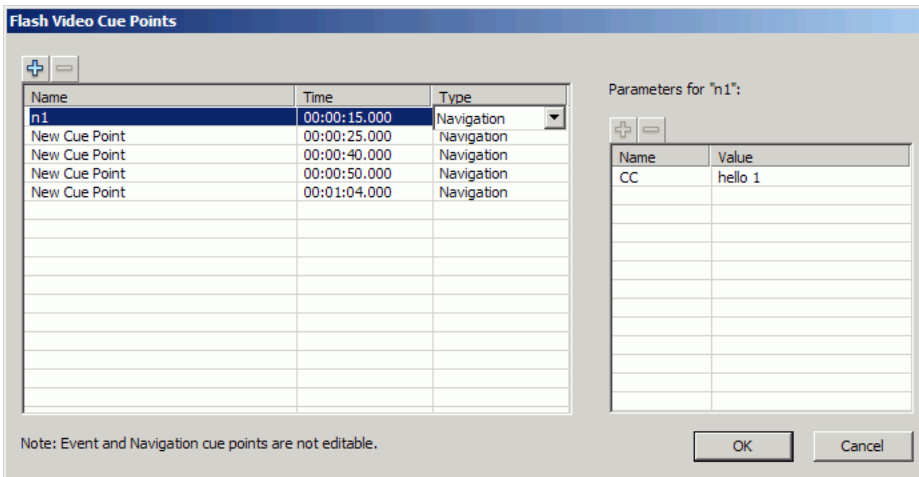
An *ActionScript cue point* is an external cue point that you can add either through the component’s Flash Video Cue Points dialog box or through the `FLVPlayback.addASCuePoint()` method. The component stores and tracks ActionScript cue points apart from the FLV file, and consequently, they are less accurate than embedded cue points. ActionScript cue points are accurate to a tenth of a second. You can increase the accuracy of ActionScript cue points by lowering the value of the `playheadUpdateInterval` property because the component generates the `cuePoint` event for ActionScript cue points when the playhead updates. For more information, see [“FLVPlayback.playheadUpdateInterval” on page 626](#).

In ActionScript and within the FLV file’s metadata, a cue point is represented as an object with the following properties: `name`, `time`, `type`, and `parameters`. The `name` property is a string that contains the assigned name of the cue point. The `time` property is a number representing the time in hours, minutes, seconds, and milliseconds (HH:MM:SS.mmm) when the cue point occurs. The `type` property is a string whose value is “navigation”, “event”, or “actionscript”, depending on the type of cue point that you created. The `parameters` property is an array of specified name-and-value pairs.

When a `cuePoint` event occurs, the cue point object is available in the event object through the `info` property. For more information, see [“Listening for cuePoint events” on page 517](#).

Using the Flash Video Cue Points dialog box

Open the Flash Video Cue Points dialog box by double-clicking the Value cell of the `cuePoints` parameter in the Component inspector. The dialog box looks like the following figure:



The dialog box displays embedded and ActionScript cue points. You can use this dialog box to add and delete ActionScript cue points as well as cue point parameters. You can also enable or disable embedded cue points. However, you cannot add, change, or delete embedded cue points.

To add an ActionScript cue point:

1. Double-click the value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.
2. Click the plus (+) sign in the upper-left corner, above the list of cue points, to add a default ActionScript cue point entry.
3. Click the New Cue Point text in the Name column, and edit the text to name the cue point.
4. Click the Time value of 00:00:00:000 to edit it, and assign a time for the cue point to occur. You can specify the time in hours, minutes, seconds, and milliseconds (HH:MM:SS.mmm).

If multiple cue points exist, the dialog box moves the new cue point to its chronological position in the list.

5. To add a parameter for the selected cue point, click the plus (+) sign above the Parameters section, and enter values in the Name and Value columns. Repeat this step for each parameter.
6. To add more ActionScript cue points, repeat steps 2 through 5 for each one.
7. Click OK to save your changes.

To delete an ActionScript cue point:

1. Double-click the value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.
2. Select the cue point that you want to delete.
3. Click the minus (-) sign in the upper-left corner, above the list of cue points, to delete it.
4. Repeat steps 2 and 3 for each cue point that you want to delete.
5. Click OK to save your changes.

To enable or disable an embedded FLV file cue point:

1. Double-click the value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.
2. Select the cue point you want to enable or disable.
3. Click the value in the Type column to trigger the pop-up menu, or click the Down arrow.
4. Click the name of the type of cue point (for example, Event or Navigation) to enable it. Click Disabled to disable it.
5. Click OK to save your changes.

Using ActionScript with cue points

You can use ActionScript to add ActionScript cue points, listen for cuePoint events, find cue points of any type or a specific type, seek to a navigation cue point, enable or disable a cue point, check whether a cue point is enabled, and remove a cue point.

The examples in this section use an FLV file called `cuepoints.flv`, which contains the following three cue points:

Name	Time	Type
point1	00:00:00.418	Navigation
point2	00:00:07.748	Navigation
point3	00:00:16.020	Navigation

Adding ActionScript cue points

You can add ActionScript cue points to an FLV file using the `addASCuePoint()` method. The following example adds two ActionScript cue points to the FLV file when it is ready to play. It adds the first cue point using a cue point object, which specifies the time, name, and type of the cue point in its properties. The second call specifies the time and name using the method's `time` and `name` parameters.

```
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv"
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 2.02;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlybk.addASCuePoint(5, "ASpt2");
```

For more information, see [FLVPlayback.addASCuePoint\(\) on page 552](#).

Listening for cuePoint events

The `cuePoint` event allows you to receive control in your ActionScript code when a `cuePoint` event occurs. When cue points occur in the following example, the `cuePoint` listener calls an event handler function that displays the value of the `playheadTime` property and the name and type of the cue point.

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    trace("Elapsed time in seconds: " + my_FLVPlybk.playheadTime);
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point type is: " + eventObject.info.type);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

For more information on the `cuePoint` event, see [FLVPlayback.cuePoint on page 580](#).

Finding cue points

Using ActionScript, you can find a cue point of any type, find the nearest cue point to a time, or find the next cue point with a specific name.

The ready event handler in the following example calls the `findCuePoint()` method to find the cue point `ASpt1` and then calls the `findNearestCuePoint()` method to find the navigation cue point that is nearest to the time of cue point `ASpt1`:

```
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv"
var rtn_obj:Object = new Object(); //create cue point object
my_FLVPlayback.addASCuePoint(2.02, "ASpt1"); //add AS cue point
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    rtn_obj = my_FLVPlayback.findCuePoint("ASpt1");
    traceit(rtn_obj);
    rtn_obj = my_FLVPlayback.findNearestCuePoint(rtn_obj.time,
    FLVPlayback.NAVIGATION);
    traceit(rtn_obj);
}
my_FLVPlayback.addEventListener("ready", listenerObject);
function traceit(cuePoint:Object):Void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

In the following example, the ready event handler finds cue point `ASpt` and calls the `findNextCuePointWithName()` method to find the next cue point with the same name:

```
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv"
var rtn_obj:Object = new Object(); //create cue point object
my_FLVPlayback.addASCuePoint(2.02, "ASpt"); //add AS cue point
my_FLVPlayback.addASCuePoint(3.4, "ASpt"); //add 2nd ASpt
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    rtn_obj = my_FLVPlayback.findCuePoint("ASpt");
    traceit(rtn_obj);
    rtn_obj = my_FLVPlayback.findNextCuePointWithName(rtn_obj);
    traceit(rtn_obj);
}
my_FLVPlayback.addEventListener("ready", listenerObject);
function traceit(cuePoint:Object):Void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

For more information about finding cue points, see [FLVPlayback.findCuePoint\(\)](#) on page 586, [FLVPlayback.findNearestCuePoint\(\)](#) on page 589, and [FLVPlayback.findNextCuePointWithName\(\)](#) on page 592.

Seeking navigation cue points

You can seek to a navigation cue point, seek to the next navigation cue point from a specified time, and seek to the previous navigation cue point from a specified time. The following example plays the FLV file `cuepoints.flv` and seeks to the cue point at 7.748 when the `ready` event occurs. When the `cuePoint` event occurs, the example calls the `seekToPrevNavCuePoint()` method to seek to the first cue point. When that `cuePoint` event occurs, the example calls the `seekToNextNavCuePoint()` method to seek to the last cue point by adding 10 seconds to `eventObject.info.time`, which is the time of the current cue point.

```
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    my_FLVPlybk.seekToNavCuePoint("point2");
}
my_FLVPlybk.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
    trace(eventObject.info.time);
    if(eventObject.info.time == 7.748)
        my_FLVPlybk.seekToPrevNavCuePoint(eventObject.info.time - .005);
    else
        my_FLVPlybk.seekToNextNavCuePoint(eventObject.info.time + 10);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
my_FLVPlybk.contentPath = "http://helpexamples.com/flash/video/
cuepoints.flv";
```

For more information, see [FLVPlayback.seekToNavCuePoint\(\) on page 660](#), [FLVPlayback.seekToNextNavCuePoint\(\) on page 661](#), and [FLVPlayback.seekToPrevNavCuePoint\(\) on page 663](#).

Enabling and disabling embedded FLV file cue points

You can enable and disable embedded FLV file cue points using the `setFLVCuePointEnabled()` method. Disabled cue points do not trigger `cuePoint` events and do not work with the `seekToCuePoint()`, `seekToNextNavCuePoint()`, and `seekToPrevNavCuePoint()` methods. You can find disabled cue points, however, with the `findCuePoint()`, `findNearestCuePoint()`, and `findNextCuePointWithName()` methods.

You can test whether an embedded FLV file cue point is enabled using the `isFLVCuePointEnabled()` method. The following example disables the embedded cue points `point2` and `point3` when the video is ready to play. When the first `cuePoint` event occurs, however, the event handler tests to see if cue point `point3` is disabled and, if so, enables it.

```
import mx.video.*;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlaybk.setFLVCuePointEnabled(false, "point2");
    my_FLVPlaybk.setFLVCuePointEnabled(false, "point3");
}
my_FLVPlaybk.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    trace("Cue point time is: " + eventObject.info.time);
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point type is: " + eventObject.info.type);
    if (my_FLVPlaybk.isFLVCuePointEnabled("point2") == false) {
        my_FLVPlaybk.setFLVCuePointEnabled(true, "point2");
    }
}
my_FLVPlaybk.addEventListener("cuePoint", listenerObject);
```

For more information, see [FLVPlayback.isFLVCuePointEnabled\(\) on page 599](#) and [FLVPlayback.setFLVCuePointEnabled\(\) on page 665](#).

Removing an ActionScript cue point

You can remove an ActionScript cue point using the `removeASCuePoint()` method. The following example removes the cue point `ASpt2` when cue point `ASpt1` occurs:

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    trace("Cue point name is: " + eventObject.info.name);
    if (eventObject.info.name == "ASpt1") {
        my_FLVPlaybk.removeASCuePoint("ASpt2");
        trace("Removed cue point ASpt2");
    }
}
my_FLVPlaybk.addEventListener("cuePoint", listenerObject);
```

For more information, see [FLVPlayback.removeASCuePoint\(\) on page 638](#).

Playing multiple FLV files

You can play FLV files sequentially in an FLVPlayback instance simply by loading a new URL in the `contentPath` property when the previous FLV file finishes playing. For example, the following ActionScript code listens for the `complete` event, which occurs when an FLV file finishes playing. When this event occurs, the code sets the name and location of a new FLV file in the `contentPath` property and calls the `play()` method to play the new video.

```
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
var listenerObject:Object = new Object();
// listen for complete event; play new FLV
listenerObject.complete = function(eventObject:Object):Void {
    if (my_FLVPlayback.contentPath == "http://www.helpexamples.com/flash/video/
      clouds.flv") {
        my_FLVPlayback.play("http://www.helpexamples.com/flash/video/water.flv");
    }
};
my_FLVPlayback.addEventListener("complete", listenerObject);
```

Using multiple video players

You can also open multiple video players within a single instance of the FLVPlayback component to play multiple videos and switch between them as they play.

You create the initial video player when you drag the FLVPlayback component to the Stage. The component automatically assigns the initial video player the number 0 and makes it the default player. To create an additional video player, simply set the `activeVideoPlayerIndex` property to a new number. Setting the `activeVideoPlayerIndex` property also makes the specified video player the *active* video player, which is the one that will be affected by the properties and methods of the FLVPlayback class. Setting the `activeVideoPlayerIndex` property does not make the video player visible, however. To make the video player *visible*, set the `visibleVideoPlayerIndex` property to the video player's number. For more information on how these properties interact with the methods and properties of the FLVPlayback class, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The following ActionScript code loads the `contentPath` property to play an FLV file in the default video player and adds a cue point for it. When the `ready` event occurs, the event handler opens a second video player by setting the `activeVideoPlayerIndex` property to the number 1. It specifies an FLV file and a cue point for the second video player and then makes the default player (0) the active video player again.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
// add a cue point to the default player
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
my_FLVPlybk.addASCuePoint(3, "1st_switch");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    // add a second video player and create a cue point for it
    my_FLVPlybk.activeVideoPlayerIndex = 1;
    my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
      water.flv";
    my_FLVPlybk.addASCuePoint(3, "2nd_switch");
    my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
```

To switch to another FLV file while one is playing, you must obtain control to make the switch in your ActionScript code. Cue points allow you to intervene at specific points in the FLV file using a `cuePoint` event. The following code creates a listener for the `cuePoint` event and calls a handler function that pauses the active video player (0), switches to the second player (1), and plays its FLV file:

```
// create a listener object
var listenerObject:Object = new Object();
// add a handler function for the cuePoint event
listenerObject.cuePoint = function(eventObject:Object):Void {
    // display the no. of the video player causing the event
    trace("Hit cuePoint event for player: " + eventObject.vp);
    // test for the video player and switch FLVs accordingly
    if (eventObject.vp == 0) {
        my_FLVPlybk.pause(); //pause the first FLV
        my_FLVPlybk.activeVideoPlayerIndex = 1; // make the 2nd player active
        my_FLVPlybk.visibleVideoPlayerIndex = 1; // make the 2nd player
        visible
        my_FLVPlybk.play(); // begin playing the new player/FLV
    } else if (eventObject.vp == 1) {
        my_FLVPlybk.pause(); // pause the 2nd FLV
        my_FLVPlybk.activeVideoPlayerIndex = 0; // make the 1st player active
    }
};
```

```

        my_FLVPlayback.visibleVideoPlayerIndex = 0; // make the 1st player
        visible
        my_FLVPlayback.play(); // begin playing the 1st player
    }
}
// add listener for a cuePoint event
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
    trace("Hit complete event for player: " + eventObject.vp);
    if (eventObject.vp == 0) {
        my_FLVPlayback.activeVideoPlayerIndex = 1;
        my_FLVPlayback.visibleVideoPlayerIndex = 1;
        my_FLVPlayback.play();
    } else {
        my_FLVPlayback.closeVideoPlayer(1);
    }
}
my_FLVPlayback.addEventListener("complete", listenerObject);

```

When you create a new video player, the `FLVPlayback` instance sets its properties to the value of the default video player, except for the `contentPath`, `totalTime`, and `isLive` properties, which the `FLVPlayback` instance always sets to the default values: empty string, 0, and `false`, respectively. It sets the `autoPlay` property, which defaults to `true` for the default video player, to `false`. The `cuePoints` property has no effect, and it has no effect on a subsequent load into the default video player.

The methods and properties that control volume, positioning, dimensions, visibility, and user interface controls are always global and their behavior is *not* affected by setting the `activeVideoPlayerIndex` property. For more information on these methods and properties and the effect of setting the `activeVideoPlayerIndex` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#). The remaining properties and methods target the video player identified by the value of the `activeVideoPlayerIndex` property.

Properties and methods that control dimensions *do interact* with the `visibleVideoPlayerIndex` property, however. For more information, see [“FLVPlayback.visibleVideoPlayerIndex” on page 688](#).

Streaming FLV files from a FCS

If you use a FCS to stream FLV files to the FLVPlayback component, you must add the main.asc file to your Flash Communication Server FLV application. You can find the main.asc file in your Flash 8 application folder under Flash 8/Samples and Tutorials/Samples/Components/FLVPlayback/main.asc.

To set up your FCS for streaming FLV files:

1. Create a folder in your FCS application folder, and give it a name such as **my_application**.
2. Copy the main.asc file into the my_application folder.
3. Create a folder named **streams** in the my_application folder.
4. Create a folder named **_definst_** inside the streams folder.
5. Place your FLV files in the **_definst_** folder.

To access your FLV files on the Flash Communication Server, use a URL such as `rtmp://my_servername/my_application/stream.flv`.

For more information on administering the Flash Communication Server, including how to set up a live stream, see the FCS documentation at www.macromedia.com/support/documentation/en/flashcom/. When playing a live stream with FCS, you need to set the FLVPlayback property `isLive` to `true`. For more information, see [FLVPlayback.isLive](#) on page 601.

Customizing the FLVPlayback component

This section explains how to customize the FLVPlayback component. For a comprehensive overview of customizing components, which includes terminology and the basic concepts of working with styles, skins, and themes, see “Customizing Components” in *Using Components*. Most of the methods used to customize other components, however, do not work with the FLVPlayback component. To customize the FLVPlayback component, use only the techniques described in this section.

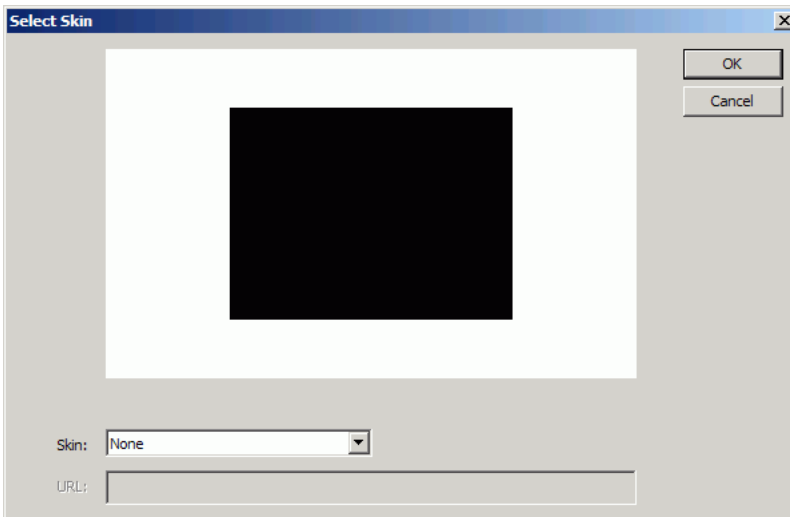
You have the following choices for customizing the FLVPlayback component: select a predesigned skin, skin FLV Playback Custom UI components individually, or create a new skin. You can also use FLVPlayback properties to modify the behavior of a skin.

NOTE

You must upload your skin SWF file to the web server along with your application SWF file for the skin to work with your FLVPlayback component.

Selecting a predesigned skin

You can select a skin for the FLVPlayback component by clicking the value cell for the `skin` parameter in the Component inspector. Then click the magnifying glass icon to open the following Select Skin dialog box, in which you can select a skin or provide a URL that specifies the location of the skin SWF file.



Skins that are listed in the Skin pop-up menu are located in the Flash 8 Configuration/Skins folder or in the user's local Configuration/Skins folder. You can make new skins available to this dialog box by creating them and placing the SWF file in the Skins folder. The name appears in the pop-up menu with a `.swf` extension. For more information about creating a skin set, see [“Creating a new skin” on page 532](#).

If you want to skin the FLVPlayback component using the FLV Playback Custom UI components, select `None` from the pop-up menu.

Skinning FLV Playback Custom UI components individually

The FLV Playback Custom UI components allow you to customize the appearance of the FLVPlayback controls within your FLA file and allow you to see the results when you preview your web page. These components are not designed to be scaled, however. You should edit a movie clip and its contents to be a specific size. For this reason, it is generally best to have the FLVPlayback component on the Stage at the desired size, with the `autoSize` and `maintainAspectRatio` properties set to `false`.

To begin, simply drag the FLV Playback Custom UI components that you want from the Components panel, place them where you want them on the Stage, and give each one an instance name in the Property inspector. After your components are on the Stage, you edit them as you would any other symbol.

After you open the components, you can see that each one is set up a little differently from the others.

Button components

The button components have a similar structure. The buttons include the BackButton, ForwardButton, MuteButton, PauseButton, PlayButton, PlayPauseButton, and StopButton. Most have a single movie clip on Frame 1 with the instance name placeholder_mc. This is usually an instance of the normal state for the button, but not necessarily so. On Frame 2, there are four movie clips on the Stage for each display state: normal, over, down and disabled. (At runtime, the component never actually goes to Frame 2; these movie clips are placed here to make editing more convenient and to force them to load into the SWF file without checking the Export in First Frame check box in the Symbol Properties dialog box. You must still select the Export for ActionScript option, however.)

To skin the button, you simply edit each of these movie clips. You can change their size as well as their appearance.

Some ActionScript usually appears on Frame 1. You should not need to change this script. It simply stops the playhead on Frame 1 and specifies which movie clips to use for which states.

PlayPauseButton and MuteButton buttons

The PlayPauseButton and MuteButton buttons are set up differently than the other buttons; they have only one frame with two layers and no script. On that frame, there are two buttons, one on top of the other—in the first case, a Play and a Pause button; in the second case, a Mute-on and a Mute-off button. To skin the PlayPauseButton or MuteButton buttons, skin each of these two internal buttons as described in [“Skinning FLV Playback Custom UI components individually” on page 525](#); no additional action is required.

BackButton and ForwardButton buttons

The BackButton and ForwardButton buttons are also set up differently than the other buttons. On Frame 2, they have extra movie clips that you can use as a frame around one or both of the buttons. These movie clips are not required and have no special capability; they are provided only as a convenience. To use them, simply drag them on the Stage from your Library panel and place them where you want them. If you don't want them, either don't use them or delete them from your Library panel.

Most of the buttons, as supplied, are based on a common set of movie clips so that you can change the appearance of all the buttons at once. You can use this capability, or you can replace those common clips and make every button look different.

BufferingBar component

The buffering bar component is simple: It consists of an animation that is made visible when the component enters the buffering state, and it does not require any special ActionScript to configure it. By default, it is a striped bar moved from left to right with a rectangular mask on it to give it a “barber pole” effect, but there is nothing special about this configuration.

Although the buffering bars in the skin SWF files use 9-slice scaling because they need to be scaled at runtime, the BufferingBar FLV Custom UI Component does not and *cannot* use 9-slice scaling because it has nested movie clips. If you want to make the BufferingBar wider or taller, you might want to change its contents rather than scale it.

SeekBar and VolumeBar components

The SeekBar and VolumeBar components are similar, although they have different functions. Each has handles, uses the same handle tracking mechanisms, and has support for clips nested within to track progress and fullness.

There are many places where the ActionScript code in the FLVPlayback component assumes that the registration point of your SeekBar or VolumeBar component is at the upper-left corner of the content, so it is important to maintain this convention. Otherwise, you might have problems with handles and with progress and fullness movie clips.

Although the seek bars in the skin SWF files use 9-slice scaling because they need to be scaled at runtime, the SeekBar FLV Custom UI component does not and *cannot* use 9-slice scaling because it has nested movie clips. If you want to make the SeekBar wider or taller, you might want to change its contents rather than scale it.

Handle

An instance of the handle movie clip is on Frame 2. As with the BackButton and ForwardButton components, the component never actually goes to Frame 2; these movie clips are placed here to make editing more convenient and as a way to force them to be loaded into the SWF file without checking the Export in First Frame check box in the Symbol Properties dialog box. You still must select the Export for ActionScript option, however.

You might notice that the handle movie clip has a rectangle in the background with alpha set to 0. This rectangle increases the size of the handle's hit area, making it easier to grab without changing its appearance, similar to the hit state of a button. Because the handle is created dynamically at runtime, it must be a movie clip and not a button. This rectangle with alpha set to 0 is not necessary for any other reason and, generally, you can replace the inside of the handle with any image you want. It works best, however, to keep the registration point centered horizontally in the middle of the handle movie clip.

The following ActionScript code is on Frame 1 of the SeekBar component to manage the handle:

```
stop();
handleLinkageID = "SeekBarHandle";
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

The call to the `stop()` function is necessary due to the content of Frame 2.

The second line specifies which symbol to use as the handle, and you should not need to change this if you simply edit the handle movie clip instance on Frame 2. At runtime, the `FLVPlayback` component creates an instance of the specified movie clip on the Stage as a sibling of the `Bar` component instance, which means that they have the same parent movie clip. So, if your bar is at the root level, your handle must also be at the root level.

The variable `handleLeftMargin` determines the handle's original location (0%), and the variable `handleRightMargin` determines where it is at the end (100%). The numbers give the offsets from the left and right ends of the bar control, with positive numbers marking the limits within the bar, and negative numbers marking the limits outside the bar. These offsets specify where the handle can go, based on its registration point. If you put your registration point in the middle of the handle, the handle's far left and right sides will go past the margins. A seek bar movie clip must have its registration point as the upper-left corner of its content to work properly.

The variable `handleY` determines the *y* position of the handle, relative to the bar instance. This is based on the registration points of each movie clip. The registration point in the sample handle is at the tip of the triangle to place it relative to the visible part, disregarding the invisible hit state rectangle. Also, the bar movie clip must keep its registration point as the upper-left corner of its content to work properly.

So, for example, with these limits, if there a bar control is set at (100, 100), and it is 100 pixels wide, the handle can range from 102 to 198 horizontally and stay at 111 vertically. If you change the `handleLeftMargin` and `handleRightMargin` to -2 and `handleY` to -11, the handle can range from 98 to 202 horizontally and stay at 89 vertically.

Progress and fullness movie clips

The SeekBar component has a *progress* movie clip and the VolumeBar has a *fullness* movie clip, but in practice, any SeekBar or VolumeBar can have either, neither, or both of these movie clips. They are structurally the same and behave similarly but track different values. A progress movie clip fills up as the FLV file downloads (which is useful for an HTTP download only, because it is always full if streaming from FCS) and a fullness movie clip fills up as the handle moves from left to right.

The FLVPlayback component finds these movie clip instances by looking for a specific instance name, so your progress movie clip instance must have your bar movie clip as its parent and have the instance name `progress_mc`. The fullness movie clip instance must have the instance name `fullness_mc`.

You can set the progress and fullness movie clips with or without the `fill_mc` movie clip instance nested inside. The VolumeBar `fullness_mc` movie clip shows the method *with* the `fill_mc` movie clip, and the SeekBar `progress_mc` movie clip shows the method *without* the `fill_mc` movie clip.

The method with the `fill_mc` movie clip nested inside is useful when you want a fill that cannot be scaled without distorting the appearance.

In the VolumeBar `fullness_mc` movie clip, the nested `fill_mc` movie clip instance is masked. You can either mask it when you create the movie clip, or a mask will be created dynamically at runtime. If you mask it with a movie clip, name the instance `mask_mc` and set it up so that `fill_mc` appears as it would when percentage is 100%. If you do not mask `fill_mc`, the dynamically created mask will be rectangular and the same size as `fill_mc` at 100%.

The `fill_mc` movie clip is revealed with the mask in one of two ways, depending on whether `fill_mc.slideReveal` is `true` or `false`.

If `fill_mc.slideReveal` is `true`, then `fill_mc` is moved from left to right to expose it through the mask. At 0%, it is all the way to the left so, that none of it shows through the mask. As the percentage increases, it moves to the right, until at 100%, it is back where it was created on the Stage.

If `fill_mc.slideReveal` is `false` or undefined (the default behavior), the mask will be resized from left to right to reveal more of `fill_mc`. When it is at 0%, the mask will be scaled to 05 horizontally, and as the percentage increases, the `_xscale` increases until, at 100%, it reveals all of `fill_mc`. This is not necessarily `_xscale = 100` because `mask_mc` might have been scaled when it was created.

The method without `fill_mc` is simpler than the method with `fill_mc`, but it distorts the fill horizontally. If you do not that distortion, you must use `fill_mc`. The SeekBar `progress_mc` illustrates this method.

The progress or fullness movie clip is scaled horizontally based on the percentage. At 0%, the instance's `_xscale` is set to 0, making it invisible. As the percentage grows, the `_xscale` is adjusted until, at 100%, the clip is the same size it was on the Stage when it was created. Again, this is not necessarily `_xscale = 100` because the clip instance might have been scaled when it was created.

Connecting your FLV Playback Custom UI components

You must write ActionScript code to connect your FLV Playback Custom UI components to your instance of the FLVPlayback component. First, you must assign a name to the FLVPlayback instance and then use ActionScript to assign your FLV Playback Custom UI component instances to the corresponding FLVPlayback properties. In the following example, the FLVPlayback instance is `my_FLVPlaybk`, the FLVPlayback property names follow the periods (`.`), and the FLV Playback Custom UI control instances are to the right of the equal (`=`) signs:

```
//FLVPlayback instance = my_FLVPlaybk
my_FLVPlaybk.playButton = playbtn; // set playButton property to playbtn,
    etc.
my_FLVPlaybk.pauseButton = pausebtn;
my_FLVPlaybk.playPauseButton = playpausebtn;
my_FLVPlaybk.stopButton = stopbtn;
my_FLVPlaybk.muteButton = mutebtn;
my_FLVPlaybk.backButton = backbtn;
my_FLVPlaybk.forwardButton = forbtn;
my_FLVPlaybk.volumeBar = volbar;
my_FLVPlaybk.seekBar = seekbar;
my_FLVPlaybk.bufferingBar = bufbar;
```

Example

The following steps create custom StopButton, PlayPauseButton, MuteButton, and SeekBar controls:

1. Drag the FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlaybk**.
2. Set the `contentPath` parameter through the Component inspector to **<http://www.helpexamples.com/flash/video/cuepoints.flv>**.
3. Set the Skin parameter to None.
4. Drag a StopButton, a PlayPauseButton, and a MuteButton to the Stage, and place them over the FLVPlayback instance, stacking them vertically on the left. Give each button an instance name in the Property inspector (such as **my_stopbtn**, **my_playpausebtn**, and **my_mutebtn**).

5. In the Library panel, open the FLVPlayback Skins folder, and then open the SquareButton folder below it.
6. Select the SquareBgDown movie clip, and double-click it to open it on the Stage.
7. Right-click (Windows) or Control-click (Macintosh), select Select All from the menu, and delete the symbol.
8. Select the oval tool, draw an oval in the same location, and set the fill to blue `#(0033FF)`.
9. In Property inspector, set the width (W:) to 40 and the height (H:) to 20. Set the x-coordinate (X:) to 0.0 and y-coordinate (Y:) to 0.0.
10. Repeat steps 6 to 8 for SquareBgNormal, but change the fill to yellow `(#FFFF00)`.
11. Repeat steps 6 to 8 for SquareBgOver, but change the fill to green `(#006600)`.
12. Edit the movie clips for the various symbol icons within the buttons (PauseIcon, PlayIcon, MuteOnIcon, MuteOffIcon, and StopIcon). You can find these movie clips in the Library panel under FLV Playback Skins/*Label* Button/Assets, where *Label* is the name of the button, such as Play, Pause, and so on. Do the following steps for each one:
 - a. Select the Select All option.
 - b. Change the color to red `(#FF0000)`.
 - c. Scale by 300%.
 - d. Change the X: location of the content to 7.0 to alter the horizontal placement of the icon in every button state.

NOTE

By changing the location this way, you avoid opening every button state and moving the icon movie clip instance.

13. Click the blue Back arrow above the Timeline to return to Scene 1, Frame 1.
14. Drag a SeekBar component to the Stage, and place it in the lower-right corner of the FLVPlayback instance.
15. In the Library panel, double-click the SeekBar to open it on the Stage.
16. Scale it to 400%.
17. Select the outline, and set the color to red `(#FF0000)`.
18. Double-click SeekBarProgress in the FLVPlayback Skins/Seek Bar folder, and set the color to yellow `(#FFFF00)`.
19. Double-click SeekBarHandle in the FLVPlayback Skins/Seek Bar folder and set the color to red `(#FF0000)`.
20. Click the blue Back arrow above the Timeline to return to Scene 1, Frame 1.
21. Select the SeekBar instance on the Stage, and give it an instance name of `my_seekbar`.

22.In the Actions panel on Frame 1 of the Timeline, add an import statement for the video classes, and assign the button and seek bar names to the corresponding FLVPlayback properties, as shown in the following example:

```
import mx.video.*;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.playPausebbtn = my_plypausbbtn;
my_FLVPlayback.muteButton = my_mutebbtn;
my_FLVPlayback.seekBar = my_seekbar;
```

23.Press Control+Enter to test the movie.

Creating a new skin

The best way to create a skin SWF file is to copy one of the skin files that come with Flash 8, and use it as a starting point. You can find the FLA files for these skins in the Flash 8 application folder in *<lang code>/Configuration/SkinFLA*. To make your finished skin SWF file available as an option in the Select Skin dialog box, put it in the *<lang code>/Configuration/Skins* folder either in the Flash 8 application folder or in a user's local Configuration/Skins folder.

You will find that simple modifications that change the appearance of a button or of the *chrome* (the background) of a button, without changing the dimensions, are fairly easy. All the installed skins have the same buttons based on different-colored chromes, so you can make dramatic changes by simply changing the color of the chrome. You can make changes, such as rearranging controls in the layout movie clip, by simply moving the placeholder clips. You can see these changes exactly as they will appear in the finished SWF file.

When looking at the installed Flash 8 skin FLA files, it might seem that certain things on the Stage are unnecessary, but many of these things are put into guide layers. To quickly see what actually appears in the SWF file, type **Ctl-Enter** to test the movie. This will also show you how 9-slice scaling affects certain controls, because 9-slice scaling is not in effect when you are authoring.

The following sections cover more complex customizations and changes to the SeekBar, BufferingBar, and VolumeBar movie clips.

Using layout_mc

When you open a Flash 8 skin FLA file, you will find a movie clip named `layout_mc` in the upper-left corner of the Stage. This clip *must* be named `layout_mc`. The `layout_mc` clip and the ActionScript code that you find on the same frame define how the controls will be laid out at runtime.

Although `layout_mc` looks a lot like how the skin will look like at runtime, the contents of this clip are not visible at runtime. It is used only to calculate where to place the controls. The other controls on the Stage will be used at runtime.

Within `layout_mc` is a placeholder for the `FLVPlayback` component named `video_mc`. All the other controls are laid out relative to `video_mc`. If you start with one of the Flash 8 FLA files and change the size of the controls, you can probably fix the layout by moving these placeholder clips.

Each of the placeholder clips has a specific instance name. The names of the placeholder clips are: `playpause_mc`, `play_mc`, `pause_mc`, `stop_mc`, `back_mc`, `bufferingBar_mc`, `seekBar_mc`, `volumeMute_mc`, and `volumeBar_mc`.

Which clip is used for a control is not important. Generally, for buttons the normal state clip is used. For other controls the clip for that control is used, but this is only for convenience. All that is important are the x (horizontal) and y (vertical) location and the height and the width of the placeholder.

You can also have as many background and foreground clips as you want besides the standard controls. You must use the following naming convention, however: `bg1_mc`, `bg2_mc`, and so on for background clips; and `fg1_mc`, `fg2_mc`, and so on for foreground clips. You cannot skip numbers. For example, if you have `bg1_mc` and `bg3_mc` but no `bg2_mc`, `bg3_mc` will not be used. This scheme is designed to put background clips behind the controls, with `bg1_mc` on the bottom and `bg2_mc` above it, and the foreground clips above the controls, with `fg1_mc` first and `fg2_mc` above `fg1_mc`, and so on. However, the layered relationship of the clips is actually determined by the ordering of the corresponding controls on the Stage, so make sure that is correct.

The `bg1_mc` clip is special. If you set the `FLVPlayback.skinAutoHide` property to `true`, the skin shows when the mouse is over the `bg1_mc` clip. This is important for skins that appear outside the bounds of the video player. For information on the `skinAutoHide` property, see [“Modifying skin behavior” on page 538](#).

In the Flash 8 FLA files, `bg1_mc` is used for the chrome, and some use `bg2_mc` for the border around the Forward and Back buttons.

ActionScript

The following ActionScript code applies to all controls generally. Some controls have specific ActionScript that defines additional behavior, and that is explained in the section for that control.

The initial ActionScript code defines the minimum width and height for the skin. The Select Skin dialog box shows these values and they are used at runtime to prevent the skin from scaling below its minimum size. If you do not want to specify a minimum size, leave it as undefined or less than or equal to zero.

```
// minimum width and height of video recommended to use this skin,  
// leave as undefined or <= 0 if there is no minimum  
layout_mc.minWidth = 270;  
layout_mc.minHeight = 60;
```

Each placeholder can have the following properties applied to it:

Property	Description
<code>mc:MovieClip</code>	The instance on the Stage for this control. If not set, <code>layout_mc.foo_mc.mc</code> would default to <code>foo_mc</code> .
<code>anchorLeft:Boolean</code>	Positions the control relative to the left side of the FLVPlayback instance. Defaults to <code>true</code> unless <code>anchorRight</code> is explicitly set to <code>true</code> , and then it defaults to <code>false</code> .
<code>anchorRight:Boolean</code>	Positions the control relative to the right side of the FLVPlayback instance. Defaults to <code>false</code> .
<code>anchorBottom:Boolean</code>	Positions the control relative to the bottom of the FLVPlayback instance. Defaults to <code>true</code> , unless <code>anchorTop</code> is explicitly set to <code>true</code> , and then it defaults to <code>false</code> .
<code>anchorTop:Boolean</code>	Positions the control relative to the top of the FLVPlayback instance. Defaults to <code>false</code> .

If both the `anchorLeft` and `anchorRight` properties are `true`, the control will be scaled horizontally at runtime. If both the `anchorTop` and `anchorBottom` properties are `true`, the control will be scaled vertically at runtime.

To see the effects of these properties, see how they are used in the Flash 8 skins. The `BufferingBar` and `SeekBar` controls are the only ones that scale, and they are laid on top of one another and have both the `anchorLeft` and `anchorRight` properties set to `true`. All controls to the left of the `BufferingBar` and `SeekBar` have `anchorLeft` set to `true`, and all controls to their right have `anchorRight` set to `true`. All controls have `anchorBottom` set to `true`.

You can try editing the `layout_mc` movie clip to make a skin where the controls sit at the top rather than at the bottom. You simply need to move the controls to the top, relative to `video_mc`, and set `anchorTop` equal to `true` for all controls.

Button states

All the button states are laid out on the Stage, but where these movie clip instances are placed on the Stage is not important. It is important, however, that they are nested within movie clips in a specific way and that every clip instance has the correct instance name.

The structure of the clip instances and their instance names are shown in the following example:

```
playpause_mc
  play_mc
    up_mc, over_mc, down_mc, disabled_mc
  pause_mc
    up_mc, over_mc, down_mc, disabled_mc
stop_mc
  up_mc, over_mc, down_mc, disabled_mc
back_mc
  up_mc, over_mc, down_mc, disabled_mc
forward_mc
  up_mc, over_mc, down_mc, disabled_mc
volumeMute_mc
  on_mc
    up_mc, over_mc, down_mc, disabled_mc
  off_mc
    up_mc, over_mc, down_mc, disabled_mc
```

Notice that the Flash 8 FLA files have additional Forward and Back buttons on the Stage. These are on guide layers and are there to show the use of the ForwardBackBorder, ForwardBorder, and BackBorder movie clips. For more information, see [“Background and foreground clips” on page 538](#).

You can edit the various states as desired. Remember that all states are placed in the same place by their registration points, so if some states are bigger than others, you might not be able to place your art at (0, 0) as it is in most of the Flash 8 button skins. You might find it easier, in some cases, to keep the registration point in the center of the art.

If you do not want to use all the states, you can omit some, but you should include `up_mc`. The `up_mc` clip is used for omitted states.

If you want to have separate Play and Pause buttons, rather than a combined Play-Pause button, simply place the `play_mc` and `pause_mc` clips on the Stage without wrapping them with a `playpause_mc` clip.

No additional ActionScript code is necessary to set up the buttons besides the code described in [“Using layout_mc” on page 532](#).

Buffering bar

The buffering bar has two movie clips: `bufferingBar_mc` and `bufferingBarFill_mc`. Each clip's position on the Stage relative to the other clip is important because this relative positioning is maintained. The buffering bar uses two separate clips because the component scales `bufferingBar_mc` but not `bufferingBarFill_mc`.

The `bufferingBar_mc` clip has 9-slice scaling applied to it, so the borders won't distort when it scales. The `bufferingBarFill_mc` clip is extremely wide, so that it will always be wide enough without needing to be scaled. It is automatically masked at runtime to show only the portion above the stretched `bufferingBar_mc`. By default, the exact dimensions of the mask will maintain an equal margin on the left and right within the `bufferingBar_mc`, based on the difference between the x (horizontal) positions of `bufferingBar_mc` and `bufferingBarFill_mc`. You can customize the positioning with ActionScript code.

If your buffering bar does not need to scale or does not use 9-slice scaling, you could set it up like the FLV Playback Custom UI BufferingBar component. For more information, see [“BufferingBar component” on page 527](#).

The buffering bar has the following additional property:

Property	Description
<code>fill_mc:MovieClip</code>	Specifies the instance name of the buffering bar fill. Defaults to <code>bufferingBarFill_mc</code> .

Seek bar and volume bar

The seek bar also has two movie clips: `seekBar_mc` and `seekBarProgress_mc`. Each clip's position on the Stage relative to the other clip is important because this relative positioning is maintained. Although both clips scale, the `seekBarProgress_mc` cannot be nested within `seekBar_mc` because `seekBar_mc` uses 9-slice scaling, which does not work well with nested movie clips.

The `seekBar_mc` clip has 9-slice scaling applied to it, so the borders won't distort when it scales. The `seekBarProgress_mc` clip also scales, but it does distort. It does not use 9-slice scaling because it is a fill, which looks fine when distorted.

The `seekBarProgress_mc` clip works without a `fill_mc`, much like the way a `progress_mc` clip works in FLV Playback Custom UI Components. In other words, it is not masked and is scaled horizontally. The exact dimensions of the `bufferingBarProgress_mc` at 100% is defined by left and right margins within the `bufferingBar_mc` clip. These dimensions are, by default, equal and based on the difference between the x (horizontal) positions of `seekBar_mc` and `seekBarProgress_mc`. You can customize the dimensions with ActionScript in the seek bar movie clip, as shown in the following example:

```
progressLeftMargin = 2;
progressRightMargin = 2;
progressY = 11;
fullnessLeftMargin = 2;
fullnessRightMargin = 2;
fullnessY = 11;
```

As with the FLV Playback Custom UI SeekBar component, it is possible to create a fullness movie clip for the seek bar. If your seek bar does not need to scale, or if it does scale but does not use 9-slice scaling, you could set up your `progress_mc` or `fullness_mc` using any of the methods used for FLV Playback Custom UI components. For more information, see [“Progress and fullness movie clips” on page 529](#).

Because the volume bar in the Flash 8 skins does not scale, it is constructed the same way as the VolumeBar FLV Playback Custom UI component. For more information, see [“SeekBar and VolumeBar components” on page 527](#). The exception is that the handle is implemented differently. For more information on that, see the following section.

Handle

The SeekBar and VolumeBar handles are placed on the Stage next to the bar. By default, the handle’s left margin, right margin, and y -axis values are set by its position relative to the bar movie clip. The left margin is set by the difference between the handle’s x (horizontal) location and the bar’s x (horizontal) location, and the right margin is equal to the left margin. You can customize these values through ActionScript in the SeekBar or VolumeBar movie clip. The following example is the same ActionScript code that is used with the FLV Playback Custom UI components:

```
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

Beyond these properties, the handles are simple movie clips, set up the same way as they are in the FLV Playback Custom UI components. Both have rectangle backgrounds with the `alpha` property set to 0. These are present only to increase the hit region and are not required.

ActionScript

The seek bar and volume bar support the following additional properties:

Property	Description
<code>handle_mc:MovieClip</code>	The movie clip for the handle. Defaults to <code>seekBarHandle_mc</code> or <code>volumeBarHandle_mc</code>
<code>progress_mc:MovieClip</code>	The movie clip for progress. Defaults to <code>seekBarProgress_mc</code> or <code>volumeBarProgress_mc</code> .
<code>fullness_mc:MovieClip</code>	The movie clip for fullness. Defaults to <code>seekBarFullness</code> or <code>volumeBarFullness</code> .

Background and foreground clips

The movie clips `chrome_mc` and `forwardBackBorder_mc` are implemented as background clips.

Of the `ForwardBackBorder`, `ForwardBorder`, and `BackBorder` movie clips on the Stage and the placeholder `Forward` and `Back` buttons, the only one that is *not* on a guide layer is `ForwardBackBorder`. It is only in the skins that actually use the `Forward` and `Back` buttons.

No additional ActionScript is required to set up the background and foreground clips.

Modifying skin behavior

The `bufferingBarHidesAndDisablesOthers` property and the `skinAutoHide` property allow you to customize the behavior of your `FLVPlayback` skin.

Setting the `bufferingBarHidesAndDisablesOthers` property to `true` causes the `FLVPlayback` component to hide the `SeekBar` and its handle as well as disable the `Play` and `Pause` buttons when the component enters the buffering state. This can be useful when an `FLV` file is streaming from `FCS` over a slow connection with a high setting for the `bufferTime` property (10, for example). In this situation, an impatient user might try to start seeking by clicking the `Play` and `Pause` buttons, which could delay playing the file even longer. You can prevent this activity by setting `bufferingBarHidesAndDisablesOthers` to `true` and disabling the `SeekBar` element and the `Pause` and `Play` buttons while the component is in the buffering state.

The `skinAutoHide` property affects only predesigned skin `SWF` files and not controls created from the `FLV Playback Custom UI` components. If set to `true`, the `FLVPlayback` component hides the skin when the mouse is not over the viewing area. The default value of this property is `true`.

FLVPlayback class

Inheritance MovieClip > FLVPlayback class

ActionScript Class Name mx.video.FLVPlayback

FLVPlayback extends the MovieClip class and wraps a VideoPlayer object. For information on the VideoPlayer class, see [“VideoPlayer class” on page 706](#).

Unlike other components, the FLVPlayback component does not extend UIObject or UIComponent and, therefore, does not support the methods and properties of these classes. For example, you must call the MovieClip method, `attachMovie()`, rather than the UIObject method, `createClassObject()`, to create an instance of the component in ActionScript.

The methods and properties of the FLVPlayback class enable you to play and manipulate FLV files using the FLVPlayback component in your Flash application.

Setting a property of the FLVPlayback class with ActionScript overrides an equivalent parameter that initially set the property in the Property inspector or the Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. The following code shows the version in the Output panel:

```
trace(mx.video.FLVPlayback.version);
```

Method summary for the FLVPlayback class

The following table lists the methods of the FLVPlayback class:

Method	Description
<code>FLVPlayback.addASCuePoint()</code>	Adds an ActionScript cue point.
<code>FLVPlayback.addEventListener()</code>	Creates a listener for a specified event.
<code>FLVPlayback.bringVideoPlayerToFront()</code>	Brings a video player to the front of the stack of video players.
<code>FLVPlayback.closeVideoPlayer()</code>	Close NetStream for the video player with the specified index and deletes the video player.
<code>FLVPlayback.findCuePoint()</code>	Finds the specified type of cue point that has the specified time, the specified name, or the specified time and name.
<code>FLVPlayback.findNearestCuePoint()</code>	Finds the specified type of cue point at or near the given time or with the given name.

Method	Description
<code>FLVPlayback.findNextCuePointWithName()</code>	Finds the next cue point with the same name as a cue point returned by the <code>findCuePoint()</code> or <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.getVideoPlayer()</code>	Gets the video player specified by the <code>index</code> parameter.
<code>FLVPlayback.isFLVCuePointEnabled()</code>	Returns <code>false</code> if the FLV file embedded cue point is disabled by ActionScript.
<code>FLVPlayback.load()</code>	Begins loading the FLV file with the <code>autoPlay</code> property set to <code>false</code> .
<code>FLVPlayback.pause()</code>	Pauses playing the video stream.
<code>FLVPlayback.play()</code>	Begins playing the video stream and also allows loading and playing of a new FLV file.
<code>FLVPlayback.removeASCuePoint()</code>	Removes an ActionScript cue point.
<code>FLVPlayback.removeEventListener()</code>	Removes an event listener.
<code>FLVPlayback.seek()</code>	Seeks to a given time in the file, given in seconds, with decimal precision up to milliseconds.
<code>FLVPlayback.seekPercent()</code>	Seeks to a percentage of the way through the file.
<code>FLVPlayback.seekSeconds()</code>	Same as <code>FLVPlayback.seek()</code> .
<code>FLVPlayback.seekToNavCuePoint()</code>	Seeks to the navigation cue point with the given name at or after the specified time.
<code>FLVPlayback.seekToNextNavCuePoint()</code>	Seeks to the next navigation cue point, based on the specified time.
<code>FLVPlayback.seekToPrevNavCuePoint()</code>	Seeks to the previous navigation cue point, based on the specified time.
<code>FLVPlayback.setFLVCuePointEnabled()</code>	Enables or disables one or more FLV file cue points.
<code>FLVPlayback.setScale()</code>	Sets <code>scaleX</code> and <code>scaleY</code> simultaneously.
<code>FLVPlayback.setSize()</code>	Sets <code>width</code> and <code>height</code> simultaneously.
<code>FLVPlayback.stop()</code>	Stops playing the video stream.

Property summary for the FLVPlayback class

The FLVPlayback class has both class and instance properties.

FLVPlayback Class properties

The following properties occur only for the FLVPlayback class. They are read-only constants that apply to all instances of the FLVPlayback component in your application.

Property	Value	Description
<code>FLVPlayback.ACTIONSCRIPT</code>	"actionscript"	Can use as type parameter for <code>findCuePoint()</code> and <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.ALL</code>	"all"	Can use as type parameter for <code>findCuePoint()</code> and <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.BUFFERING</code>	"buffering"	A value for testing state property.
<code>FLVPlayback.CONNECTION_ERROR</code>	"connectionError"	A value for testing state property.
<code>FLVPlayback.DISCONNECTED</code>	"disconnected"	A value for testing state property.
<code>FLVPlayback.EVENT</code>	"event"	Can use as type parameter for <code>findCuePoint()</code> and <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.FLV</code>	"flv"	Can use as type parameter for <code>findCuePoint()</code> and <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.LOADING</code>	"loading"	A value for testing state property.
<code>FLVPlayback.NAVIGATION</code>	"navigation"	Can use as type parameter for <code>findCuePoint()</code> and <code>findNearestCuePoint()</code> methods.
<code>FLVPlayback.PAUSED</code>	"paused"	A value for testing state property.
<code>FLVPlayback.PLAYING</code>	"playing"	A value for testing state property.
<code>FLVPlayback.REWINDING</code>	"rewinding"	A value for testing state property.
<code>FLVPlayback.SEEKING</code>	"seeking"	A value for testing state property.
<code>FLVPlayback.STOPPED</code>	"stopped"	A value for testing state property.
<code>FLVPlayback.version</code>	A number that is the component's version number	To determine what version of the FLVPlayback component you are using.

Instance properties

The following table lists the instance properties of the FLVPlayback class. This set of properties applies to each instance of an FLVPlayback component in the application. For example, if you drag two instances of the FLVPlayback component to the Stage, each instance has a set of these properties.

Property	Description
<code>FLVPlayback.activeVideoPlayerIndex</code>	A number that specifies which FLV file stream is affected by other methods, properties, and events. Use this property to manage multiple FLV file streams; Default is 0.
<code>FLVPlayback.autoPlay</code>	A Boolean value that, if <code>true</code> , specifies that the component plays the FLV file immediately when it is loaded. Default is <code>true</code> .
<code>FLVPlayback.autoRewind</code>	A Boolean value that, if <code>true</code> , causes the FLV file to rewind to the Frame 1 when play stops.
<code>FLVPlayback.autoSize</code>	A Boolean value that, if <code>true</code> , causes the video to automatically size to the source dimensions.
<code>FLVPlayback.backButton</code>	A MovieClip object that is the backButton control.
<code>FLVPlayback.bitrate</code>	A number that specifies the user's bandwidth in bits per second. Used in some cases to decide which FLV file to play.
<code>FLVPlayback.buffering</code>	A Boolean value that is <code>true</code> if the video is in a buffering state. Read-only.
<code>FLVPlayback.bufferingBar</code>	A MovieClip object that is the bufferingBar control.
<code>FLVPlayback.bufferingBarHidesAndDisablesOthers</code>	Affects behavior of controls when the component enters the buffering state.
<code>FLVPlayback.bufferTime</code>	A number that specifies the number of seconds to buffer in memory before beginning playback of a video stream.

Property	Description
<code>FLVPlayback.bytesLoaded</code>	A number that indicates the extent of downloading in number of bytes for an HTTP download. Read only.
<code>FLVPlayback.bytesTotal</code>	A number that specifies the total number of bytes downloaded for an HTTP download. Read-only.
<code>FLVPlayback.contentPath</code>	A string that specifies the URL of an FLV file to load.
<code>FLVPlayback.cuePoints</code>	An array that describes ActionScript cue points and disabled embedded FLV file cue points. Should never be used directly with ActionScript. Should be set only through the Cue Points dialog box.
<code>FLVPlayback.forwardButton</code>	A MovieClip object that is the ForwardButton control.
<code>FLVPlayback.height</code>	A number that specifies the height of the video in pixels.
<code>FLVPlayback.idleTimeout</code>	The amount of time, in milliseconds, before Flash terminates an idle connection to the FCS because playing is paused or stopped.
<code>FLVPlayback.isLive</code>	A Boolean value that is <code>true</code> if the video stream is live.
<code>FLVPlayback.isRTMP</code>	A Boolean value that is <code>true</code> if the FLV file is streaming from an FCS or an FVSS. Read-only.
<code>FLVPlayback.maintainAspectRatio</code>	A Boolean value that, if <code>true</code> , maintains the video aspect ratio.
<code>FLVPlayback.metadata</code>	An object that is a metadata information packet that is received from a call to the <code>onMetaData()</code> callback function, if available. Read-only.
<code>FLVPlayback.metadataLoaded</code>	A Boolean value that is <code>true</code> if a metadata packet has been encountered and processed or if it is clear that it will not be processed. Read-only.
<code>FLVPlayback.muteButton</code>	A MovieClip object that is the MuteButton control.

Property	Description
<code>FLVPlayback.ncMgr</code>	An <code>INCManager</code> object that provides access to an instance of the class implementing <code>INCManager</code> . Read-only.
<code>FLVPlayback.pauseButton</code>	A <code>MovieClip</code> object that is the <code>PauseButton</code> control.
<code>FLVPlayback.paused</code>	A Boolean value that is <code>true</code> if the FLV file is in a paused state. Read-only.
<code>FLVPlayback.playButton</code>	A <code>MovieClip</code> object that is the <code>PlayButton</code> control.
<code>FLVPlayback.playheadPercentage</code>	A number that specifies playhead time as a percentage of the total FLV file duration.
<code>FLVPlayback.playheadTime</code>	A number that is the current playhead time or position, measured in seconds, which can be a fractional value.
<code>FLVPlayback.playheadUpdateInterval</code>	A number that is the amount of time, in milliseconds, between each <code>playheadUpdate</code> event.
<code>FLVPlayback.playing</code>	A Boolean value that is <code>true</code> if the FLV file is playing. Read-only.
<code>FLVPlayback.playPauseButton</code>	A <code>MovieClip</code> object that is the <code>PlayPauseButton</code> control.
<code>FLVPlayback.preferredHeight</code>	A number that specifies the height of the source FLV file.
<code>FLVPlayback.preferredWidth</code>	A number that specifies the width of the source FLV file.
<code>FLVPlayback.progressInterval</code>	A number that is the amount of time, in milliseconds, between each <code>progress</code> event.
<code>FLVPlayback.scaleX</code>	A number that specifies the horizontal scale.
<code>FLVPlayback.scaleY</code>	A number that specifies the vertical scale.
<code>FLVPlayback.scrubbing</code>	A Boolean value that is <code>true</code> if the user is dragging the <code>seekBar</code> handle. Read-only.

Property	Description
<code>FLVPlayback.seekBar</code>	A <code>MovieClip</code> object that is the <code>SeekBar</code> control.
<code>FLVPlayback.seekBarInterval</code>	A number that specifies, in milliseconds, how often to check the <code>SeekBar</code> handle when scrubbing. The default value is 250.
<code>FLVPlayback.seekBarScrubTolerance</code>	A number (percentage) that specifies how far a user can move the <code>scrubBar</code> handle before an update occurs. The value is specified as a percentage, ranging from 1 to 100.
<code>FLVPlayback.seekToPrevOffset</code>	A number, in seconds, that the <code>seekToPrevNavCuePoint()</code> method uses when it compares its time against the previous cue point.
<code>FLVPlayback.skin</code>	A string that specifies the name of a skin SWF file.
<code>FLVPlayback.skinAutoHide</code>	A Boolean value that, if <code>true</code> , hides the component skin when the mouse is not over the video. Defaults to <code>false</code> .
<code>FLVPlayback.state</code>	A string that specifies the state of the component. Set with the <code>load()</code> , <code>play()</code> , <code>stop()</code> , <code>pause()</code> and <code>seek()</code> methods. Read-only.
<code>FLVPlayback.stateResponsive</code>	A Boolean value that is <code>true</code> if the state is responsive. Read-only.
<code>FLVPlayback.stopButton</code>	A <code>MovieClip</code> object that is the <code>StopButton</code> control.
<code>FLVPlayback.stopped</code>	A Boolean value that is <code>true</code> if the state is stopped. Read-only.
<code>FLVPlayback.totalTime</code>	A number that is the total playing time for the video in seconds
<code>FLVPlayback.transform</code>	An object that provides direct access to the <code>Sound.setTransform()</code> and <code>Sound.getTransform()</code> methods to provide more sound control.
<code>FLVPlayback.visible</code>	A Boolean value that, if <code>true</code> , makes the <code>FLVPlayback</code> component visible.

Property	Description
<code>FLVPlayback.visibleVideoPlayerIndex</code>	A number that you can use to manage multiple FLV file streams. Sets which video player instance is visible and audible. Default is 0.
<code>FLVPlayback.volume</code>	A number in the range of 0 to 100 that indicates the volume control setting.
<code>FLVPlayback.volumeBar</code>	A <code>MovieClip</code> object that is the <code>VolumeBar</code> control.
<code>FLVPlayback.volumeBarInterval</code>	A number that specifies, in milliseconds, how often to check the <code>volumeBar</code> handle when scrubbing. The default value is 250.
<code>FLVPlayback.volumeBarScrubTolerance</code>	A number (percentage) that specifies how far a user can move the volume bar handle before an update occurs.
<code>FLVPlayback.width</code>	A number that specifies the width of the component instance in pixels.
<code>FLVPlayback.x</code>	A number that specifies the horizontal location of the video player in pixels.
<code>FLVPlayback.y</code>	A number that specifies the vertical location of the video player in pixels.

Event summary for the FLVPlayback class

The following table lists the events of the `FLVPlayback` class:

Event	Description
<code>FLVPlayback.buffering</code>	Dispatched when the buffering state is entered.
<code>FLVPlayback.close</code>	Dispatched when <code>NetConnection</code> is closed, whether through timeout or a call to the <code>close()</code> method.
<code>FLVPlayback.complete</code>	Dispatched when playing completes by reaching the end of the FLV file.
<code>FLVPlayback.cuePoint</code>	Dispatched when a cue point is reached.
<code>FLVPlayback.fastForward</code>	Dispatched when the location of the playhead is moved forward by a call to the <code>seek()</code> method.
<code>FLVPlayback.metadata</code>	Dispatched the first time the FLV file metadata is reached.

Event	Description
<code>FLVPlayback.paused</code>	Dispatched when the pause state is entered.
<code>FLVPlayback.playheadUpdate</code>	Dispatched every .25 seconds by default while the FLV file is playing. You can specify frequency with the <code>playheadUpdateInterval</code> property.
<code>FLVPlayback.playing</code>	Dispatched when the playing state is entered.
<code>FLVPlayback.progress</code>	Dispatched every .25 seconds, starting when the <code>load()</code> method is called and ending when all bytes are loaded or there is a network error. You can specify frequency with the <code>progressInterval</code> property.
<code>FLVPlayback.ready</code>	Dispatched when the FLV file is loaded and ready to display.
<code>FLVPlayback.resize</code>	Dispatched when the video is resized.
<code>FLVPlayback.rewind</code>	Dispatched when the location of the playhead is moved backward by a call to <code>seek()</code> or when the automatic rewind operation completes.
<code>FLVPlayback.scrubFinish</code>	Dispatched when the user stops scrubbing the timeline with the <code>SeekBar</code> .
<code>FLVPlayback.scrubStart</code>	Dispatched when user begins scrubbing the timeline with the <code>SeekBar</code> .
<code>FLVPlayback.seek</code>	Dispatched when the location of the playhead is changed by a call to <code>seek()</code> or by using the corresponding control.
<code>FLVPlayback.skinError</code>	Dispatched when an error occurs loading a skin SWF file.
<code>FLVPlayback.skinLoaded</code>	Dispatched when a skin SWF file is loaded.
<code>FLVPlayback.stateChange</code>	Dispatched when the playback state changes.
<code>FLVPlayback.stopped</code>	Dispatched when the stopped state is entered.
<code>FLVPlayback.volumeUpdate</code>	Dispatched when the volume is changed through the <code>volume</code> property.

FLVPlayback.ACTIONSCRIPT

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.ACTIONSCRIPT
```

Description

A read-only FLVPlayback class property that contains the string constant, "actionscript", for use as the type property with the `findCuePoint()` and `findNearestCuePoint()` methods.

Example

The following example uses the `ACTIONSCRIPT` constant to set the type property of the `findCuePoint()` method.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    var cuePt:Object = new Object(); // create cue point object
    cuePt.time = 2.444;
    cuePt.name = "ASCuePt1";
    my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
}
my_FLVPlybk.addEventListener("ready", listenerObject)
listenerObject.playing = function(eventObject:Object) {
    var rtn_obj:Object = new Object();
    if(rtn_obj = my_FLVPlybk.findCuePoint(2.444,
    FLVPlayback.ACTIONSCRIPT)){
        trace("Found cue point " + rtn_obj.name);
    }
}
my_FLVPlybk.addEventListener("playing", listenerObject)
```

See also

[FLVPlayback.findCuePoint\(\)](#)

FLVPlayback.activeVideoPlayerIndex

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.activeVideoPlayerIndex
```

Description

Property; a number that specifies which video player instance is affected by other APIs. Use this property to manage multiple FLV file streams. The default value is 0.

This property does not make the video player visible; use the `visibleVideoPlayerIndex` property to do that.

A new video player is created the first time `activeVideoPlayerIndex` is set to a number. When the new video player is created, its properties are set to the value of the default video player (`activeVideoPlayerIndex == 0`) except for `contentPath`, `totalTime`, and `isLive`, which are always set to the default values (empty string, 0, and `false`, respectively), and `autoPlay`, which is always `false` (the default is `true` only for the default video player, 0). The `cuePoints` property has no effect, as it would have no effect on a subsequent load into the default video player.

APIs that control volume, positioning, dimensions, visibility, and UI controls are always global, and their behavior is *not* affected by setting `activeVideoPlayerIndex`. Specifically, setting the `activeVideoPlayerIndex` property does not affect the following properties and methods.

Properties and Methods Not Affected by `activeVideoPlayerIndex`

<code>backButton</code>	<code>playPauseButton</code>	<code>skin</code>	<code>width</code>
<code>bufferingBar</code>	<code>scaleX</code>	<code>stopButton</code>	<code>x</code>
<code>bufferingBarHidesAndDisablesOthers</code>		<code>transform</code>	<code>y</code>
<code>forwardButton</code>	<code>scaleY</code>	<code>visible</code>	<code>setSize()</code>
<code>height</code>	<code>seekBar</code>	<code>volume</code>	<code>setScale()</code>
<code>muteButton</code>	<code>seekBarInterval</code>	<code>volumeBar</code>	
<code>pauseButton</code>	<code>seekBarScrubTolerance</code>	<code>volumeBarInterval</code>	
<code>playButton</code>	<code>seekToPrevOffset</code>	<code>volumeBarScrubTolerance</code>	

NOTE

The `visibleVideoPlayerIndex` property, *not* the `activeVideoPlayerIndex` property, determines which video player the skin controls.

APIs that control dimensions do interact with the `visibleVideoPlayerIndex` property, however. For more information, see “[FLVPlayback.visibleVideoPlayerIndex](#)” on page 688.

The remaining APIs target a specific video player based on the setting of `activeVideoPlayerIndex`.

When listening for events, you get all events for all video players. To distinguish which video player the event is for, use the event’s `vp` property, a number corresponding to the number set in `activeVideoPlayerIndex` and `visibleVideoPlayerIndex`. All events have this property *except* for `resize` and `volume`, which are not specific to a video player but are global for the `FLVPlayback` instance.

For example, to load a second FLV file in the background, set `activeVideoPlayerIndex` to 1 and call the `load()` method. When you are ready to show this FLV file and hide the first one, set `visibleVideoPlayerIndex` to 1.

Example

The following example creates two video players to play two FLV files consecutively in a single `FLVPlayback` instance. It sets the `activeVideoPlayerIndex` property to switch between the video players and their respective FLV files.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
// specify name and location of FLV for default player
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    // add a second video player and specify the name and loc of its FLV
    my_FLVPlybk.activeVideoPlayerIndex = 1;
    my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
      water.flv";
    // reset to default video player, which plays its FLV automatically
    my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
    // if complete is for 2nd FLV, make default active and visible
    if (eventObject.vp == 1) {
        my_FLVPlybk.activeVideoPlayerIndex = 0;
        my_FLVPlybk.visibleVideoPlayerIndex = 0;
    }
    else { // make 2nd player active & visible and play FLV
        my_FLVPlybk.activeVideoPlayerIndex = 1;
        my_FLVPlybk.visibleVideoPlayerIndex = 1;
        my_FLVPlybk.play();
    }
};
// add listener for complete event
my_FLVPlybk.addEventListener("complete", listenerObject);
```

See also

[FLVPlayback.bringVideoPlayerToFront\(\)](#), [FLVPlayback.getVideoPlayer\(\)](#),
[VideoPlayer](#) class, [FLVPlayback.visibleVideoPlayerIndex](#),

FLVPlayback.addASCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.addASCuePoint(cuePoint:Object)  
my_FLVplybk.addASCuePoint(time:Number, name:String[, parameters:Object])
```

Parameters

cuePoint An object having *name:String* and *time:Number* (in seconds) properties, which describe the cue point. It also might have a *parameters:Object* property that holds name/value pairs. It may have *type:String* set to "actionscript". If *type* is missing or set to something else, it is set automatically. If the object does not conform to these conventions, the method throws a `VideoError` error.

time A number that is the time for the new cue point to be added. If you use the *time* parameter, the *name* parameter must follow.

name A string that specifies the name of the cue point if you submit a *time* parameter instead of the *CuePoint* object.

parameters Optional parameters for the cue point.

Returns

A copy of the cue point object that was added with the following additional properties:

- *array* The array of cue points that were searched. Treat this array as read-only, because adding, removing, or editing objects within it can cause cue points to malfunction.
- *index* The index into the array for the returned cue point.

Description

Method; adds an ActionScript cue point and has the same effect as adding an ActionScript cue point using the Cue Points dialog box, except that it occurs when an application executes rather than during application development.

Cue point information is wiped out when the `contentPath` property is set. To set cue point information for the next FLV file to be loaded, set the `contentPath` property first.

It is valid to add multiple ActionScript cue points with the same name and time. When you remove ActionScript cue points with the `removeASCuePoint()` method, all cue points with the same name and time are removed.

Example

The following example adds two ActionScript cue points to an FLV file. The example adds the first one using a *CuePoint* parameter and the second one using the *time* and *name* parameters. When each cue point occurs while playing, a listener for cuePoint events shows the value of the `playheadTime` property in a text area.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Drag a `TextArea` component to the Stage below the `FLVPlayback` instance, and give it an instance name of `my_ta`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - TextArea component on the Stage with an instance name of my_ta
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
my_ta.visible = false;
// create cue point object
var cuePt:Object = new Object(); // create cue point object
cuePt.time = 2.444;
cuePt.name = "ASCuePt1";
cuePt.type = "actionscript";
my_FLVPlayback.addASCuePoint(cuePt); //add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlayback.addASCuePoint(5, "ASCuePt2");
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    my_ta.text = "Elapsed time in seconds: " + my_FLVPlayback.playheadTime;
    my_ta.visible = true;
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.removeASCuePoint\(\)](#)

FLVPlayback.addEventListener()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage:

```
my_FLVPlayback.addEventListener(event:String, listener:Object):Void  
my_FLVPlayback.addEventListener(event:String, listener:Function):Void
```

Parameters

event A string that specifies the name of the event for which you are registering a listener. If the listener is an object, this is also the name of the listener object function to call.

listener The name of the listener object or function that you are registering for the event.

Returns

Nothing.

Description

Method; registers a listener object or function for a specified event. If the listener is an object, the object must have a function defined for it with the same name as the event. If the listener is a function, it is the name of the function that will be called to handle the event.

Example

The following example listens for a `complete` event and displays a message in a text area when it occurs.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Drag a `TextArea` component to the Stage below the `FLVPlayback` instance, and give it an instance name of `my_ta`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
Usage 1:: listener object  
/**  
  Requires:  
  - FLVPlayback component on the Stage with an instance name of my_FLVPlayback  
  - TextArea component on the Stage with an instance name of my_ta  
*/  
import mx.video.*;  
my_ta.visible = false;  
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/  
  water.flv";  
var listenerObject:Object = new Object(); // create listener object  
listenerObject.complete = function(eventObject:Object):Void {  
    my_ta.text = "That's All Folks!";  
    my_ta.visible = true;  
};  
my_FLVPlayback.addEventListener("complete", listenerObject);
```

Usage 2: listener function

```
/**  
  Requires:  
  - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk  
  - TextArea component on the Stage with an instance name of my_ta  
*/  
import mx.video.*;  
my_ta.visible = false;  
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/  
  water.flv";  
function the_end(eventObject:Object):Void {  
    my_ta.text = "That's All Folks!";  
    my_ta.visible = true;  
};  
my_FLVPlaybk.addEventListener("complete", the_end);
```

See also

[Event summary for the FLVPlayback class](#), [FLVPlayback.removeEventListener\(\)](#),

FLVPlayback.ALL

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.ALL
```

Description

A read-only FLVPlayback class property that contains the string constant, "all". You can use this property as the type parameter for the `findCuePoint()` and `findNearestCuePoint()` methods.

Example

The following example looks among all cue points for a cue point named `point2` that has a time of 7.748. The example shows the `type` and `time` properties for the cue point that was found.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
// create cue point object
var listenerObject = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    var cuePt:Object = new Object(); // create cue point object
    cuePt.name = "point2";
    cuePt.time = 7.748;
    if(cuePt = my_FLVPlayback.findCuePoint(cuePt, FLVPlayback.ALL)) //find cue
    point
        trace("found a " + cuePt.type + " cue point at " + cuePt.time);
    else
        trace("cue point not found");
}
my_FLVPlayback.addEventListener("ready", listenerObject);
```

See also

[FLVPlayback.findCuePoint\(\)](#)

FLVPlayback.autoPlay

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.autoPlay
```

Description

Property; a Boolean value that, if set to `true`, causes the FLV file to play immediately when the `contentPath` property is set. If set to `false`, the component waits for the play command. Even if `autoPlay` is set to `false`, the component loads the content immediately. The default value is `true`.

If you set the property to `true` between the loading of new FLV files, it has no effect until `contentPath` is set.

Example

The following example disables the FLV file from playing to set the playhead 30 percent into the playing time and begin playing at that point.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.autoPlay = false;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlayback.seekPercent(30);
    my_FLVPlayback.play();
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#)

FLVPlayback.autoRewind

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.autoRewind`

Description

Property; a Boolean value that, if `true`, causes the FLV file to rewind to Frame 1 when play stops, either because the player reached the end of the stream or the `stop()` method was called. This property is meaningless for live streams. The default value is `true`.

Example

The following example sets the `autoRewind` property to `false` to prevent the FLV file from automatically rewinding when it finishes playing.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component in the Library
 */
my_FLVPlayback.autoRewind = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

FLVPlayback.autoSize

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.autoSize
```

Description

Property; a Boolean value that, if `true`, causes the video to size automatically to the dimensions of the source FLV file. If this property is set from `false` to `true` after an FLV file has been loaded, the automatic resizing starts immediately. The default value is `false`.

Example

The following example first show the source dimensions of the FLV file (`preferredWidth` and `preferredHeight`) when the FLV file is ready to play. Then it calls the `setSize()` to change the dimensions of the `FLVPlayback` instance, triggering a `resize` event. Next, it sets the `autoSize` property to `true`, triggering another `resize` event that restores the size to the dimensions of the source FLV file. The `resize` event handler displays the dimensions of the `FLVPlayback` instance in the Output panel after each event.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component in the Library
 */
import mx.video.*;
my_FLVPlayback.maintainAspectRatio = false;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    trace("FLV width is: " + my_FLVPlayback.preferredWidth + " FLV height is: "
        + my_FLVPlayback.preferredHeight);
    my_FLVPlayback.setSize(400, 400);
    my_FLVPlayback.autoSize = true;
};
my_FLVPlayback.addEventListener("ready", listenerObject);
listenerObject.resize = function(eventObject:Object) {
    trace("my_FLVPlayback width is: " + my_FLVPlayback.width + "
        my_FLVPlayback.height is: " + my_FLVPlayback.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.maintainAspectRatio](#), [FLVPlayback.preferredHeight](#),
[FLVPlayback.preferredWidth](#), [FLVPlayback.resize](#)

FLVPlayback.backButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.backButton`

Description

Property; a MovieClip object that is the BackButton playback control. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Clicking the BackButton control causes a `rewind` event.

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, and `stopButton` properties to attach individual FLV Playback Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback` and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Playback Custom UI components and give them the instance names shown in parentheses: BackButton (`my_bkbbtn`), ForwardButton (`my_fwdbbtn`), PlayButton (`my_plybbtn`), PauseButton (`my_pausbbtn`), and StopButton (`my_stopbbtn`). Then add the following lines of code to the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and
 *   StopButton components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbbtn;
my_FLVPlayback.forwardButton = my_fwdbbtn;
my_FLVPlayback.playButton = my_plybbtn;
my_FLVPlayback.pauseButton = my_pausbbtn;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.forwardButton](#), [FLVPlayback.rewind](#)

FLVPlayback.bitrate

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVplybk.bitrate

Description

Property; a number that specifies the bits per second at which to transfer the FLV file.

When doing a progressive download, you can use the SMIL format, but you must set the bit rate because there is no automatic detection.

When streaming video from a FCS, you can provide a SMIL file that describes how to switch between multiple streams based on bandwidth. Bandwidth is automatically detected by FCS, and, in this case, *bitrate* is ignored, if it is set.

For more information on using a SMIL file, see [“Using a SMIL file” on page 712](#).

Example

The following example checks two radio buttons to determine what bit rate to use when selecting an FLV file from the specified SMIL file. The relevant `video` tags in the SMIL file are shown in the following code:

```
<switch>
  <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">
  <video src="myvideo_isdn.flv" dur="3:00.1">
</switch>
```

For a low-speed connection, the code sets the `bitrate` property to force selection of the appropriate FLV file. For higher speed connections, it takes advantage of automatic bandwidth detection for streaming from FCS and does not set the bit rate.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Drag a RadioButton component and then a Label component to the Library panel. Then add the following code to the Actions panel on Frame 1 of the Timeline. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of your SMIL file.

```
/**
 * Requires:
 * - FLVPlayback component in the Library
 * - RadioButton component in the Library
 * - Label component in the Library
 */
import mx.video.*;
import mx.controls.*;

this.createClassObject(Label, "my_prompt", 10);
my_prompt.text = "Please indicate your connection speed: ";
this.createClassObject(RadioButton, "dialup", 20, {label:"Dialup modem (56
    KB)", groupName:"radioGroup"});
this.createClassObject(RadioButton, "isdn", 30, {label:"ISDN (128 KB)",
    groupName:"radioGroup"});
my_prompt.autoSize = "left";
dialup.setSize(200, 30);
isdn.setSize(200, 30);
// Position RadioButtons on Stage.
my_prompt.move(my_FLVPlybk.x, my_FLVPlybk.y + my_FLVPlybk.height + 40);
dialup.move(my_FLVPlybk.x, my_prompt.y + my_prompt.height + 5);
isdn.move(my_FLVPlybk.x, dialup.y + 15);

// Create listener object
var rbListener:Object = new Object();
rbListener.click = function(eventObject:Object){
    if(dialup.selected) { // for modem
        my_FLVPlybk.bitrate = 56000;
    } // for isdn (or higher bandwidths) allow automatic detection
}
// Add listener
radioGroup.addEventListener("click", rbListener);
my_FLVPlybk.contentPath = "http://www.someserver.com/video/sample.smil";
```

FLVPlayback.bringVideoPlayerToFront()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.bringVideoPlayerToFront(index:Number)
```

Parameters

index A number that is the index of the video player to move to the front

Description

Method; brings a video player to the front of the stack of video players. Useful for custom transitions between video players. The stack order is the same as it is for the `activeVideoPlayerIndex` property: 0 is on the bottom, 1 is above it, 2 is above 1, and so on.

Example

The following example uses two video players to play two FLV files. When each of the three cue points in the first FLV file (`cuepoints.flv`) occurs, the example calls the `bringVideoPlayerToFront()` method to bring the other FLV file to the front. Because the example sets the `_alpha` property to 75 for video player number 1, the FLV file (`plane_cuepoints`) playing in that player is transparent, making both FLV files visible simultaneously when that FLV file is in front.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
my_FLVPlybk.load("http://www.helpexamples.com/flash/video/cuepoints.flv");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    if (eventObject.target.contentPath == "http://www.helpexamples.com/
flash/video/cuepoints.flv") {
        //this fires after the first flv is ready
        my_FLVPlybk.activeVideoPlayerIndex = 1;
        my_FLVPlybk.load("http://www.helpexamples.com/flash/video/
plane_cuepoints.flv");
    } else {
        //this fires after the second flv is ready
        eventObject.target.activeVideoPlayerIndex = 0;
        eventObject.target.play();
        eventObject.target.activeVideoPlayerIndex = 1;
        eventObject.target.play();
        var layerOnTop:MovieClip = eventObject.target.getVideoPlayer(1);
        layerOnTop._alpha = 75;
        layerOnTop._visible = true;
    }
}
my_FLVPlybk.addEventListener("ready", listenerObject);

var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {

    //based upon the cue name, bring one or the other to the front
    if (eventObject.info.name == "point1") {
        trace(eventObject.info.name + " : 0 to front");
        eventObject.target.bringVideoPlayerToFront(1);
    } else if (eventObject.info.name == "point2") {
        trace(eventObject.info.name + " : 1 to front");
        eventObject.target.bringVideoPlayerToFront(0);
    } else if (eventObject.info.name == "point3") {
        trace(eventObject.info.name + " : 0 to front");
        eventObject.target.bringVideoPlayerToFront(1);
    }
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.getVideoPlayer\(\)](#), [VideoPlayer](#) class, [FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.buffering

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.buffering = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("buffering", listenerObject);
```

Description

Event; dispatched when the FLVPlayback instance enters the buffering state. The FLVPlayback instance typically enters this immediately after a call to the `play()` method or when the Play control is clicked, before entering the playing state. The event object has the properties `state`, `playheadTime`, and `vp`, which is the index number of the video player to which the event applies. See [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The FLVPlayback instance also dispatches the `stateChange` event when buffering begins.

Example

The following example creates a listener for the buffering event. When a buffering event occurs, the event handler calls the `trace()` method to display the values of the `state` and `vp` properties.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.buffering = function(eventObject:Object) {
    trace("The state property has a value of " + eventObject.target.state);
    trace("The video player number is: " + eventObject.vp);
};
my_FLVPlayback.addEventListener("buffering", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#), [FLVPlayback.state](#),
[FLVPlayback.removeEventListener\(\)](#)

FLVPlayback.BUFFERING

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.BUFFERING
```

Description

Property; a read-only FLVPlayback class property that contains the string constant, "buffering". You can compare this property to the state property to determine whether the component is in the buffering state.

Example

The following example displays the value of the FLVPlayback.BUFFERING property when the component enters a buffering state.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.BUFFERING)
        trace(FLVPlayback.BUFFERING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.buffering

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.buffering

Description

Property; a Boolean value that is `true` if the video is in a buffering state. Read-only.

Example

The following example creates a listener for the `buffering` event and displays a message in the Output panel when the event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
    if(my_FLVPlybk.buffering){
        trace("The video is buffering");
    }
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.buffering](#), [FLVPlayback.BUFFERING](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.bufferingBar

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlybk.bufferingBar`

Description

Property; a MovieClip object that is the buffering bar control. This control displays when the FLV file is in a loading or buffering state. See [“Skinning FLV Playback Custom UI components individually” on page 525](#) for more information on using FLV Playback Custom UI components for playback controls.

Example

The following example attaches individual FLV Playback Custom UI controls to an FLVPlayback component by setting the following properties: playPauseButton, stopButton, backButton, forwardButton, and bufferingBar. The buffering bar appears only while the FLV file is buffering before it begins to play.

Drag an FLVPlayback component to the Stage, give it an instance name of **my_FLVPlaybk**, and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Playback Custom UI components and give them the instance names shown in parentheses: PlayPauseButton (**my_bkbtn**), StopButton (**my_stopbtn**), BackButton (**my_bkbtn**), ForwardButton (**my_fwdbtn**), and BufferingBar (**my_buffrbar**). Then add the following lines of code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 * - FLV Playback Custom UI PlayPauseButton, StopButton, BackButton,
 *   ForwardButton, BufferingBar components in the Library
 */
import mx.video.*;
my_FLVPlaybk.playPauseButton = my_plypausbbtn;
my_FLVPlaybk.stopButton = my_stopbtn;
my_FLVPlaybk.backButton = my_bkbtn;
my_FLVPlaybk.forwardButton = my_fwdbtn;
my_FLVPlaybk.bufferingBar = my_buffrbar;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#)

FLVPlayback.bufferingBarHidesAndDisablesOthers

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.bufferingBarHidesAndDisablesOthers
```

Description

Property; if set to `true`, hides the `SeekBar` control and disables the `Play`, `Pause`, `PlayPause`, `BackButton` and `ForwardButton` controls while the FLV file is in the buffering state. This can be useful to prevent a user from using these controls to try to speed up playing the FLV file when it is downloading or streaming over a slow connection.

Example

The following example assumes playing a streaming FLV file from a FCS or FVSS. It sets the `bufferingBarHidesAndDisablesOthers` property to disable the `Play`, `Pause`, `PlayPause`, `BackButton`, and `ForwardButton` controls and to hide the `SeekBar` control while the FLV file is buffering.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of an FLV file on your FCS.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.bufferTime = 15;
my_FLVPlayback.bufferingBarHidesAndDisablesOthers = true;
my_FLVPlayback.contentPath = "rtmp://host_name/somefolder/vid_name.flv";
```

See also

[FLVPlayback.bufferingBar](#)

FLVPlayback.bufferTime

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.bufferTime

Description

Property; a number that specifies the number of seconds to buffer in memory before beginning to play a video stream. For FLV files streaming over RTMP, which are not downloaded and buffer only in memory, it can be important to increase this setting from the default value of 0.1. For a progressively downloaded FLV file over HTTP, there is little benefit to increasing this value although it could improve viewing a high-quality video on an older, slower computer.

NOTE

This property does not specify the amount of the FLV file to download before starting playback.

Example

The following example sets the buffer time to 8 seconds for an FLV file streaming from a FCS.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to Frame 1 of the Timeline in the Actions panel. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of an FLV file on your FCS.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.bufferTime = 8;
my_FLVPlayback.contentPath = "rtmp://host_name/somefolder/vid_name.flv";
```

FLVPlayback.bytesLoaded

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.bytesLoaded

Description

Property; a number that indicates the extent of downloading, in number of bytes, for an HTTP download. Returns -1 when there is no stream, when the stream is from a FCS, or if the information is not yet available. The returned value is useful only for an HTTP download. Read-only.

Example

The following example shows the initial value of the bytesLoaded property and its value when the ready event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    trace("State is " + eventObject.state + "; ready to play");
    // display the no. of bytes loaded at this point
    trace("Bytes loaded: " + my_FLVPlayback.bytesLoaded);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
trace("Bytes loaded: " + my_FLVPlayback.bytesLoaded); // -1 if loading not
begun
```

FLVPlayback.bytesTotal

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.bytesTotal
```

Description

Property; a number that specifies the total number of bytes downloaded for an HTTP download. Returns -1 when there is no stream, when the stream is from a FCS, or if the information is not yet available. The returned value is useful only for an HTTP download.

Read-only.

Example

The following example uses the `bytesTotal` property to display the number of bytes being loaded for an HTTP download. When the `metadataReceived` event occurs, the event handler displays this value in the text area `my_ta`.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Drag a `TextArea` component to the Stage below the `FLVPlayback` instance and give it an instance name of `my_ta`. Then add the following code to Frame 1 of the Timeline in the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - TextArea component on the Stage with an instance name of my_ta
 */
import mx.video.*;
my_ta.visible = false;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
    my_ta.text = "Loading: " + my_FLVPlayback.bytesTotal + " bytes.";
    my_ta.visible = true;
};
my_FLVPlayback.addEventListener("metadataReceived", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

FLVPlayback.close

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("close", listenerObject);
```

Description

Event; the FLVPlayback instance dispatches this event when it closes the NetConnection, by timing out or through a call to the `closeVideoPlayer()` method, or when you call the `load()` method or the `play()` method or set `contentPath` and cause the RTMP connection to close as a result. The FLVPlayback instance dispatches this event only when streaming from FCS or FVSS. The event object has the properties `state` and `playheadTime`.

Event has property `vp`, which is the index number of the video player to which this event applies.

Example

The following example assumes playing a streaming FLV file from a FCS or FVSS. When the FLV file completes, a listener for the `complete` event sets the `contentPath` property to the location of a new FLV file, which triggers a `close` event on the RTMP connection for the first FLV file. The listener for the `close` event displays the index number of the video player for which the event occurred.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVplybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of an FLV file on your FCS.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
// listen for close event on RTMP connection; display index of video player
listenerObject.close = function(eventObject:Object) {
```



```
        trace("Closed connection for video player: " + eventObject.vp);
    };
    my_FLVPlayback.addListener("close", listenerObject);
    // listen for complete event; play new FLV
    listenerObject.complete = function(eventObject:Object) {
        if (my_FLVPlayback.contentPath != "http://www.helpexamples.com/flash/video/
        water.flv") {
            my_FLVPlayback.play("http://www.helpexamples.com/flash/video/water.flv");
        }
    };
    my_FLVPlayback.addListener("complete", listenerObject);
    my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.closeVideoPlayer\(\)](#),
[FLVPlayback.contentPath](#), [FLVPlayback.load\(\)](#), [FLVPlayback.play\(\)](#),
[FLVPlayback.visibleVideoPlayerIndex](#),

FLVPlayback.closeVideoPlayer()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.closeVideoPlayer(index:Number)
```

Parameters

index A number that is the index of the video player to close.

Returns

The `VideoPlayer` object that closed.

Description

Method; closes `NetStream` and deletes the video player specified by the *index* parameter. If the closed video player is the active or visible video player, the `FLVPlayback` instance sets the active and or visible video player to the default player (with index 0). You cannot close the default player, and trying to do so causes the component to throw an error.

Example

The following example creates two video players to play two FLV files consecutively in a single FLVPlayback instance. When the second FLV file finishes, the event handler for the complete event calls the closeVideoPlayer() method to close the second player. If you click the Play button to play the FLV files a second time, you see that the video player for the second player is gone, which causes the component to throw an error (VideoError) and show a message that says the FLVPlayback instance cannot find the FLV file.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlaybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 */
// specify name and location of FLV for default player
import mx.video.*;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    // add a second video player and specify the name and loc of its FLV
    my_FLVPlaybk.activeVideoPlayerIndex = 1;
    my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
      water.flv";
    // reset to default video player, which plays its FLV automatically
    my_FLVPlaybk.activeVideoPlayerIndex = 0;
};
my_FLVPlaybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
    // if complete is for 2nd FLV, make default active and visible
    if (eventObject.vp == 1) {
        my_FLVPlaybk.activeVideoPlayerIndex = 0;
        my_FLVPlaybk.visibleVideoPlayerIndex = 0;
        my_FLVPlaybk.closeVideoPlayer(1); // close 2nd video player
    } else { // make 2nd player active & visible and play FLV
        my_FLVPlaybk.activeVideoPlayerIndex = 1;
        my_FLVPlaybk.visibleVideoPlayerIndex = 1;
        my_FLVPlaybk.play();
    }
};
// add listener for complete event
my_FLVPlaybk.addEventListener("complete", listenerObject);
```

See also

[FLVPlayback.close](#), [FLVPlayback.activeVideoPlayerIndex](#),
[FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.complete

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("complete", listenerObject);
```

Description

Event; dispatched when playing completes because the player reached the end of the FLV file. The component does not dispatch the event if you call the `stop()` or `pause()` methods or click the corresponding controls. The event object has the properties `state` and `playheadTime`.

When the application uses progressive download, does not set the `totalTime` property explicitly, and downloads an FLV file that does not specify the duration in the metadata, the video player sets `totalTime` to an approximate total value before it dispatches this event.

The video player also dispatches the `stateChange` and `stopped` events.

Example

The following example uses the `playheadTime` property to show the elapsed playing time of the FLV file in the Output panel when the `complete` event occurs.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVplybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
    trace("Elapsed play time at completion is: " + my_FLVplybk.playheadTime);
};
my_FLVplybk.addEventListener("complete", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.state](#), [FLVPlayback.stateChange](#), [FLVPlayback.stopped](#), [FLVPlayback.totalTime](#)

FLVPlayback.CONNECTION_ERROR

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.CONNECTION_ERROR
```

Description

A read-only FLVPlayback class property that contains the string constant "connectionError". You can compare this property to the state property to determine if a connection error state has occurred.

Example

The following example forces a connection error by specifying an invalid FLV filename (nosuch.flv) in the contentPath property. The example uses the CONNECTION_ERROR property to detect the error in a listener for the stateChange event.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  no_such.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(my_FLVPlybk.state == FLVPlayback.CONNECTION_ERROR)
        trace("State: " + FLVPlayback.CONNECTION_ERROR);
}
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.contentPath

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.contentPath
```

Description

Property; a string that specifies the URL of the FLV file to stream and how to stream it. The URL can be an HTTP URL to an FLV file, an RTMP URL to a stream, or an HTTP URL to an XML file.

If you set this property through the Component inspector or the Property inspector, the FLV file begins loading and playing at the next `MovieClip.onEnterFrame` event. The delay provides time to set the `isLive`, `autoPlay`, and `cuePoints` properties, among others, which affect loading. It also allows ActionScript that is placed on the first Frame to affect the FLVPlayback component before it starts playing.

If you set this property through ActionScript, the FLVPlayback instance closes the current FLV file and immediately begins loading the new FLV file. The `autoPlay`, `totalTime`, and `isLive` properties affect how the new FLV file is loaded, so if you set these properties, you must set them *before* setting `contentPath` property.

Set the `autoPlay` property to `false` to prevent the new FLV file from playing automatically.

Example

The following example sets the `contentPath` property in ActionScript to specify the location of the FLV file to play.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. *Do not* assign a value to the `contentPath` parameter in the Component inspector. Add the following code to Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
    my_FLVPlayback.setSize(my_FLVPlayback.preferredWidth,
    my_FLVPlayback.preferredHeight);
}
my_FLVPlayback.addEventListener("metadataReceived", listenerObject);
```

See also

[FLVPlayback.autoPlay](#), [FLVPlayback.isLive](#), [FLVPlayback.play\(\)](#),
[FLVPlayback.totalTime](#)

FLVPlayback.cuePoint

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```

Description

Event; dispatched when a cue point is reached. The event object has an `info` property that contains the `info` object received by the `NetStream.onCuePoint` callback for FLV file cue points. For ActionScript cue points, it contains the object that was passed into the ActionScript cue point methods or properties.

This event has a `vp` property, which is the index number of the video player to which this event applies.

Example

The following example adds two ActionScript cue points to an FLV file. The example adds the first one using a *cuePoint* parameter and the second one using the *time* and *name* parameters. When each cue point occurs, a listener for *cuePoint* events shows the value of the *playheadTime* property in a text area.

Drag an *FLVPlayback* component to the Stage, and give it an instance name of **my_FLVPlybk**. Drag a *TextArea* component to the Stage below the *FLVPlayback* instance, and give it an instance name of **my_ta**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 * - TextArea component on the Stage with an instance name of my_ta
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
my_ta.visible = false;
// create cue point object
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 1.444;
cuePt.name = "elapsed_time";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlybk.addASCuePoint(5.3, "elapsed_time2");
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    my_ta.text = "Cue at: " + eventObject.info.time + " occurred";
    my_ta.visible = true;
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.cuePoints

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.cuePoints`

Description

Property; an array that describes ActionScript cue points and disabled embedded FLV file cue points. This property is created specifically for use by the Component inspector and does not work if it is set any other way. Its value has an effect only on the first FLV file loaded and only if it is loaded by setting the `contentPath` property in the Component inspector or the Property inspector.

NOTE

This property is not accessible in ActionScript. To access cue point information in ActionScript, use the `metadata` property.

To add, remove, enable, or disable cue points with ActionScript, use `addASCuePoint()`, `removeASCuePoint()`, or `setFLVCuePointEnabled()`.

See also

`FLVPlayback.contentPath`, `FLVPlayback.addASCuePoint()`,
`FLVPlayback.findCuePoint()`, `FLVPlayback.findNearestCuePoint()`,
`FLVPlayback.findNextCuePointWithName()`, `FLVPlayback.isFLVCuePointEnabled()`,
`FLVPlayback.metadata`, `FLVPlayback.metadataReceived`,
`FLVPlayback.removeASCuePoint()`, `FLVPlayback.seekToNavCuePoint()`,
`FLVPlayback.seekToNextNavCuePoint()`, `FLVPlayback.seekToPrevNavCuePoint()`,
`FLVPlayback.setFLVCuePointEnabled()`

FLVPlayback.DISCONNECTED

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.DISCONNECTED
```

Description

A read-only FLVPlayback class property that contains the string constant "disconnected". You can compare this property to the `state` property to determine if a disconnected state exists.

The FLVPlayback instance is in a disconnected state until you set the `contentPath` property.

Example

The following example simply shows a message in the Output panel that confirms that the FLVPlayback instance is in a disconnected state before setting the `contentPath` property.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;

if(my_FLVPlybk.state == FLVPlayback.DISCONNECTED)
    trace("FLVPlayback instance is currently disconnected");
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
```

See also

[FLVPlayback.contentPath](#), [FLVPlayback.state](#)

FLVPlayback.EVENT

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.EVENT
```

Description

A read-only FLVPlayback class property that contains the String constant, "event". You can use this property as the type parameter for the `findCuePoint()` and `findNearestCuePoint()` methods.

Example

The following example uses the `FLVPlayback.EVENT` property to specify that it wants to find a cue point named `myCue` that is of the `event` type. It shows the returned cue point name, time, and type properties, and the `cuePoint` listener displays "Hit it!" in the Output panel when the cue point occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  plane_cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    if(rtn_cuePt = my_FLVPlybk.findCuePoint("myCue", FLVPlayback.EVENT)){
        trace("Cue point name is: " + rtn_cuePt.name);
        trace("Cue point time is: " + rtn_cuePt.time);
        trace("Cue point type is: " + rtn_cuePt.type);
    }
}
```

```
my_FLVPlayback.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    if(eventObject.info.name == "myCue")
        trace("Hit it!");
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.findCuePoint\(\)](#)

FLVPlayback.fastForward

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.fastForward = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVPlayback.addEventListener("fastForward", listenerObject);
```

Event; dispatched when the location of the playhead moves forward by seeking, either manually or through ActionScript, or by clicking the ForwardButton control. The event object has the properties `state`, `playheadTime`, and `vp`. The `playheadTime` property reflects the destination time, and the `vp` property is the index number of the video player to which the event applies.

The FLVPlayback instance also dispatches the `seek` and `playheadUpdate` events.

Example

The following example catches occurrences of the `fastForward` event as it occurs and shows the elapsed playhead time in the Output panel. When the `ready` event occurs, a call to the `seekPercent()` method triggers the `fastForward` event.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlayback.seekPercent(35);
};
my_FLVPlayback.addEventListener("ready", listenerObject);

listenerObject.fastForward = function(eventObject:Object):Void {
    trace("fastforward event; playhead time is: " +
        eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("fastForward", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.forwardButton](#),
[FLVPlayback.seek](#), [FLVPlayback.seek\(\)](#), [FLVPlayback.seekBar](#),
[FLVPlayback.seekPercent\(\)](#), [FLVPlayback.seekSeconds\(\)](#),
[FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToNextNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.seekToPrevOffset](#),
[FLVPlayback.state](#), [FLVPlayback.playheadTime](#)

FLVPlayback.findCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.findCuePoint(time:Number[, type:String]):Object
my_FLVPlayback.findCuePoint(name:String[, type:String]):Object
my_FLVPlayback.findCuePoint(cuePoint:Object[, type:String]):Object
```

Parameters

time A number that is the time, in seconds, of the cue point for which to search. The method uses only the first three decimal places and rounds any additional decimal places that you provide. The method returns the cue point that matches this time. If multiple ActionScript cue points have the same time, the method returns only the name that is first in alphabetical order. It returns `null` if it does not find a match.

name A string that is the name of the cue point for which to search. The method returns the first cue point that matches this name, or it returns `null` if it does not find a match.

cuePoint An object that is a cue point object containing *time* and *name* properties for which to search. If the *name* property has no value or is `null`, the search behaves as if the parameter is a number representing the time for which to search. If the *time* property has no value, is `null`, or is less than zero, then the search behaves as if the parameter is a string containing a name for which to search. If you provide both the *time* and *name* properties and a cue point exists that matches those values, the method returns it. Otherwise, the method returns `null`.

type Optional. A string that specifies the type of cue point for which to search. The possible values for this parameter are: "actionscript", "all", "event", "flv", or "navigation". You can specify these values using the following class properties: `FLVPlayback.ACTIONSCRIPT`, `FLVPlayback.ALL`, `FLVPlayback.EVENT`, `FLVPlayback.FLV`, and `FLVPlayback.NAVIGATION`. If this parameter is not specified, the default is "all", which means the method will search all cue point types.

Returns

An object that is a copy of the found cue point object, with the following additional properties:

- array* The array of cue points that were searched. Treat this array as read-only because adding, removing, or editing objects within it can cause cue points to malfunction.
- index* The index into the array for the returned cue point.

Returns `null` if no match is found.

Description

Method; finds the cue point of the type specified by the *type* parameter and having the time, name, or combination of time and name that you specify through the parameters.

If you do not provide a value for either the time or name of the cue point, or if the time is `null`, undefined, or less than zero *and* the name is `null` or undefined, the method throws `VideoError` error 1002. For more information, see [“VideoError class” on page 698](#).

The method includes disabled cue points in the search. Use the `isFLVCuePointEnabled()` method to determine if a cue point is disabled.

Example

The following example adds two ActionScript cue points to an FLV file, and then calls the `findCuePoint()` method three times. The first call looks for a navigation cue point with a time of 7.748. The second call looks for cue point "ASCue1", using only a name value. The third call uses a cue point object that specifies both a time, 10, and a name, "ASCue2". After the third call, the example shows the content of the returned cue point object, including the array of cue points that were searched, which, in this example, were ActionScript cue points.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
// create cue point object
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var cuePt:Object = new Object();
var rtn_cuePt:Object = new Object(); // create object for return value
cuePt.time = 4.444;
cuePt.name = "ASCue1";
my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlybk.addASCuePoint(10, "ASCue2");
// find navigation cue point using time
rtn_cuePt = my_FLVPlybk.findCuePoint(7.748, FLVPlayback.NAVIGATION);
//find actionscript cue point using name only
rtn_cuePt = my_FLVPlybk.findCuePoint("ASCue1");
//find actionscript cue point using cue point object
cuePt.time = 10;
cuePt.name = "ASCue2";
rtn_cuePt = my_FLVPlybk.findCuePoint(cuePt, FLVPlayback.ACTIONSCRIPT);
// see what returned cue point object contains
for (i in rtn_cuePt) {
    //if an array object, open it up and trace contents
    if (typeof rtn_cuePt[i] == "object") {
        tracer(rtn_cuePt[i]);
    } else trace name (i) and value (rtn_cuePt[i]) pair
    } else {
        trace(i+ " " + rtn_cuePt[i]);
    }
}
}
```

```

// trace array of cue points
function tracer(cuepts:Array) {
    for (i in cuepts) {
        if (typeof cuepts[i] == "object") { //if object in object
            tracer(cuepts[i]); //trace object
        } else {
            // trace the name : value pair
            trace(i + " " + cuepts[i]);
        }
    }
}
}

```

See also

[FLVPlayback.addASCuePoint\(\)](#), [FLVPlayback.cuePoints](#),
[FLVPlayback.findNearestCuePoint\(\)](#), [FLVPlayback.findNextCuePointWithName\(\)](#),
[FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.removeASCuePoint\(\)](#),
[FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToNextNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.setFLVCuePointEnabled\(\)](#)

FLVPlayback.findNearestCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```

my_FLVplybk.findNearestCuePoint(time:Number[, type:String]):Object
my_FLVplybk.findNearestCuePoint(name:String[, type:String]):Object
my_FLVplybk.findNearestCuePoint(cuePoint:Object[, type:String]):Object

```

Parameters

time A number that is the time, in seconds, of the cue point for which to search. The method uses only the first three decimal places and rounds any additional decimal places that you provide. The method returns the cue point that matches this time or the nearest *earlier* cue point of the specified type. If multiple cue points have the same time, which can only occur with ActionScript cue points, the method returns only the name that is first in alphabetical order. It returns `null` if it does not find a match.

name A string that is the name of the cue point for which to search. The method returns the first cue point that matches this name or `null`, if it does not find a match.

cuePoint An object that is a cue point object containing `time` and `name` properties for which to search. If you provide a time and the `name` property has no value or is `null`, the search behaves as if the parameter is a number representing the time for which to search. If you provide a name and the `time` property has no value, is `null`, or is less than zero, the search behaves as if the parameter is a string containing a name for which to search. If you provide both the `time` and `name` properties and a cue point exists that matches those values, the method returns it. If it does not find a match for both the time and name, it returns the first cue point that matches the name and has an *earlier* time. If there is no earlier cue point with that name, it returns the first cue point that matches that name. Otherwise, the method returns `null`.

type Optional. A string that specifies the type of cuepoint for which to search. The possible values for this parameter are: "actionscript", "all", "event", "flv", or "navigation". You can specify these values using the following class properties: `FLVPlayback.ACTIONSCRIPT`, `FLVPlayback.ALL`, `FLVPlayback.EVENT`, `FLVPlayback.FLV`, and `FLVPlayback.NAVIGATION`. If this parameter is not specified, the default is "all", which means the method will search all cue point types.

Returns

An object that is a copy of the found cue point object with the following additional properties:

- `array` The array of cue points searched. Treat this array as read-only as adding, removing or editing objects within it can cause cue points to malfunction.
- `index` The index into the array for the returned cue point.

Returns `null` if no match was found.

Description

Method; finds a cue point of the specified type that matches *or is earlier* than the time that you specify, or that matches the name that you specify, if you specify both a time and a name and no earlier cue point matches that name. Otherwise, it returns `null`.

The method includes disabled cue points in the search. Use the `isFLVCuePointEnabled()` method to determine if a cue point is disabled.

If the time is `null`, undefined, or less than 0 *and* the name is `null` or undefined, the method throws a `VideoError` error (1002).

Example

The following example creates an ActionScript cue point for the FLV file at 4.07 seconds. When this cue point occurs, the `cuePoint` event handler calls the `findNearestCuePoint()` method to find a cue point of any type that is nearest to 5 seconds later. The Output panel shows the name, time, and type of the cue point that is returned.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 4.07;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
function cuePoint(eventObject:Object):Void {
    if (eventObject.info.name == "ASpt1") {
        var rtn_obj:Object = new Object();
        rtn_obj = my_FLVPlybk.findNearestCuePoint(eventObject.info.time + 5);
        trace("Cue point name is: " + rtn_obj.name);
        trace("Cue point time is: " + rtn_obj.time);
        trace("Cue point type is: " + rtn_obj.type);
    }
}
my_FLVPlybk.addEventListener("cuePoint", cuePoint);
```

See also

[FLVPlayback.addASCuePoint\(\)](#), [FLVPlayback.cuePoints](#),
[FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.findNextCuePointWithName\(\)](#),
[FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.removeASCuePoint\(\)](#),
[FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToNextNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.setFLVCuePointEnabled\(\)](#)

FLVPlayback.findNextCuePointWithName()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.findNextCuePointWithName(my_cuePoint:Object)
```

Parameters

my_cuePoint A cue point object that has been returned by either the `findCuePoint()` method, the `findNearestCuePoint()` method, or a previous call to this method.

Returns

An object that is a copy of the found cue point object with the following additional properties:

- `array` The array of cue points searched. Treat this array as read-only because adding, removing or editing objects within it can cause cue points to malfunction.
- `index` The index into the array for the returned cue point.

Returns `null` if no match was found.

Description

Method; finds the next cue point in `my_cuePoint.array` that has the same name as `my_cuePoint.name`. The `my_cuePoint` object must be a cue point object that has been returned by the `findCuePoint()` method, the `findNearestCuePoint()` method, or a previous call to this method. This method uses the `array` property that these methods add to the cue point object.

The method includes disabled cue points in the search. Use the `isFLVCuePointEnabled()` method to determine if a cue point is disabled.

Returns `null` if there are no more cue points in the array with a matching name.

Example

The following example creates three ActionScript cue points with the name "transition". When the `ready` event occurs, the event handler calls the `findCuePoint()` method to find the first cue point with this name. If it finds a match, it calls the `findNextName()` function, which calls the `findNextCuePointWithName()` method, passing the returned cue point object (`rtn_obj`), to find any additional cue points with the same name.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 6.27;
cuePt.name = "transition";
cuePt.type = "actionscript";
my_FLVPlayback.addASCuePoint(cuePt); //add AS cue point
cuePt.time = 7.06;
my_FLVPlayback.addASCuePoint(cuePt); //add AS cue point
cuePt.time = 11.13;
my_FLVPlayback.addASCuePoint(cuePt); //add AS cue point
var listenerObject:Object = new Object();
listenerObject.ready = function():Void {
    var rtn_obj:Object = new Object();
    // Find cue point using name string
    if (rtn_obj = my_FLVPlayback.findCuePoint("transition")) {
        trace("Cue point name is: " + rtn_obj.name);
        trace("Cue point time is: " + rtn_obj.time);
        trace("Cue point type is: " + rtn_obj.type);
        findNextName(rtn_obj);
    }
}
my_FLVPlayback.addEventListener("ready", listenerObject);
// Find additional cue points with the same name
function findNextName(cuePt:Object):Void {
    while(cuePt = my_FLVPlayback.findNextCuePointWithName(cuePt)) {
        trace("Cue point name is: " + cuePt.name);
        trace("Cue point time is: " + cuePt.time);
        trace("Cue point type is: " + cuePt.type);
    }
}
```

See also

[FLVPlayback.addASCuePoint\(\)](#), [FLVPlayback.cuePoints](#),
[FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.findNearestCuePoint\(\)](#),
[FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.removeASCuePoint\(\)](#),
[FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToNextNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.setFLVCuePointEnabled\(\)](#)

FLVPlayback.FLV

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`mx.video.FLVPlayback.FLV`

Description

A read-only FLVPlayback class property that contains the string constant, "flv". You can use this property as the `type` parameter for the `findCuePoint()` and `findNearestCuePoint()` methods.

Example

The following example looks for a cue point named `point2` with a time of 7.748 among FLV file cue points and displays the type and time found. FLV file cue points are navigation and event cue points.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
// create cue point object
var listenerObject = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    var cuePt:Object = new Object(); // create cue point object
    cuePt.name = "point3";
    cuePt.time = 16.02;
    if(cuePt = my_FLVPlybk.findCuePoint(cuePt, FLVPlayback.FLV)) //find cue
    point
        trace("found a " + cuePt.type + " cue point at " + cuePt.time);
    else
        trace("cue point not found");
}
my_FLVPlybk.addEventListener("ready", listenerObject);
```

See also

[FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.findNearestCuePoint\(\)](#)

FLVPlayback.forwardButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.forwardButton
```

Description

Property; a MovieClip object that is the Forward button control. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, and `stopButton` properties to attach individual FLV Playback Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback` and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Custom UI components and give them the instance names shown in parentheses: `BackButton` (`my_bkbbtn`), `ForwardButton` (`my_fwdbbtn`), `PlayButton` (`my_plybbtn`), `PauseButton` (`my_pausbbtn`), and `StopButton` (`my_stopbbtn`). Then add the following lines of code to the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and
 *   StopButton components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbbtn;
my_FLVPlayback.forwardButton = my_fwdbbtn;
my_FLVPlayback.playButton = my_plybbtn;
my_FLVPlayback.pauseButton = my_pausbbtn;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.fastForward](#), [FLVPlayback.seek](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.getVideoPlayer()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.getVideoPlayer(index:Number)
```

Returns

A VideoPlayer object.

Description

Method; gets the video player specified by *index*. When possible, it is best to access VideoPlayer methods and properties using FLVPlayback methods and properties. Each VideoPlayer._name property is its index.

Example

The following example uses two video players to play two FLV files. When the second FLV file triggers the `ready` event, the example calls the `getVideoPlayer()` method to obtain video player number 1 and set its `_alpha` property to 50. This causes the FLV file (`plane_cuepoints`) in that player to be transparent and makes both FLV files visible simultaneously.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVplybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
my_FLVplybk.load("http://www.helpexamples.com/flash/video/cuepoints.flv");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    if (eventObject.target.contentPath == "http://www.helpexamples.com/
    flash/video/cuepoints.flv") {
```

```

        //this fires after the first flv is ready
        my_FLVPlayback.activeVideoPlayerIndex = 1;
        my_FLVPlayback.load("http://www.helpexamples.com/flash/video/
plane_cuepoints.flv");
    } else {
        //this fires after the second flv is ready
        eventObject.target.activeVideoPlayerIndex = 0;
        eventObject.target.play();
        eventObject.target.activeVideoPlayerIndex = 1;
        eventObject.target.play();
        var layerOnTop:MovieClip = eventObject.target.getVideoPlayer(1);
        layerOnTop._alpha = 50;
        layerOnTop._visible = true;
    }
}
my_FLVPlayback.addEventListener("ready", listenerObject);

```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.bringVideoPlayerToFront\(\)](#), [FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.height

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.height

Description

Property; a number that specifies the height of the FLVPlayback instance. This property affects only the height of the FLVPlayback instance and does not include the height of a skin SWF file that might be loaded. Use the FLVPlayback height property and not the MovieClip._height property because the _height property might give a different value if a skin SWF file is loaded.

Example

The following example sets the width and height properties to change the size of the video player. It first sets the maintainAspectRatio property to false to prevent the video player from resizing automatically when the dimensions change.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.maintainAspectRatio = false;
my_FLVPlayback.width = 300;
my_FLVPlayback.height = 350;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.preferredHeight](#), [FLVPlayback.preferredWidth](#),
[FLVPlayback.maintainAspectRatio](#), [FLVPlayback.resize](#), [FLVPlayback.setSize\(\)](#),
[FLVPlayback.width](#)

FLVPlayback.idleTimeout

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.idleTimeout

Description

Property; the amount of time, in milliseconds, before Flash terminates an idle connection to a FCS because playing paused or stopped. This property has no effect on an FLV file downloading over HTTP.

If this property is set when a video stream is already idle, it restarts the timeout period with the new value.

The default value is 300,000, or 5 minutes.

Example

The following example assumes playing a streaming FLV file from a FCS or FVSS. The example sets the `idleTimeout` property to a low value of 10 milliseconds, which triggers a timeout and, consequently, a `close` event on the RTMP connection. The listener for the `close` event shows the index number of the video player for which the event occurred.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. In the Component inspector, assign the `contentPath` parameter a value that specifies the location of a streaming FLV file from a FCS or FVSS. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.idleTimeout = 10;
var listenerObject:Object = new Object();
//listen for close event on RTMP connection; display index of video player
listenerObject.close = function(eventObject:Object) {
    trace("Closed connection for video player: " + eventObject.vp);
};
my_FLVPlybk.addEventListener("close", listenerObject);
```

See also

[FLVPlayback.close](#)

FLVPlayback.isFLVCuePointEnabled()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.isFLVCuePointEnabled(time:Number)
my_FLVplybk.isFLVCuePointEnabled(name:String)
my_FLVplybk.isFLVCuePointEnabled(cuePoint:Object)
```

Parameters

time A number that is the time, in seconds, of the cue point for which to search.

name A string that is the name of the cue point for which to search.

cuePoint A cue point object with `time` and `name` properties for the cue point. The method does not check any other properties on the incoming cue point object. If `time` or `name` is undefined, the method uses only the property that is defined.

Returns

A Boolean value that is `false` if the cue point or cue points are found and are disabled, and `true` if the cue point is not disabled or does not exist. If the time given is undefined, `null`, less than 0, or only a cue point name is provided, the method returns `false` only if all cue points with this name are disabled.

Description

Method; returns `false` if the FLV file embedded cue point is disabled. You can disable cue points either by setting the `cuePoints` property through the Flash Video Cue Points dialog box or by calling the `setFLVCuePointEnabled()` method.

The return value from this function is meaningful only when the `metadataLoaded` property is `true`, the `metadata` property is not `null`, or after a `metadataReceived` event. When `metadataLoaded` is `false`, this function always returns `true`.

Example

The following example disables the `point2` cue point when the `ready` event occurs. When the first `cuePoint` event occurs, the event handler calls the `isFLVCuePointEnabled()` method to see if the cue point is disabled and, if so, the event handler enables it. The FLV file contains the following embedded cue points: `point1, 00:00:00:418; point2, 00:00:07.748; point3, 00:00:16:020.`

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
function ready(eventObject:Object) {
    my_FLVPlybk.setFLVCuePointEnabled(false, "point2");
}
```

```

my_FLVPlayback.addEventListener("ready", ready);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
    trace("Elapsed time in seconds: " + my_FLVPlayback.playheadTime);
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point type is: " + eventObject.info.type);
    if (my_FLVPlayback.isFLVCuePointEnabled("point2") == false) {
        my_FLVPlayback.setFLVCuePointEnabled(true, "point2");
    }
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";

```

See also

[FLVPlayback.cuePoint](#), [FLVPlayback.findCuePoint\(\)](#),
[FLVPlayback.findNearestCuePoint\(\)](#), [FLVPlayback.findNextCuePointWithName\(\)](#),
[FLVPlayback.setFLVCuePointEnabled\(\)](#), [FLVPlayback.seekToNavCuePoint\(\)](#),
[FLVPlayback.seekToNextNavCuePoint\(\)](#), [FLVPlayback.seekToPrevNavCuePoint\(\)](#)

FLVPlayback.isLive

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.isLive

Description

Property; a Boolean value that is `true` if the video stream is live. This property is effective only when streaming from a FCS or FVSS. The value of this property is ignored for an HTTP download.

If you set this property between loading new FLV files, it has no effect until the `contentPath` parameter is set for the new FLV file.

Example

The following example assumes playing a live stream from a FCS. When the playing event occurs, the example shows the value of the `isLive` property.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of an FLV file on your FCS.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playing = function(eventObject:Object) {
    trace("The isLive property is " + my_FLVPlayback.isLive);
};
my_FLVPlayback.addEventListener("playing", listenerObject);
my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

See also

[FLVPlayback.contentPath](#), [FLVPlayback.load\(\)](#)

FLVPlayback.isRTMP

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.isRTMP

Description

Property; a Boolean value that is `true` if the FLV file is streaming from a FCS or FVSS using RTMP. Its value is `false` for any other FLV file source. Read-only.

Example

The following example assumes playing a streaming FLV file from a FCS or FVSS. When the `playing` event occurs, the example shows the value of the `isRTMP` property to indicate whether the FLV file is coming from an RTMP URL.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline. In the statement that loads the `contentPath` property, replace the italicized text with the name and location of an FLV file on your FCS.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.bufferTime = 7;
var listenerObject:Object = new Object();
// listen for playing event on RTMP connection; display result of isRTMP
listenerObject.playing = function(eventObject:Object) {
    trace("Value of isRTMP property is: " + my_FLVPlayback.isRTMP);
};
my_FLVPlayback.addEventListener("playing", listenerObject);
my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

See also

[FLVPlayback.contentPath](#), [FLVPlayback.load\(\)](#), [FLVPlayback.play\(\)](#)

FLVPlayback.load()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.load(contentPath:String[, totalTime:Number, isLive:Boolean])
```

Parameters

contentPath A string that specifies the URL of the FLV file to stream and how to stream it. The URL can be a local path, an HTTP URL to an FLV file, an RTMP URL to an FLV file stream, or an HTTP URL to an XML file.

totalTime A number that is the total playing time for the video. Optional.

isLive A Boolean value that is `true` if the video stream is live. This value is effective only when streaming from FVSS or FCS. The value of this property is ignored for an HTTP download. Optional.

Returns

Nothing.

Description

Method; begins loading the FLV file and provides a shortcut for setting the `autoPlay` property to `false` and setting the `contentPath`, `totalTime`, and `isLive` properties, if given. If the `totalTime` and `isLive` properties are undefined, they are not set. If the `contentPath` property is undefined, `null`, or an empty string, this method does nothing.

Example

The following example calls the `load()` method to load an FLV file that is specified by the `contentPath` parameter. It shows the value of the `autoPlay` property before and after loading the FLV file and calls the `play()` method to begin playing the FLV file.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
trace("Before load, autoPlay is: " + my_FLVPlayback.autoPlay);
my_FLVPlayback.load("http://www.helpexamples.com/flash/video/water.flv");
trace("After load, autoPlay is: " + my_FLVPlayback.autoPlay);
my_FLVPlayback.play();
```

See also

[FLVPlayback.contentPath](#), [FLVPlayback.isLive](#), [FLVPlayback.totalTime](#)

FLVPlayback.LOADING

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.LOADING
```

Description

A read-only FLVPlayback class property that contains the string constant, "loading". You can compare this property to the `state` property to determine whether the component is in the loading state.

Example

The following example displays the value of the `FLVPlayback.LOADING` property if the FLV file is in the loading state.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlaybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(my_FLVPlaybk.state == FLVPlayback.LOADING)
        trace("State is " + FLVPlayback.LOADING);
}
my_FLVPlaybk.addEventListener("stateChange", listenerObject);
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.maintainAspectRatio

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlaybk.maintainAspectRatio
```

Description

Property; a Boolean value that, if `true`, maintains the video aspect ratio. If this property is changed from `false` to `true` and the `autoSize` property is `false` after an FLV file has been loaded, an automatic resize of the video starts immediately. The default value is `true`.

Example

The following example calls the `setSize()` method to change the size of the `FLVPlayback` instance, causing a `resize` event. The `maintainAspectRatio` property, which defaults to `true`, forces a second `resize` event to maintain the aspect ratio. The `resize` event handler displays the width and height of the resized `FLVPlayback` instance for both occurrences in the Output panel. If you set `maintainAspectRatio` to `false`, the dimensions specified by the `setSize()` method take effect.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
// maintainAspectRatio defaults to true, causing resize when size changes.
// Remove the comment delimiters from the following line to disable resize.

// my_FLVPlybk.maintainAspectRatio = false;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object) {
    trace("resize event; Width is: " + eventObject.target.width + " Height
    is: " + eventObject.target.height);
};
my_FLVPlybk.addEventListener("resize", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
my_FLVPlybk.setSize(300, 300);
```

See also

[FLVPlayback.autoSize](#), [FLVPlayback.height](#), [FLVPlayback.preferredHeight](#), [FLVPlayback.preferredWidth](#), [FLVPlayback.resize](#), [FLVPlayback.setSize\(\)](#), [FLVPlayback.width](#)

FLVPlayback.metadata

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.metadata
```

Description

Property; an object that is a metadata information packet that is received from a call to the `NetStream.onMetaData()` callback function, if available. Read only.

If the FLV file is encoded with the Flash 8 encoder, the `metadata` property contains the following information. Older FLV files contain only the `height`, `width`, and `duration` values.

Parameter	Description
<code>canSeekToEnd</code>	A Boolean value that is <code>true</code> if the FLV file is encoded with a keyframe on the last frame that allows seeking to the end of a progressive download movie clip. It is <code>false</code> if the FLV file is not encoded with a keyframe on the last frame.
<code>cuePoints</code>	An array of objects, one for each cue point embedded in the FLV file. Value is undefined if the FLV file does not contain any cue points. Each object has the following properties: <ul style="list-style-type: none">• <code>type</code> a string that specifies the type of cue point as either "navigation" or "event".• <code>name</code> a string that is the name of the cue point.• <code>time</code> a number that is the time of the cue point in seconds with a precision of three decimal places (milliseconds).• <code>parameters</code> an optional object that has name-value pairs that are designated by the user when creating the cue points.
<code>audiocodecid</code>	A number that indicates the audio codec (code/decode technique) that was used.
<code>audiodelay</code>	A number that indicates what time in the FLV file "time 0" of the original FLV file exists. The video content needs to be delayed by a small amount to properly synchronize the audio.
<code>audiodatarate</code>	A number that is the kilobytes per second of audio.
<code>videocodecid</code>	A number that is the codec version that was used to encode the video.

Parameter	Description
framerate	A number that is the frame rate of the FLV file.
videodatarate	A number that is the video data rate of the FLV file.
height	A number that is the height of the FLV file.
width	A number that is the width of the FLV file.
duration	A number that specifies the duration of the FLV file in seconds.

Example

The following example shows in the Output panel a sampling of metadata values from the FLV file `cuepoints.flv`. It displays the data when the `metadataReceived` event occurs.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
    trace("canSeekToEnd is " + my_FLVPlybk.metadata.canSeekToEnd);
    trace("Number of cue points is " +
        my_FLVPlybk.metadata.cuePoints.length);
    trace("Frame rate is " + my_FLVPlybk.metadata.framerate);
    trace("Height is " + my_FLVPlybk.metadata.height);
    trace("Width is " + my_FLVPlybk.metadata.width);
    trace("Duration is " + my_FLVPlybk.metadata.duration + " seconds");
};
my_FLVPlybk.addEventListener("metadataReceived", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
```

See also

[FLVPlayback.metadataLoaded](#), [FLVPlayback.metadataReceived](#)

FLVPlayback.metadataLoaded

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.metadataLoaded`

Description

Property; a Boolean value that is `true` if a metadata packet has been encountered and processed *or* if the FLV file was encoded without the metadata packet. In other words, the value is `true` if the metadata is received, or if you are never going to get any metadata. So, you know if you have the metadata; and if you don't have the metadata, you know not to wait around for it. If you just want to know whether or not you have metadata, you can check the value with:

```
FLVPlayback.metadata != null
```

Use this property to check whether you can retrieve useful information with the methods for finding and enabling or disabling cue points. Read-only.

Example

The following example creates a listener for the `progress` event. When the event occurs, the example checks whether the `metadataLoaded` property is `true` and, if so, shows the metadata values `height`, `width`, and `duration` in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
    if(my_FLVPlayback.metadataLoaded){
        trace("Height is " + my_FLVPlayback.metadata.height);
        trace("Width is " + my_FLVPlayback.metadata.width);
        trace("Duration is " + my_FLVPlayback.metadata.duration + " seconds");
    }
};
my_FLVPlayback.addEventListener("progress", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.metadata](#), [FLVPlayback.metadataReceived](#)

FLVPlayback.metadataReceived

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("metadataReceived", listenerObject);
```

Description

Event; dispatched the first time the FLV file metadata is reached. The event object has an `info` property that contains the `info` object received by the `NetStream.onMetaData` callback.

The event also has the `vp` property, which is the index number of the video player to which the event applies. For more information, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

Example

The following example creates a listener for the `metadataReceived` event. When the event occurs, the event handler sends the name, time, and type of each cue point that is described in the `metadata` property to the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
    var i:Number = 0;
    trace("This FLV contains the following cue points:");
    while(i < my_FLVPlybk.metadata.cuePoints.length) {
        trace("\nName: " + my_FLVPlybk.metadata.cuePoints[i].name);
        trace(" Time: " + my_FLVPlybk.metadata.cuePoints[i].time);
        trace(" Type is " + my_FLVPlybk.metadata.cuePoints[i].type);
        ++i;
    }
};
my_FLVPlybk.addEventListener("metadataReceived", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.metadata](#), [FLVPlayback.metadataLoaded](#)

FLVPlayback.muteButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.muteButton
```

Description

Property; a MovieClip object that is the mute button control. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Clicking the muteButton control dispatches a `volumeUpdate` event.

Example

The following example uses the `backButton`, `forwardButton`, `playPauseButton`, `stopButton`, and `muteButton` properties to attach individual FLV Custom UI controls to an `FLVPlayback` component.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlaybk` and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Custom UI components and give them the instance names shown in parentheses: `BackButton` (`my_bkbbtn`), `ForwardButton` (`my_fwdbbtn`), `PlayPauseButton` (`my_plypausbbtn`), `StopButton` (`my_stopbbtn`), and `MuteButton` (`my_mutebbtn`). Then add the following lines of code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 * - FLV Custom UI BackButton, ForwardButton, PlayPauseButton, StopButton,
 *   and MuteButton components in the Library
 */
import mx.video.*;
my_FLVPlaybk.backButton = my_bkbbtn;
my_FLVPlaybk.forwardButton = my_fwdbbtn;
my_FLVPlaybk.playPauseButton = my_plypausbbtn;
my_FLVPlaybk.stopButton = my_stopbbtn;
my_FLVPlaybk.muteButton = my_mutebbtn;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.skin](#), [FLVPlayback.volume](#), [FLVPlayback.volumeBar](#),
[FLVPlayback.volumeUpdate](#)

FLVPlayback.NAVIGATION

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.NAVIGATION
```

Description

A read-only `FLVPlayback` class property that contains the string constant, "navigation". You can use this property for the `type` parameter of the `findCuePoint()` and `findNearestCuePoint()` methods.

Example

The following example uses the `FLVPlayback.NAVIGATION` property to specify the type of cue point to find.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlaybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 */
import mx.video.*;
// find navigation cue point using time
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    var rtn_cuePt:Object = new Object();
    rtn_cuePt = my_FLVPlaybk.findCuePoint(7.748, FLVPlayback.NAVIGATION);
    trace("Found cue point at " + rtn_cuePt.time + " of type " +
        rtn_cuePt.type);
}
my_FLVPlaybk.addEventListener("ready", listenerObject)
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
```

See also

[FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.findNearestCuePoint\(\)](#)

FLVPlayback.ncMgr

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlaybk.ncMgr`

Description

Property; an INCManager object that provides access to an instance of the class implementing INCManager, which is an interface to the NCManager class.

You can use this property to implement a custom INCManager that requires custom initialization. Read-only.

Example

The following example shows the value of the NetConnection DEFAULT_TIMEOUT property when the ready event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
// specify name and location of FLV
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    var NC:Object = new Object();
    NC = my_FLVPlybk.ncMgr;
    trace("Net connection timeout is " + NC.DEFAULT_TIMEOUT + "
      milliseconds");
};
my_FLVPlybk.addEventListener("ready", listenerObject);
```

See also

[VideoPlayer class](#)

FLVPlayback.pause()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.pause()
```


Parameters

None.

Returns

Nothing.

Description

Method; pauses playing the video stream.

Example

The following example creates a listener for the `playheadUpdate` event. When it occurs, the event handler checks to see whether the playhead time is between 5 and 5.05 seconds. If it is, the event handler calls the `pause()` method to suspend playing the FLV file. The paused event handler prompts you to push the Play button to continue.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Drag a `TextArea` component to the Stage below the `FLVPlayback` instance, and give it an instance name of `my_ta`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_ta.visible = false;
my_FLVPlayback.playheadUpdateInterval = 5;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    if ((eventObject.playheadTime >= 5) && (eventObject.playheadTime < 5.05))
    {
        my_FLVPlayback.pause();
    }
};
my_FLVPlayback.addEventListener("playheadUpdate", listenerObject);
listenerObject.paused = function(eventObject:Object):Void {
    my_ta.text = "Paused; push Play to continue";
    my_ta.visible = true;
};
my_FLVPlayback.addEventListener("paused", listenerObject);
```

See also

[FLVPlayback.paused](#), [FLVPlayback.play\(\)](#), [FLVPlayback.rewind](#)

FLVPlayback.pauseButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.pauseButton
```

Description

Property; a MovieClip that is the PauseButton control. For more information on using the FLV Playback Custom UI components for playback control, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, and `stopButton` properties to attach individual FLV Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback` and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Custom UI components and give them the instance names shown in parentheses: `BackButton` (`my_bkbbtn`), `ForwardButton` (`my_fwdbbtn`), `PlayButton` (`my_plybbtn`), `PauseButton` (`my_pausbbtn`), and `StopButton` (`my_stopbbtn`). Then add the following lines of code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and
 *   StopButton components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbbtn;
my_FLVPlayback.forwardButton = my_fwdbbtn;
my_FLVPlayback.playButton = my_plybbtn;
my_FLVPlayback.pauseButton = my_pausbbtn;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.playButton](#), [FLVPlayback.playPauseButton](#), [FLVPlayback.skin](#)

FLVPlayback.PAUSED

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.PAUSED
```

Description

A read-only FLVPlayback class property that contains the string constant, "paused". You can compare this property to the `state` property to see if the component is in the paused state.

Example

The following example uses the `FLVPlayback.PAUSED` property to show the state of the FLV file when the user clicks the Pause button.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.PAUSED)
        trace("FLV is " + FLVPlayback.PAUSED);
}
my_FLVPlybk.addEventListener("stateChange", listenerObject)
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.paused

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.paused = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("paused", listenerObject);
```

Description

Event; dispatched when the player enters the paused state. This happens when you call the `pause()` method or click the corresponding control and it also happens in some cases when the FLV file is loaded if `autoPlay` is `false` (the state may be stopped instead). The event object has the properties `state`, `playheadTime`, and `vp`, which is the index number of the video player to which this event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The `stateChange` event is also dispatched.

Example

The following example creates a listener for the `playheadUpdate` event. When the event occurs, the event handler checks to see whether the `playheadTime` property is between 5 and 5.05 seconds. If so, the event handler calls the `pause()` method to suspend playing the FLV file. This triggers a `paused` event for which the `paused` event handler shows, “The FLV is paused!”

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.playheadUpdateInterval = 5;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    if ((eventObject.playheadTime >= 5) && (eventObject.playheadTime < 5.05))
    {
        my_FLVPlybk.pause();
    }
}
my_FLVPlybk.addEventListener("playheadUpdate", listenerObject);
listenerObject.paused = function(eventObject:Object) {
    trace("FLV is " + my_FLVPlybk.state + "!");
};
my_FLVPlybk.addEventListener("paused", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.pause\(\)](#), [FLVPlayback.paused](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.paused

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.paused

Description

Property; a Boolean value that is `true` if the FLV file is in a paused state. Read-only.

Example

The following example creates a listener for the `stateChange` event. When the event occurs, it checks the `paused` property to determine whether the component is in the paused state. If so, it shows a message to that effect in the Output panel. You must click the Pause button while the FLV file is playing to cause the paused state to occur.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object) {
    if(my_FLVPlayback.paused)
        trace("FLV is in " + FLVPlayback.PAUSED + " state");
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.paused](#), [FLVPlayback.PAUSED](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.play()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.play ([contentPath:String, totalTime:Number, isLive:Boolean])
```

Parameters

contentPath A string that specifies the URL of the FLV file to stream and how to stream it. The URL can be a local path, an HTTP URL to an FLV file, an RTMP URL to an FLV file stream, or an HTTP URL to an XML file. It is optional, but the `contentPath` property must be set either through the Component inspector or through ActionScript, or this method has no effect.

totalTime A number that is the total playing time for the video. Optional.

isLive A Boolean value that is `true` if the video stream is live. This value is effective only when streaming from a FCS or FVSS. The value of this property is ignored for an HTTP download. Optional.

Returns

Nothing.

Description

Method; plays the video stream. With no parameters, the method simply takes the FLV file from a paused or stopped state to the playing state.

If parameters are used, the method acts as a shortcut for setting the `autoplay` property to `true` and setting the `isLive`, `totalTime` and, `contentPath` properties. If the `totalTime` and `isLive` properties are undefined, they are not set.

Example

The following example disables the FLV file from playing automatically, calls the `seekSeconds()` method to set the playhead 20 seconds into the video, and calls the `play()` method to begin playing the FLV file at that point.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoplay = false;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlybk.seekSeconds(4);
    my_FLVPlybk.play();
};
my_FLVPlybk.addEventListener("ready", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.autoplay](#), [FLVPlayback.contentPath](#), [FLVPlayback.load\(\)](#),
[FLVPlayback.pause\(\)](#), [FLVPlayback.stop\(\)](#)

FLVPlayback.playButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.playButton
```

Description

Property; a MovieClip object that is the Play button. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, and `stopButton` properties to attach individual FLV Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback`. Next, add the following individual FLV Custom UI components, and give them the instance names shown in parentheses: BackButton (`my_bkbtn`), ForwardButton (`my_fwdbtn`), PlayButton (`my_plybtn`), PauseButton (`my_pausbtn`), and StopButton (`my_stopbtn`). Then add the following lines of code to the Actions panel:

```
/**  
  Requires:  
  - FLVPlayback component on the Stage with an instance name of my_FLVPlayback  
  - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and  
    StopButton components in the Library  
*/  
import mx.video.*;  
my_FLVPlayback.backButton = my_bkbtn;  
my_FLVPlayback.forwardButton = my_fwdbtn;  
my_FLVPlayback.playButton = my_plybtn;  
my_FLVPlayback.pauseButton = my_pausbtn;  
my_FLVPlayback.stopButton = my_stopbtn;
```

See also

[FLVPlayback.playing](#), [FLVPlayback.skin](#)

FLVPlayback.playheadPercentage

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.playheadPercentage
```

Description

Property; a number that specifies the current `playheadTime` as a percentage of the `totalTime` property. If you access this property, it contains the percentage of playing time that has elapsed. If you set this property, it causes a seek operation to the point representing that percentage of the FLV file's playing time.

The value of this property is relative to the value of the `totalTime` property.

The component throws a `VideoError` if you specify a percentage that is invalid or if the `totalTime` property is undefined, null, or less than or equal to zero.

Example

The following example displays the percentage of the FLV file that has played when the `point2` cue point occurs. At the `point3` cue point, it sets `playheadPercentage` to 10, causing a seek operation to the point that is 10 percent from the beginning of the FLV file and creating a playback loop.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    if(eventObject.info.name == "point2")
        trace("point2 occurred at " + my_FLVPlybk.playheadPercentage + "
  percent of FLV");
    if(eventObject.info.name == "point3")
        my_FLVPlybk.playheadPercentage = 10;
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.seekPercent\(\)](#), [FLVPlayback.totalTime](#)

FLVPlayback.playheadTime

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.playheadTime
```

Description

Property; a number that is the current playhead time or position, measured in seconds, which can be a fractional value. Setting this property triggers a seek and has all the restrictions of a seek.

When the playhead time changes, which includes once every .25 seconds while the FLV file plays, the component dispatches the `playheadUpdate` event.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not fire until the `playheadTime` property has updated.

Example

The following example catches occurrences of the `stateChange` event as it occurs while the FLV file plays and shows the elapsed playhead time in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
    trace(my_FLVPlybk.state + ": playhead time is: " +
        my_FLVPlybk.playheadTime);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.playheadUpdate](#), [FLVPlayback.playheadUpdateInterval](#),
[FLVPlayback.seek\(\)](#), [FLVPlayback.stateChange](#)

FLVPlayback.playheadUpdate

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("playheadUpdate", listenerObject);
```

Description

Event; dispatched while the FLV file is playing at the frequency specified by the `playheadUpdateInterval` property. The default is .25 seconds. The component does not dispatch this event when the video player is paused or stopped unless a seek occurs. The event object has the `state`, `playheadTime`, and `vp` properties.

Example

The following example catches occurrences of the `playheadUpdate` event as it occurs while the FLV file plays and displays the elapsed playhead time in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVplybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    trace(my_FLVplybk.state + ": playhead time is: " +
        eventObject.playheadTime);
};
my_FLVplybk.addEventListener("playheadUpdate", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.playheadUpdateInterval](#)

FLVPlayback.playheadUpdateInterval

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.playheadUpdateInterval`

Description

Property; a number that is the amount of time, in milliseconds, between each `playheadUpdate` event. Setting this property while the FLV file is playing restarts the timer. The default is 250.

Because ActionScript cue points start on playhead updates, lowering the value of the `playheadUpdateInterval` property can increase the accuracy of ActionScript cue points.

Because the playhead update interval is set by a call to the global `setInterval()` function, the update cannot fire more frequently than the SWF file frame rate, as with any interval that is set this way. So, as an example, for the default frame rate of 12 frames per second, the lowest effective interval that you can create is approximately 83 milliseconds, or one second (1000 milliseconds) divided by 12.

Example

The following example sets the `playheadUpdateInterval` property to 3000 and creates a listener that catches occurrences of the `playheadUpdate` event as it occurs while the FLV file plays. When the event occurs, the event handler shows the elapsed playhead time in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.playheadUpdateInterval = 3000;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    trace("playhead time is: " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("playheadUpdate", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.playheadUpdate](#)

FLVPlayback.PLAYING

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.PLAYING
```

Description

A read-only FLVPlayback class property that contains the string constant, "playing". You can compare this property to the `state` property to determine if the component is in the playing state.

Example

The following example uses the `FLVPlayback.PLAYING` property to see if the state equals "playing" when a `stateChange` event occurs. It also includes the constant as part of a message in the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.PLAYING)
        trace(my_FLVPlybk.contentPath + " is now " + FLVPlayback.PLAYING);
}
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.playing

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var :Object = new Object();
listenerObject.playing = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("playing", listenerObject);
```

Description

Event; dispatched when the playing state is entered. This may not occur immediately after the `play()` method is called or the corresponding control is clicked; often the buffering state is entered first, and then the playing state. The event object has the `state`, `playheadTime`, and `vp` properties, which is the index number of the video player to which this event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The FLVPlayback instance also dispatches the `stateChange` event.

Example

The following example shows the value of the `contentPath` property in a text area when the `playing` event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVplybk**. Drag a TextArea component to the Stage below the FLVPlayback instance, and give it an instance name of **my_ta**. Then add the following code to Frame 1 of the Timeline in the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playing = function(eventObject:Object):Void {
    my_ta.text = "Now playing: " + my_FLVplybk.contentPath;
}
my_FLVplybk.addEventListener("playing", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.play\(\)](#), [FLVPlayback.playing](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.playing

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.playing
```

Description

Property; a Boolean value that is `true` if the FLV file is in the playing state. Read-only.

Example

The following example listens for occurrences of the `stateChange` event as it occurs while the FLV file plays. When the event occurs, the example shows the value of the `playing` property in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
trace(my_FLVPlayback.state + ": playing property is " + my_FLVPlayback.playing);
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object:Object):Void {
    trace(my_FLVPlayback.state + ": playing property is " +
        my_FLVPlayback.playing);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
```

See also

[FLVPlayback.playing](#), [FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.playPauseButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.playPauseButton
```

Description

Property; a MovieClip object that is the PlayPauseButton. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `playPauseButton`, `stopButton`, `backButton`, and `forwardButton` properties to attach individual FLV Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback`. Next, add the following individual FLV Custom UI components, and give them the instance names shown in parentheses: BackButton (`my_bkbtn`), ForwardButton (`my_fwdbtn`), PlayPauseButton (`my_plpausebtn`), and StopButton (`my_stopbtn`). Then add the following lines of code to the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI PlayPauseButton, StopButton, BackButton, and ForwardButton
 *   components in the Library
 */
import mx.video.*;
my_FLVPlayback.playPauseButton = my_plpausebtn;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.backButton = my_bkbtn;
my_FLVPlayback.forwardButton = my_fwdbtn;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.playButton](#), [FLVPlayback.playPauseButton](#), [FLVPlayback.paused](#),
[FLVPlayback.skin](#)

FLVPlayback.preferredHeight

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.preferredHeight
```

Description

Property; a number that specifies the height of the source FLV file. This information is not valid immediately upon calling the `play()` or `load()` methods. It is valid when the `ready` event starts. If the value of the `autoSize` property or `maintainAspectRatio` property is `true`, it is best to read the value when the `resize` event starts. Read-only.

Example

The following example sets the size of the `FLVPlayback` instance when the `ready` event occurs. When the `cuePoint` event occurs, it resets the size to the size specified by `preferredHeight` and `preferredWidth` properties.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
    trace("width is: " + my_FLVPlayback.width);
    trace("height is: " + my_FLVPlayback.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlayback.setSize(250, 350);
};
```

```
my_FLVPlayback.addEventListener("ready", listenerObject);
listenerObject.cuePoint = function(eventObject:Object):Void {
    my_FLVPlayback.setSize(my_FLVPlayback.preferredWidth,
        my_FLVPlayback.preferredHeight);
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
my_FLVPlayback.addASCuePoint(1.5, "AScp1");
```

See also

[FLVPlayback.autoSize](#), [FLVPlayback.height](#), [FLVPlayback.maintainAspectRatio](#),
[FLVPlayback.preferredWidth](#), [FLVPlayback.ready](#), [FLVPlayback.setSize\(\)](#),
[FLVPlayback.setScale\(\)](#), [FLVPlayback.width](#)

FLVPlayback.preferredWidth

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.preferredWidth

Description

Property; gives the width of the source FLV file. This information is not valid immediately when the `play()` or `load()` methods are called; it is valid when the `ready` event starts. If the value of the `autoSize` or `maintainAspectRatio` properties is `true`, it is best to read the value when the `resize` event starts. Read-only.

Example

The following example sets the size of the `FLVPlayback` instance when the `ready` event occurs. When the `cuePoint` event occurs, it resets the size to the size specified by `preferredHeight` and `preferredWidth` properties.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
    trace("width is: " + my_FLVPlayback.width);
    trace("height is: " + my_FLVPlayback.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlayback.setSize(250, 350);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
listenerObject.cuePoint = function(eventObject:Object):Void {
    my_FLVPlayback.setSize(my_FLVPlayback.preferredWidth,
        my_FLVPlayback.preferredHeight);
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
my_FLVPlayback.addASCuePoint(1.5, "AScp1");
```

See also

[FLVPlayback.autoSize](#), [FLVPlayback.height](#), [FLVPlayback.maintainAspectRatio](#),
[FLVPlayback.preferredHeight](#), [FLVPlayback.ready](#), [FLVPlayback.setSize\(\)](#),
[FLVPlayback.setScale\(\)](#), [FLVPlayback.width](#)

FLVPlayback.progress

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("progress", listenerObject);
```

Description

Event; dispatched at the frequency specified by the `progressInterval` property, starting when the load begins and ending when all bytes are loaded or there is a network error. Default is every .25 seconds.

Dispatched only for a progressive HTTP download. Indicates progress in number of downloaded bytes. The event object has the `bytesLoaded` and `bytesTotal` properties, which are the same as the FLVPlayback properties of the same names.

The event also has the property `vp`, which is the index number of the video player to which this event applies. For more information on the `vp` property, see `FLVPlayback.activeVideoPlayerIndex` and `FLVPlayback.visibleVideoPlayerIndex`.

Example

The following example sets the `progressInterval` property to 001 milliseconds because the FLV file is short and then shows the number of bytes loaded for each occurrence of the progress event.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVplybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
my_FLVplybk.progressInterval = 001;
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
    water.flv";
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
    trace(eventObject.bytesLoaded);
}
my_FLVplybk.addEventListener("progress", listenerObject);
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.addEventListener\(\)](#), [FLVPlayback.bytesLoaded](#), [FLVPlayback.bytesTotal](#), [FLVPlayback.progressInterval](#), [FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.progressInterval

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.progressInterval
```

Description

Property; a number that is the amount of time, in milliseconds, between each `progress` event. If you set this property while the video stream is playing, the timer restarts. Default value is 250.

Example

The following example sets the `progressInterval` property to 001 millisecond because the FLV file is small, and then shows the number of bytes loaded for each occurrence of the `progress` event.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.progressInterval = 001;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
    trace(eventObject.bytesLoaded);
}
my_FLVPlayback.addEventListener("progress", listenerObject);
```

See also

[FLVPlayback.progress](#)

FLVPlayback.ready

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("ready", listenerObject);
```

Description

Event; dispatched when FLV file is loaded and ready to display. It starts the first time you enter a responsive state after you load a new FLV file with the `play()` or `load()` method. It starts only once for each FLV file that is loaded.

The event object has the `state`, `playheadTime`, and `vp` properties. The `vp` property is the index number of the video player to which this event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

Example

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 * - TextArea component on the Stage with an instance name of my_ta
 */
import mx.video.*;
my_ta.visible = false;
my_FLVplybk.autoPlay = false;
my_ta.setSize(260, 30);
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_ta.text = "The FLV is ready. Push Play to start playing";
    my_ta.visible = true;
};
my_FLVplybk.addEventListener("ready", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.addEventListener\(\)](#), [FLVPlayback.state](#), [FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.removeASCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.removeASCuePoint(CuePoint:Object):Object  
my_FLVplybk.removeASCuePoint(time:Number):Object  
my_FLVplybk.removeASCuePoint(name:String):Object
```

Parameters

CuePoint A cue point object containing the `time` and `name` properties for the cue point to remove. The method does not check any other properties on the incoming cue point object. If `time` or `name` is `null` or `undefined`, the method uses only the available property. If only `name` is given, the method removes the first cue point with this name.

time A number containing the time of the cue point to remove. The method removes the first cue point with this time.

name A string that contains the name of the cue point to remove. The method removes the first cue point with this name.

Returns

The cue point object that was removed. If there is no matching cue point, the method returns `null`.

Description

Method; removes an `ActionScript` cue point from the currently loaded `FLV` file. Only the `name` and `time` properties are used from `CuePoint` parameter to find the cue point to remove. If multiple `ActionScript` cue points match the search criteria, only one is removed. To remove all, call this function repeatedly in a loop with the same parameters until it returns `null`.

Cue point information is wiped out when the `contentPath` property is set, so to set cue point information for the next `FLV` file to be loaded, set the `contentPath` property first.

Example

The following example adds an ActionScript cue point to the FLV file, and then calls the `removeASCuePoint()` method to remove it. It shows the name of the removed cue point in the Output panel.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
// create cue point object
var cuePt:Object = new Object(); // create cue point object
var rtn_cuePt:Object = new Object(); // create object for return value
cuePt.time = 4.444;
cuePt.name = "ripples";
my_FLVPlybk.addASCuePoint(cuePt); // add AS cue point
if ((rtn_cuePt = my_FLVPlybk.removeASCuePoint(cuePt)) != null) {
    trace("Removed cue point: " + rtn_cuePt.name);
}
```

See also

[FLVPlayback.addASCuePoint\(\)](#), [FLVPlayback.findCuePoint\(\)](#),
[FLVPlayback.findNearestCuePoint\(\)](#), [FLVPlayback.findNextCuePointWithName\(\)](#)

FLVPlayback.removeEventListener()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.removeEventListener(event:String, listener:Object):Void
my_FLVPlybk.removeEventListener(event:String, listener:Function):Void
```

Parameters

event A string that specifies the name of the event for which you are removing a listener.

listener A reference to the listener object or function that you are removing.

Returns

Nothing.

Description

Method; removes an event listener from a component instance.

Example

The following example removes the listener for a `cuePoint` event when the first cue point occurs. This causes only the first of three cue points to be detected.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

Usage 1: listener object

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var listenerObject:Object = new Object(); // create listener object
listenerObject.cuePoint = function(eventObject:Object):Void {
    trace("Hit cue point at " + eventObject.info.time);
    my_FLVPlybk.removeEventListener("cuePoint", listenerObject);
};
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

Usage 2: listener function

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_ta.visible = false;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
function cuePoint(eventObject:Object):Void {
    trace("Hit cue point at " + eventObject.info.time);
    my_FLVPlybk.removeEventListener("cuePoint", cuePoint);
};
my_FLVPlybk.addEventListener("cuePoint", cuePoint);
```

See also

[FLVPlayback.addEventListener\(\)](#)

FLVPlayback.resize

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("resize", listenerObject);
```

Description

Event; dispatched when video is resized. This occurs when you set the `visibleVideoPlayerIndex` property and switch to a video player with different dimensions. The event object has the properties `auto`, `x`, `y`, `width`, `height` and `vp`, which is the index number of the video player to which the event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The `auto` property is `true` when the resizing is automatic because the `autoSize` or `maintainAspectRatio` property is `true`. In this case, the event might be dispatched for a video player other than the visible video player. The event might be dispatched even if the dimensions do not actually change after an attempt to automatically resize the component occurs.

When the `auto` property is `false`, the event always applies to the visible video player. The `vp` property still appears but will always be equal to the `visibleVideoPlayerIndex` property.

The component dispatches the event (with `auto` set to `false`) when you set the `visibleVideoPlayerIndex` property if you are switching to a video player with different dimensions than the currently visible player.

Example

The following example plays two FLV files. It adds an ActionScript cue point to the first FLV file and, when the `cuePoint` event occurs, it switches to a second, smaller video player to play the second FLV file. When it sets the `visibleVideoPlayerIndex` property for the video player, it triggers the `resize` event, which displays the size and location of the current video player.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
// turn off autoSize and maintainAspectRatio
my_FLVPlayback.autoSize = false;
my_FLVPlayback.maintainAspectRatio = false;
// play this FLV
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";
// add a cue point
my_FLVPlayback.addAScuePoint(3, "switch_here");
var listenerObject:Object = new Object();// create listener
listenerObject.cuePoint = function(eventObject:Object):Void {
    // add a second video player
    my_FLVPlayback.activeVideoPlayerIndex = 1;
    // play this FLV
    my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
    // change size of this video player
    my_FLVPlayback.setSize(240, 180);
    my_FLVPlayback.visibleVideoPlayerIndex = 1; // make it visible
    my_FLVPlayback.play(); // play VLV
};
// add listener for cuePoint event
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
listenerObject.resize = function(eventObject:Object):Void {
    // display location and dimensions
    trace("Video player is #" + my_FLVPlayback.activeVideoPlayerIndex);
    trace("X coordinate is: " + eventObject.x);
    trace("Y coordinate is: " + eventObject.y);
    trace("Width is: " + eventObject.width);
    trace("Height is: " + eventObject.height);
};
// add listener for resize event
my_FLVPlayback.addEventListener("resize", listenerObject);
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.autoSize](#), [FLVPlayback.height](#), [FLVPlayback.maintainAspectRatio](#), [FLVPlayback.preferredHeight](#), [FLVPlayback.preferredWidth](#), [FLVPlayback.setSize\(\)](#), [FLVPlayback.state](#), [FLVPlayback.width](#), [FLVPlayback.x](#), [FLVPlayback.y](#)

FLVPlayback.rewind

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.rewind = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("rewind", listenerObject);
```

Description

Event; dispatched when the location of the playhead moves backward by a call to `seek()` or when an automatic rewind completes.

The event object has the properties `auto`, `state`, and `playheadTime`. If the event results from a seek backward, the `auto` property is `false`. If it results from an automatic rewind, the `auto` property is `true`.

The `playheadTime` property is the destination time.

The `stateChange` event is dispatched with a state of "rewinding" when an automatic rewind occurs. The `stateChange` event does not start until rewinding has completed. The `seek` event is dispatched when rewinding occurs through seeking. The `FLVPlayback` instance also dispatches the `playheadUpdate` event when rewinding occurs.

The `rewind` event has the property `vp`, the index number of the video player to which this event applies. For more information on the `vp` property, see the [FLVPlayback.activeVideoPlayerIndex](#) and [FLVPlayback.visibleVideoPlayerIndex](#) properties.

Example

The following example sets the `autoRewind` property to `true` and listens for the `rewind` event. When the event occurs, the event handler shows the values of the `vp`, `state`, and `playheadTime` properties in the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoRewind = true;
var listenerObject:Object = new Object();
listenerObject.rewind = function(eventObject:Object) {
    trace("Video player is #" + eventObject.vp);
    trace("State is: " + eventObject.state);
    trace("Playhead time is: " + eventObject.playheadTime);
};
my_FLVPlybk.addEventListener("rewind", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.playheadTime](#),
[FLVPlayback.playheadUpdateFLVPlayback.seek\(\)](#), [FLVPlayback.seekPercent\(\)](#),
[FLVPlayback.seekSeconds\(\)](#), [FLVPlayback.seekToNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.state](#),
[FLVPlayback.stateChange](#)

FLVPlayback.REWINDING

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.REWINDING
```

Description

A read-only FLVPlayback class property that contains the string constant, "rewinding". You can compare this property to the `state` property to determine if the component is in the rewinding state.

Example

The following example creates a listener for the `stateChange` event and uses the `FLVPlayback.REWINDING` property to determine whether the component is in the rewinding state.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.REWINDING)
        trace("The current state is " + FLVPlayback.REWINDING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.scaleX

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.scaleX
```

Description

Property; a number that is the horizontal scale. The standard scale is 100.

Example

The following example sets the `scaleX` (horizontal) and `scaleY` (vertical) properties of the `FLVPlayback` instance to 150 percent.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.scaleX = 150;
my_FLVPlybk.scaleY = 150;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.setScale\(\), FLVPlayback.scaleY](#)

FLVPlayback.scaleY

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.scaleY
```

Description

Property; a number that is the vertical scale. The standard scale is 100.

Example

The following example sets the horizontal (`scaleX`) and vertical (`scaleY`) scale of the FLVPlayback instance to 150 percent.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.scaleX = 150;
my_FLVPlybk.scaleY = 150;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.scaleX](#), [FLVPlayback.setScale\(\)](#)

FLVPlayback.scrubbing

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.scrubbing

Description

Property; a Boolean value that is `true` if the user is scrubbing with the SeekBar and `false` otherwise. Read-only.

Scrubbing refers to grabbing the handle of the seek bar and dragging it in either direction to locate a particular scene in the FLV file.

Example

The following example shows the value of the `scrubbing` property when a seek event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

NOTE

You must grab the handle of the SeekBar, drag it, and release it to cause the event.

```

/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
    if(my_FLVPlayback.scrubbing)
        trace("User is scrubbing at: " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("seek", listenerObject);

```

See also

[FLVPlayback.seek](#), [FLVPlayback.seekBar](#), [FLVPlayback.scrubFinish](#),
[FLVPlayback.scrubStart](#)

FLVPlayback.scrubFinish

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```

var listenerObject:Object = new Object();
listenerObject.scrubFinish = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVPlayback.addEventListener("scrubFinish", listenerObject);

```

Description

Event; dispatched when the user stops scrubbing the FLV file with the SeekBar. Scrubbing refers to grabbing the handle of the seek bar and dragging it in either direction to locate a particular scene in the FLV file. Scrubbing stops when the user releases the handle of the SeekBar.

The event object has the properties `state` and `playheadTime`. The `state` will be "seeking" until after scrubbing stops.

The component also dispatches the `stateChange` event with the `state` property equal to the new state, which should be "playing", "paused", "stopped", or "buffering".

Example

The following example listens for the `scrubFinish` event and shows the time at which scrubbing stops.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

NOTE

You must grab the handle of the `SeekBar`, drag it, and release it to cause the event.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.scrubFinish = function(eventObject:Object):Void {
    trace("Scrubbing stopped at " + eventObject.playheadTime);
    trace("Current state is " + eventObject.state);
};
my_FLVPlybk.addEventListener("scrubFinish", listenerObject);
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.seek](#), [FLVPlayback.seekBar](#),
[FLVPlayback.scrubStart](#), [FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.scrubStart

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.scrubStart = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVPlybk.addEventListener("scrubStart", listenerObject);
```

Description

Event; dispatched when the user begins scrubbing the FLV file with the SeekBar. Scrubbing refers to grabbing the handle of the SeekBar and dragging it in either direction to locate a particular scene in the FLV file. Scrubbing begins when the user clicks on the SeekBar handle and ends when the user releases it.

The event object has the properties `state` and `playheadTime`.

The component also dispatches the `stateChange` event with the `state` property equal to "seeking". The state remains "seeking" until the user stops scrubbing.

Example

The following example listens for the `scrubStart` event and shows the time when scrubbing begins.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

NOTE

You must grab the handle of the `SeekBar` and drag it to cause the event.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.scrubStart = function(eventObject:Object):Void {
    trace("Scrubbing began at " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("scrubStart", listenerObject);
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.scrubbing](#), [FLVPlayback.scrubFinish](#),
[FLVPlayback.seekBar](#), [FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.seek

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("seek", listenerObject);
```

Description

Event; dispatched when the location of playhead is changed by a call to `seek()`, by setting the `playheadTime` property or by using the `seekBar` control. The `playheadTime` property is the destination time. The event object has the properties `state`, `playheadTime`, and `vp`, which is the index number of the video player to which the event applies.

The `FLVPlayback` instance dispatches the `rewind` event when the seek is backward and the `fastForward` event when the seek is forward. It also dispatches the `playheadUpdate` event.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example seeks 2 seconds into the FLV file when the `ready` event occurs. The `seek()` function triggers a `seek` event, at which point the listener displays the `playheadTime` and the name of the `FLVPlayback` instance.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object) {
    trace("A seek event occurred at " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("seek", listenerObject);
listenerObject.ready = function(eventObject:Object) {
    my_FLVPlayback.seek(2);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.fastForward](#),
[FLVPlayback.playheadTime](#), [FLVPlayback.playheadUpdate](#), [FLVPlayback.rewind](#),
[FLVPlayback.seek\(\)](#), [FLVPlayback.seekPercent\(\)](#), [FLVPlayback.seekSeconds\(\)](#),
[FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToNextNavCuePoint\(\)](#),
[FLVPlayback.seekToPrevNavCuePoint\(\)](#)

FLVPlayback.seek()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.seek(time:Number)
```

Parameters

time A number that specifies the time, in seconds, at which to place the playhead.

Returns

Nothing.

Description

Method; seeks to a given time in the file, specified in seconds, with a precision of three decimal places (milliseconds).

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example disables the FLV file from playing automatically, calls the `seek()` method to set the playhead 3 seconds into the video, and begins playing the FLV file at that point.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlaybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 */
import mx.video.*;
my_FLVPlaybk.autoPlay = false;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
my_FLVPlaybk.seek(3);
my_FLVPlaybk.play();
```

See also

[FLVPlayback.playheadTime](#), [FLVPlayback.seek](#), [FLVPlayback.seekPercent\(\)](#),
[FLVPlayback.seekSeconds\(\)](#)

FLVPlayback.seekBar

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.seekBar
```

Description

Property; a MovieClip object that is the seek bar control at playtime. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the backButton, forwardButton, playButton, pauseButton, stopButton, and seekBar properties to attach individual FLV Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, give it an instance name of **my_FLVPlayback**, and set the skin parameter to None in the Component inspector. Next, add the following individual FLV Custom UI components, and give them the instance names shown in parentheses: BackButton (**my_bkbbtn**), ForwardButton (**my_fwdbbtn**), PlayPauseButton (**my_plypausbbtn**), StopButton (**my_stopbbtn**) and SeekBar (**my_seekBar**). Then add the following lines of code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayPauseButton, StopButton and
 *   SeekBar
 *   components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbbtn;
my_FLVPlayback.forwardButton = my_fwdbbtn;
my_FLVPlayback.playPauseButton = my_plypausbbtn;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.seekBar = my_seekBar;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```


See also

[FLVPlayback.scrubbing](#), [FLVPlayback.scrubFinish](#), [FLVPlayback.scrubStart](#), [FLVPlayback.seek](#)

FLVPlayback.seekBarInterval

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.seekBarInterval
```

Description

Property; a number that specifies, in milliseconds, how often to check the seek bar handle when scrubbing. The default value is 250.

Because this interval is set by a call to the global `setInterval()` function, the update cannot start more frequently than the SWF file frame rate. So, for the default frame rate of 12 frames per second, for example, the lowest effective interval that you can create is approximately 83 milliseconds, or 1 second (1000 milliseconds) divided by 12.

Example

The following example lowers the `seekBarInterval` setting to 50 milliseconds, and it shows the value of the `playheadTime` property, if the user is scrubbing.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
    if(my_FLVPlayback.scrubbing) {
        my_FLVPlayback.seekBarInterval = 50;
        trace("User is scrubbing at: " + eventObject.playheadTime);
    }
};
my_FLVPlayback.addEventListener("seek", listenerObject);
```

See also

[FLVPlayback.seekBar](#), [FLVPlayback.seekBarScrubTolerance](#)

FLVPlayback.seekBarScrubTolerance

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.seekBarScrubTolerance
```

Description

Property; a number that specifies how far a user can move the SeekBar handle before an update occurs. The value is specified as a percentage, ranging from 1 to 100. The default value is 5.

Example

The following example checks to see if the user is scrubbing when a seek event occurs and, if so, lowers the value of the `seekBarScrubTolerance` property to 0 to increase updating the SeekBar location and the frequency of the seek event.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

NOTE

You must grab the handle of the SeekBar, drag it, and release it to cause the event.

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
    if(my_FLVPlayback.scrubbing) {
        my_FLVPlayback.seekBarScrubTolerance = 0;
        trace("User is scrubbing at: " + eventObject.playheadTime);
    }
};
my_FLVPlayback.addEventListener("seek", listenerObject);
```

See also

[FLVPlayback.scrubbing](#), [FLVPlayback.scrubFinish](#), [FLVPlayback.scrubStart](#), [FLVPlayback.seekBar](#), [FLVPlayback.seekBarInterval](#)

FLVPlayback.SEEKING

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.SEEKING
```

Description

A read-only FLVPlayback class property that contains the string constant, "seeking". You can compare this property to the `state` property to determine whether the component is in the seeking state.

Example

The following example uses the `FLVPlayback.SEEKING` property to see if the state is "seeking" when a `stateChange` event occurs and, if so, shows a message to that effect.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.SEEKING)
        trace("The current state is " + FLVPlayback.SEEKING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.seekPercent()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.seekPercent(percent:Number)
```

Parameters

percent A number that specifies a percentage of the length of the FLV file at which to place the playhead.

Returns

Nothing.

Description

Method; seeks to a percentage of the file and places the playhead there. The percentage is a number between 0 and 100.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example disables the FLV file from playing automatically. When the FLV file is ready, it sets the playhead 30 percent into the playing time and begins playing at that point.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoPlay = false;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    my_FLVPlybk.seekPercent(30);
    my_FLVPlybk.play();
}
my_FLVPlybk.addEventListener("ready", listenerObject);
```

See also

[FLVPlayback.seek](#), [FLVPlayback.seek\(\)](#), [FLVPlayback.seekSeconds\(\)](#)

FLVPlayback.seekSeconds()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.seekSeconds(time:Number)
```

Parameters

time A number that specifies the time, in seconds, of the total play time at which to place the playhead.

Returns

Nothing.

Description

Method; seeks to a given time in the file, specified in seconds, with a precision up to three decimal places (milliseconds). This method performs the same operation as the `seek()` method; it is provided for symmetry with the `seekPercent()` method.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example disables the FLV file from playing automatically, calls the `seekSeconds()` method to set the playhead 5 seconds into the video, and begins playing the FLV file at that point.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoPlay = false;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    my_FLVPlybk.seekSeconds(4);
    my_FLVPlybk.play();
}
my_FLVPlybk.addEventListener("ready", listenerObject);
```

See also

[FLVPlayback.seek](#), [FLVPlayback.seek\(\)](#), [FLVPlayback.seekPercent\(\)](#)

FLVPlayback.seekToNavCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.seekToNavCuePoint(time:Number):Void  
my_FLVplybk.seekToNavCuePoint(name:String):Void  
my_FLVplybk.seekToNavCuePoint(cuePoint:Object):Void
```

Parameters

time A number that is the time of the navigation cue point to seek. The method uses only the first three decimal places and rounds any additional decimal places.

name A string that is the name of the cue point to seek.

cuePoint A cue point object in which you set the *time* and *name* properties to specify the cue point to seek.

Returns

Nothing.

Description

Method; seeks to a navigation cue point that matches the specified time or is later. If time is undefined, null, or less than 0, the method starts its search at time 0.

If you specify only a time, the method seeks to a cue point that matches that time or is later.

If you specify a name, the method seeks to the first enabled cue point that matches it (for more information about enabling/disabling cue points see [“FLVPlayback.setFLVCuePointEnabled\(\)” on page 665](#)).

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example seeks to the cue point named `point2` when the `ready` event occurs. The `cuePoint` event handler shows the `name`, `time`, and `type` values for each cue point that occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    my_FLVPlayback.seekToNavCuePoint("point2");
}
my_FLVPlayback.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point time is: " + eventObject.info.time);
    trace("Cue point type is: " + eventObject.info.type);
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.cuePoint](#), [FLVPlayback.seek](#), [FLVPlayback.seek\(\)](#),
[FLVPlayback.seekToNextNavCuePoint\(\)](#)

FLVPlayback.seekToNextNavCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.seekToNextNavCuePoint([ time:Number ])
```

Parameters

time A number that is the starting time, in seconds, from which to look for the next navigation cue point. The default is the current `playheadTime` property. Optional.

Returns

Nothing.

Description

Method; seeks to next navigation cue point, based on the current value of the `playheadTime` property. The method skips navigation cue points that have been disabled and goes to the end of the FLV file if there is no other cue point.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the seek event, which does not start until the `playheadTime` property has updated.

Example

The following example seeks to the next navigation cue point when the cue point named `point2` occurs. This has the effect of skipping that portion of the FLV file between the cue points named `point2` and `point3`.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
    if(eventObject.info.name == "point2")
        my_FLVPlybk.seekToNextNavCuePoint(eventObject.info.time);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
```

See also

[FLVPlayback.cuePoint](#), [FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.playheadTime](#), [FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.setFLVCuePointEnabled\(\)](#)

FLVPlayback.seekToPrevNavCuePoint()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.seekToPrevCueNavPoint([time:Number])
```

Parameters

time A number that is the starting time in seconds from which to look for the previous navigation cue point. The default is the current value of the `playheadTime` property. Optional.

Returns

Nothing.

Description

Method; seeks to the previous navigation cue point, based on the current value of the `playheadTime` property. It goes to the beginning if there is no previous cue point. The method skips navigation cue points that have been disabled.

For several reasons, the `playheadTime` property might not have the expected value immediately after calling one of the seek methods or setting `playheadTime` to cause seeking. First, for a progressive download, you can seek only to a keyframe, so a seek takes you to the time of the first keyframe after the specified time. (When streaming, a seek always goes to the precise specified time even if the source FLV file doesn't have a keyframe there.) Second, seeking is asynchronous, so if you call a seek method or set the `playheadTime` property, `playheadTime` does not update immediately. To obtain the time after the seek is complete, listen for the `seek` event, which does not start until the `playheadTime` property has updated.

Example

The following example seeks to the previous navigation cue point when the `point2` cue point occurs, creating a loop to play the FLV file.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
    if(eventObject.info.name == "point2")
        my_FLVPlybk.seekToPrevNavCuePoint(eventObject.info.time);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

See also

[FLVPlayback.cuePoint](#), [FLVPlayback.findCuePoint\(\)](#), [FLVPlayback.playheadTime](#), [FLVPlayback.seekToNavCuePoint\(\)](#), [FLVPlayback.seekToPrevNavCuePoint\(\)](#), [FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.setFLVCuePointEnabled\(\)](#)

FLVPlayback.seekToPrevOffset

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.seekToPrevOffset

Description

Property; the number of seconds that the `seekToPrevNavCuePoint()` method uses when it compares its time against the previous cue point. The method uses this value to ensure that, if you are just ahead of a cue point, you can hop over it to the previous one and avoid going to the same cue point that just occurred. Defaults to one second.

Example

The following example initially sets the `seekToPrevOffset` property to 10, causing the first call to the `seekToPrevNavCuePoint()` method to hit cue point point1. When the initial `cuePoint` event for point3 occurs, however, the example lowers the `seekToPrevOffset` property to 1 second, causing subsequent calls to the `seekToPrevNavCuePoint()` method to reach cue point point2.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
    my_FLVPlayback.seekToPrevOffset = 10;
    my_FLVPlayback.seekToNavCuePoint("point3");
}
my_FLVPlayback.addEventListener("ready", listenerObject)
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
    trace("hit cue point at " + eventObject.info.time);
    if(eventObject.info.name == "point3"){
        my_FLVPlayback.seekToPrevNavCuePoint(eventObject.info.time);
        my_FLVPlayback.seekToPrevOffset = 1;
    }
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject)
```

See also

[FLVPlayback.seekToPrevNavCuePoint\(\)](#)

FLVPlayback.setFLVCuePointEnabled()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, time:Number)
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, name:String)
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, cuePoint:Object)
```

Parameters

enabled A Boolean value that specifies whether to enable (`true`) or disable (`false`) an FLV file cue point.

time A number that is the time, in seconds, of the cue point to set.

name The name of the cue point to set.

cuePoint A cue point object with `name` and `time` properties that matches the cue point to set. The method does not check any other properties on the incoming cue point object. If `time` or `name` is undefined, the method tries to match a cue point using only the available value.

Returns

A number. If `metadataLoaded` is `true`, the method returns the number of cue points whose enabled state was changed. If `metadataLoaded` is `false`, the method returns `-1` because the component cannot yet determine which, if any, cue points to set. When the metadata arrives, however, the component sets the specified cue points appropriately.

Description

Method; enables or disables one or more FLV file cue points. Disabled cue points are disabled for purposes of being dispatched as events and for navigating to them with the `seekToPrevNavCuePoint()`, `seekToNextNavCuePoint()`, and `seekToNavCuePoint()` methods.

Cue point information is deleted when you set the `contentPath` property to a different FLV file, so set the `contentPath` property before setting cue point information for the next FLV file to be loaded.

Changes caused by this function are not reflected by calls to the `isFLVCuePointEnabled()` method until metadata is loaded.

Example

The following example disables the `point2` and `point3` cue points when the `ready` event occurs. The `cuePoint` event handler shows in the Output panel the name and time of each cue point that occurs. The FLV file contains the following embedded cue points: `point1` at `00:00:00:418`; `point2` at `00:00:07.748`; `point3` at `00:00:16:020`.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
function ready(eventObject:Object) {
    my_FLVPlayback.setFLVCuePointEnabled(false, "point2");
    my_FLVPlayback.setFLVCuePointEnabled(false, 16.02);
}
my_FLVPlayback.addEventListener("ready", ready);
function cuePoint(eventObject:Object) {
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point time is: " + eventObject.info.time);
}
my_FLVPlayback.addEventListener("cuePoint", cuePoint);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.cuePoint](#), [FLVPlayback.findCuePoint\(\)](#),
[FLVPlayback.findNearestCuePoint\(\)](#), [FLVPlayback.findNextCuePointWithName\(\)](#),
[FLVPlayback.isFLVCuePointEnabled\(\)](#), [FLVPlayback.seekToNavCuePoint\(\)](#),
[FLVPlayback.seekToNextNavCuePoint\(\)](#), [FLVPlayback.seekToPrevNavCuePoint\(\)](#)

FLVPlayback.setScale()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.setScale(xs:Number, ys:Number)
```

Parameters

xs A number representing the horizontal scale.

ys A number representing the vertical scale.

Returns

Nothing.

Description

Method; sets the `scaleX` and `scaleY` properties simultaneously. Because setting either one, individually, can cause automatic resizing, setting them simultaneously can be more efficient than setting the `scaleX` and `scaleY` properties, individually.

If `autoSize` is true, this method has no effect because the player sets its own dimensions. If the `maintainAspectRatio` property is true and `autoSize` is false, then changing `scaleX` or `scaleY` causes automatic resizing.

Example

The following example calls the `setScale()` method to scale the horizontal (*x*) and vertical (*y*) dimensions of the `FLVPlayback` instance. The example sets the `maintainAspectRatio` property to `false` to prevent automatic resizing and allow the scaling to appear as specified.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;

my_FLVPlayback.maintainAspectRatio = false;
my_FLVPlayback.setScale(200, 175);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.scaleX](#), [FLVPlayback.scaleY](#), [FLVPlayback.setSize\(\)](#)

FLVPlayback.setSize()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.setSize(w:Number, h:Number)
```

Parameters

w A number that specifies the width of the video player.

h A number that specifies the height of the video player.

Returns

Nothing.

Description

Method; sets the width and height simultaneously. Because setting either one, individually, can cause automatic resizing, setting them simultaneously can be more efficient than setting the width and height properties individually.

If `autoSize` is true, this method has no effect because the player sets its own dimensions. If the `maintainAspectRatio` is true and the `autoSize` property is false, changing the width or height causes automatic resizing.

Example

The following example calls the `setSize()` method to set the size of the `FLVPlayback` instance to a width of 150 pixels and a height of 150 pixels. The `resize` event handler shows the actual width and height because the `maintainAspectRatio` property is true by default, so an automatic resizing maintains the aspect ratio.

Drag the `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
// maintainAspectRatio is true by default so dimensions will reflect that
my_FLVPlybk.setSize(150, 150);
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
    trace("Player's width is: " + my_FLVPlybk.width)
    trace("Player's height is: " + my_FLVPlybk.height)
};
my_FLVPlybk.addEventListener("resize", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.height](#), [FLVPlayback.width](#), [FLVPlayback.setScale\(\)](#)

FLVPlayback.skin

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.skin

Description

Property; a string that specifies the URL to a skin SWF file. This string could contain a file name, a relative path such as `Skins/my_Skin.swf`, or an absolute URL such as `http://www.myskins.org/MySkin.swf`.

Example

The following example applies the skin `ArcticExternal.swf` to an instance of the `FLVPlayback` component.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlayback**. Copy the `ArcticExternalAll.swf` file from the `Flash Configuration/Skins` folder to your working folder. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.skin = "ArcticExternalAll.swf";
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#), [FLVPlayback.skinAutoHide](#), [FLVPlayback.skinError](#), [FLVPlayback.skinLoaded](#)

FLVPlayback.skinAutoHide

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVplybk.skinAutoHide
```

Description

Property; a Boolean value that, if `true`, hides the component skin when the mouse is not over the video. This property affects only skins that are loaded by setting the `skin` property and not a skin that you create from the FLV Playback Custom UI components. Defaults to `false`.

Example

The following example sets the `skinAutoHide` property to `true` so the component skin, which includes the playback controls, does not appear unless the mouse is over the video.

Drag an FLVPlayback component to the Stage, and give it an instance name of `my_FLVplybk`. Select a skin in the Component inspector. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
my_FLVplybk.skinAutoHide = true;
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
```

See also

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#), [FLVPlayback.skin](#)

FLVPlayback.skinError

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.skinError = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("skinError", listenerObject);
```

Description

Event; dispatched when an error occurs loading a skin SWF file. The event has a message property that contains the error message.

Example

The following example attempts to load the skin property with the name of a fictitious skin file and shows the content of the event message property when the skinError event occurs.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVplybk**. Then add the following code to Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.skinError = function(eventObject:Object):Void {
    trace(eventObject.message);
}
my_FLVplybk.addEventListener("skinError", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
my_FLVplybk.skin = "NoSuchSkin.swf";
```

See also

[FLVPlayback.skin](#), [FLVPlayback.skinLoaded](#)

FLVPlayback.skinLoaded

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.skinLoaded = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("skinLoaded", listenerObject);
```

Description

Event; dispatched when a skin SWF file is loaded. The component does not begin playing an FLV file until the `ready` and `skinLoaded` (or `skinError`) events have both started.

Example

The following example shows the name of the component's skin when the `skinLoaded` event starts.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPLYBK`. Then add the following code to Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPLYBK
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.skinLoaded = function(eventObject:Object):Void {
    trace("Skin: " + eventObject.target.skin + " has loaded");
};
my_FLVPLYBK.addEventListener("skinLoaded", listenerObject);
my_FLVPLYBK.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#), [FLVPlayback.skin](#), [FLVPlayback.skinError](#)

FLVPlayback.state

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.state
```

Description

Property; a string that specifies the state of the component. This property is set by the `load()`, `play()`, `stop()`, `pause()`, and `seek()` methods. Read-only.

The possible values for the `state` property are: "buffering", "connectionError", "disconnected", "loading", "paused", "playing", "rewinding", "seeking", and "stopped". You can use the FLVPlayback class properties to test for these states. For more information, see [“FLVPlayback Class properties” on page 541](#).

Example

The following example shows the `state` property in the Output panel each time the `stateChange` event occurs while the FLV file plays.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    trace(my_FLVPlayback.state);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#), [FLVPlayback.buffering](#), [FLVPlayback.paused](#), [FLVPlayback.playing](#), [FLVPlayback.stateChange](#), [FLVPlayback.stopped](#)

FLVPlayback.stateChange

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("stateChange", listenerObject);
```

Description

Event; dispatched when playback state changes. The event object has properties `state` and `playheadTime`.

This event can be used to track when playback enters or leaves unresponsive states (such as in the middle of connecting, resizing, or rewinding) during which times the `play()`, `pause()`, `stop()`, and `seek()` methods queue the requests to be executed when the player enters a responsive state.

The event has the property `vp`, which is the index number of the video player to which this event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

Example

The following example shows the `state` property in the Output panel each time the `stateChange` event occurs while the FLV file plays.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
    trace(my_FLVPlybk.state);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#), [FLVPlayback.state](#)

FLVPlayback.stateResponsive

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.stateResponsive

Description

Property; a Boolean value that is true if the state is responsive. If the state is unresponsive, calls to the `play()`, `load()`, `stop()`, `pause()` and `seek()` methods are queued and executed later, when the state changes to a responsive one. Because these calls are queued and executed later, it is usually not necessary to track the value of the `stateResponsive` property. The responsive states are: `disconnected`, `stopped`, `playing`, `paused`, and `buffering`. Read only.

Example

The following example displays the values of the `state` and `stateResponsive` properties as the state changes while the FLV file plays.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
    trace(my_FLVPlayback.state + "; responsive: " +
        my_FLVPlayback.stateResponsive);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.stop()

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.stop()
```

Parameters

None.

Returns

Nothing.

Description

Method; stops the video from playing. If the `autoRewind` property is true, the FLV file rewinds to the beginning.

Example

The following example listens for the `playheadUpdate` event, and when the elapsed `playheadTime` is greater than or equal to 5 seconds, the listener calls the `stop()` method to stop playing the FLV file. A second listener listens for the `stopped` event and displays the values of the `playheadTime` and `state` properties.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoRewind = false;
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
    trace("playhead time is: " + eventObject.playheadTime);
    trace("The video player state is: " + eventObject.state);
};
my_FLVPlybk.addEventListener("stopped", listenerObject);
listenerObject.playheadUpdate = function(eventObject:Object):Void {
    if (eventObject.playheadTime >= 5) {
        my_FLVPlybk.stop();
    }
};
my_FLVPlybk.addEventListener("playheadUpdate", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.pause\(\)](#), [FLVPlayback.play\(\)](#)

FLVPlayback.stopButton

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlybk.stopButton`

Description

Property; a MovieClip object that is the Stop button control. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, and `stopButton` properties to attach individual FLV Playback Custom UI controls to an `FLVPlayback` component.

Drag an `FLVPlayback` component to the Stage, give it an instance name of `my_FLVPlybk`, and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Playback Custom UI components, and give them the instance names shown in parentheses: `BackButton` (`my_bkbtn`), `ForwardButton` (`my_fwdbtn`), `PlayButton` (`my_plybtn`), `PauseButton` (`my_pausbtn`), and `StopButton` (`my_stopbtn`). Then add the following lines of code to the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and
 *   StopButton components in the Library
 */
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
my_FLVPlybk.playButton = my_plybtn;
my_FLVPlybk.pauseButton = my_pausbtn;
my_FLVPlybk.stopButton = my_stopbtn;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.pauseButton](#), [FLVPlayback.playButton](#), [FLVPlayback.playPauseButton](#), [FLVPlayback.skin](#), [FLVPlayback.stopped](#)

FLVPlayback.STOPPED

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.STOPPED
```

Description

A read-only FLVPlayback class property that contains the string constant, "stopped". You can compare this property to the `state` property to determine whether the component is in the stopped state.

Example

The following example displays the value of the `FLVPlayback.STOPPED` property when the component enters a stopped state.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
    if(eventObject.state == FLVPlayback.STOPPED)
        trace("State is " + FLVPlayback.STOPPED);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#)

FLVPlayback.stopped

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("stopped", listenerObject);
```

Description

Event; dispatched when entering the stopped state. This happens when you call the `stop()` method or click the `stopButton` control. It also happens, in some cases, if the `autoPlay` property is `false` (the state might become paused instead) when the FLV file is loaded. The `FLVPlayback` instance also dispatches this event when the playhead stops at the end of the FLV file. The event object has the properties `state`, `playheadTime`, and `vp`, which is the index number of the video player to which the event applies. For more information on the `vp` property, see [FLVPlayback.activeVideoPlayerIndex on page 549](#) and [FLVPlayback.visibleVideoPlayerIndex on page 688](#).

The `FLVPlayback` instance also dispatches the `stateChange` event.

Example

The following example listens for occurrences of the `stopped` event as it occurs while the FLV file plays. When the event occurs the example shows the elapsed playhead time in the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
    trace(my_FLVPlybk.state + ": playhead time is: " +
        eventObject.playheadTime);
};
my_FLVPlybk.addEventListener("stopped", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#), [FLVPlayback.playheadTime](#), [FLVPlayback.state](#), [FLVPlayback.stateChange](#), [FLVPlayback.stop\(\)](#)

FLVPlayback.stopped

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlybk.stopped

Description

Property; a Boolean value that is true if the state of the FLVPlayback instance is stopped. Read-only.

Example

The following example listens for occurrences of the stateChange event as it occurs while the FLV file plays. When the event occurs, the example shows the value of the stopped property in the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
    trace(my_FLVPlybk.state + ": stopped property is: " +
        my_FLVPlybk.stopped);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.state](#), [FLVPlayback.stateChange](#), [FLVPlayback.stop\(\)](#),
[FLVPlayback.stopped](#)

FLVPlayback.totalTime

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlybk.totalTime
```

Description

Property; a number that is the total playing time for the video in seconds. When streaming from a FCS and using the default NCMManager, this value is determined automatically by server-side APIs, and that value overrides anything set through this property or gathered from metadata. This is also true if you set this value in a SMIL file. The property is ready for reading when the stopped or playing state is reached after setting the `contentPath` property. This property is meaningless for live streams from a FCS.

For an HTTP download, the value is determined automatically if the FLV file has metadata embedded; otherwise, set it explicitly or it will be 0. If you set it explicitly, the metadata value in the stream is ignored.

When you set this property, the value takes effect for the next FLV file that is loaded by setting `contentPath`. It has no effect on an FLV file that has already loaded. Also, this property does not return the new value passed in until an FLV file is loaded.

Playback still works if this property is never set (either explicitly or automatically), but it can cause problems with seek controls.

Example

The following example shows the total time for the FLV file in seconds when the `ready` event occurs, after loading is complete.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    trace("Total play time for this video is: " + my_FLVPlybk.totalTime);
};
my_FLVPlybk.addEventListener("ready", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.contentPath](#), [FLVPlayback.playheadTime](#), [FLVPlayback.playing](#),
[FLVPlayback.stopped](#)

FLVPlayback.transform

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlybk.transform`

Description

Property; an object that provides direct access to the `Sound.setTransform()` and `Sound.getTransform()` methods to provide sound control. You must set this property to an object to initialize it and for changes to take effect. Reading the property provides you with a copy of the current settings, which you can change. The default value is undefined.

Example

The following example sets the `transform` property to play the sound for the FLV file from the left speaker only.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
/* Play all the audio from the left speaker only */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object) {
    if (eventObject.target.state == "loading") { // if loading
        myTransform = new Object();
        myTransform.ll = 100;
        myTransform.lr = 100;
        myTransform.rl = 0;
        myTransform.rr = 0;
        my_FLVPlayback.transform = myTransform;
    }
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.volume](#)

FLVPlayback.version

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.FLVPlayback.version
```

Description

A read-only FLVPlayback class property that contains the component's version number.

Example

The following example shows the component's version number in the Output panel. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
import mx.video.*;
trace(FLVPlayback.version);
```

FLVPlayback.visible

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlaybk.visible
```

Description

Property; a Boolean value that, if `true`, makes the FLVPlayback component visible. If `false`, it makes the component invisible. The default value is `true`.

Example

The following example sets the `visible` property to `false` to make the FLVPlayback instance invisible when the FLV file finishes playing.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
    my_FLVPlybk.visible = false;
};
my_FLVPlybk.addEventListener("complete", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.closeVideoPlayer\(\)](#),
[FLVPlayback.visibleVideoPlayerIndex](#)

FLVPlayback.visibleVideoPlayerIndex

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`my_FLVPlayback.visibleVideoPlayerIndex`

Description

Property; a number that you can use to manage multiple FLV file streams. Sets which video player instance is visible, audible, and controlled by the skin or playback controls, while the the rest of the video players are hidden and muted. The default is 0. It does not make the video player the target for most APIs; use the `activeVideoPlayerIndex` property instead.

Methods and properties that control dimensions interact with this property. The methods and properties that set the dimensions of the video player (`setScale()`, `setSize()`, `width`, `height`, `scaleX`, `scaleY`) can be used for all video players. However, depending on whether `autoSize` or `maintainAspectRatio` are set on those video players, they might have different dimensions. Reading the dimensions using the `width`, `height`, `scaleX`, and `scaleY` properties gives you the dimensions only of the visible video player. Other video players might have the same dimensions or might not.

To get the dimensions of various video players when they are not visible, listen for the `resize` event, and store the size value.

This property does not have any implications for visibility of the component as a whole, only which video player is visible when the component is visible. To set visibility for the entire component, use the `visible` property.

Example

The following example creates two video players to play two FLV files consecutively in a single FLVPlayback instance. It sets the `visibleVideoPlayerIndex` property to make the video players and the FLV files visible.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
// specify name and location of FLV for default player
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv"
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
    // add a second video player and specify the name and loc of its FLV
    my_FLVPlybk.activeVideoPlayerIndex = 1;
    my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
      water.flv";
    // reset to default video player, which plays its FLV automatically
    my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
    // if complete is for 2nd FLV, make default active and visible
    if (eventObject.vp == 1) {
        my_FLVPlybk.activeVideoPlayerIndex = 0;
        my_FLVPlybk.visibleVideoPlayerIndex = 0;
    } else { // make 2nd player active & visible and play FLV
        my_FLVPlybk.activeVideoPlayerIndex = 1;
        my_FLVPlybk.visibleVideoPlayerIndex = 1;
        my_FLVPlybk.play();
    }
};
// add listener for complete event
my_FLVPlybk.addEventListener("complete", listenerObject);
```

See also

[FLVPlayback.activeVideoPlayerIndex](#), [FLVPlayback.setScale\(\)](#),
[FLVPlayback.setSize\(\)](#), [FLVPlayback.height](#), [FLVPlayback.width](#),
[FLVPlayback.scaleX](#), [FLVPlayback.scaleY](#), [FLVPlayback.visible](#)

FLVPlayback.volume

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.volume

Description

Property; a number in the range of 0 to 100 that indicates the volume control setting. The default value is 100.

Example

The following example sets the initial volume to 10, a relatively low setting.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
// You can change this value from 0 to 100
my_FLVPlayback.volume = 10;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
```

See also

[FLVPlayback.transform](#), [FLVPlayback.volumeBar](#), [FLVPlayback.volumeBarInterval](#), [FLVPlayback.volumeUpdate](#)

FLVPlayback.volumeBar

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.volumeBarInterval
```

Description

Property; a MovieClip object that is the volume bar control. For more information on using FLV Playback Custom UI components for playback controls, see [“Skinning FLV Playback Custom UI components individually” on page 525](#).

Example

The following example uses the `backButton`, `forwardButton`, `playButton`, `pauseButton`, `stopButton`, and `volumeBar` properties to attach individual FLV Custom UI controls to an FLVPlayback component.

Drag an FLVPlayback component to the Stage, give it an instance name of `my_FLVPlayback`, and set the `skin` parameter to `None` in the Component inspector. Next, add the following individual FLV Custom UI components, and give them the instance names shown in parentheses: `BackButton` (`my_bkbbtn`), `ForwardButton` (`my_fwdbbtn`), `PlayButton` (`my_plybbtn`), `PauseButton` (`my_pausbbtn`), `StopButton` (`my_stopbbtn`), and `VolumeBar` (`my_vBar`). Then add the following lines of code to the Actions panel:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton,
 *   StopButton and VolumeBar components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbbtn;
my_FLVPlayback.forwardButton = my_fwdbbtn;
my_FLVPlayback.playButton = my_plybbtn;
my_FLVPlayback.pauseButton = my_pausbbtn;
my_FLVPlayback.stopButton = my_stopbbtn;
my_FLVPlayback.volumeBar = my_vBar;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
```

See also

[FLVPlayback.volume](#), [FLVPlayback.volumeBarInterval](#),
[FLVPlayback.volumeBarScrubTolerance](#), [FLVPlayback.volumeUpdate](#)

FLVPlayback.volumeBarInterval

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlayback.volumeBarInterval
```

Description

Property; a number that specifies, in milliseconds, how often to check the volume bar handle location when scrubbing. The default is 250.

Example

The following example sets the `volumeBarInterval` property to 1 second (1000 milliseconds) and creates a `volumeUpdate` event that shows the playhead time and the volume as the user drags the handle on the volume bar. The `volumeUpdate` events occur at approximately 1 second intervals because of the `volumeBarInterval` setting.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlayback`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.volumeBarInterval = 1000;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(eventObject:Object) {
    trace("Playhead time is: " + my_FLVPlayback.playheadTime);
    trace("Volume is: " + my_FLVPlayback.volume);
};
my_FLVPlayback.addEventListener("volumeUpdate", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.volume](#), [FLVPlayback.volumeBar](#),
[FLVPlayback.volumeBarScrubTolerance](#), [FLVPlayback.volumeUpdate](#)

FLVPlayback.volumeBarScrubTolerance

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
my_FLVPlaybk.volumeBarScrubTolerance
```

Description

Property; a number that specifies how far a user can move the volume bar handle before an update occurs. The value is expressed as a percentage. The default value is 5.

Example

The following example sets the `volumeBarScrubTolerance` property to 20 and creates a `volumeUpdate` event that shows the volume setting as the user drags the handle on the volume bar.

Drag an `FLVPlayback` component to the Stage, and give it an instance name of `my_FLVPlaybk`. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlaybk
 */
import mx.video.*;
my_FLVPlaybk.volumeBarScrubTolerance = 20;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(event:Object:Object) {
    trace("Playhead time is: " + my_FLVPlaybk.playheadTime);
    trace("Volume is: " + my_FLVPlaybk.volume);
};
my_FLVPlaybk.addEventListener("volumeUpdate", listenerObject);
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
    cuepoints.flv";
```

See also

[FLVPlayback.volumeBar](#), [FLVPlayback.volumeBarInterval](#)

FLVPlayback.volumeUpdate

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(eventObject:Object):Void {
    // insert event-handling code here
};
my_FLVplybk.addEventListener("volumeUpdate", listenerObject);
```

Description

Event; dispatched when the volume changes either by the user moving the handle of the volumeBar control or by setting the `volume` property in ActionScript. The event object has a `volume` property.

Example

The following example shows the value of the `volume` property in the Output panel for any adjustments that the user makes to the volume.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVplybk**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(eventObject:Object):Void {
    trace("Volume setting is: " + eventObject.volume);
};
my_FLVplybk.addEventListener("volumeUpdate", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

See also

[FLVPlayback.addEventListener\(\)](#)[FLVPlayback.volume](#), [FLVPlayback.volumeBar](#)

FLVPlayback.width

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.width

Description

Property; a number that specifies the width of the FLVPlayback instance on the Stage. This property affects only the width of the FLVPlayback instance and does not include the width of a skin SWF file that might be loaded. Use the FLVPlayback `width` property and not the `MovieClip._width` property because the `_width` property might give a different value if a skin SWF file is loaded.

Example

The following example sets the `width` and `height` properties, which causes a `resize` event because the default value of the `maintainAspectRatio` property is `true`. When the event occurs, the event handler shows the width and height of the resized FLVPlayback instance in the Output panel.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
// maintainAspectRatio (true by default) affects the actual dimensions
my_FLVPlayback.width = 400;
my_FLVPlayback.height = 350;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object) {
    trace("Width is: " + eventObject.target.width + " Height is: " +
        eventObject.target.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

See also

[FLVPlayback.height](#), [FLVPlayback.setSize\(\)](#), [FLVPlayback.preferredHeight](#), [FLVPlayback.preferredWidth](#)

FLVPlayback.x

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.x

Description

Property; a number that specifies the horizontal coordinate (location) of the video player. This property affects only the horizontal location of the FLVPlayback instance and does not include the location of a skin SWF file that, when applied, may alter the location. Use the FLVPlayback.x property, not the `MovieClip._x` property because the `_x` property might give a different value if a skin SWF file is loaded.

Example

The following example places the FLVPlayback instance 50 pixels from the left and 25 pixels from the top.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
my_FLVPlayback.x = 50;
my_FLVPlayback.y = 25;
```

See also

[FLVPlayback.y](#)

FLVPlayback.y

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

my_FLVPlayback.y

Description

Property; a number that specifies the vertical coordinate (location) of the video player. This property affects only the vertical location of the FLVPlayback instance and does not include the location of a skin SWF file that, when applied, may alter the location. Use the FLVPlayback.y property, not the MovieClip._y property because the _y property might give a different value if a skin SWF file is loaded.

Example

The following example places the FLVPlayback instance 25 pixels from the left and 50 pixels from the top.

Drag an FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlayback**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
  water.flv";
my_FLVPlayback.x = 25;
my_FLVPlayback.y = 50;
```

See also

[FLVPlayback.x](#)

VideoError class

Inheritance Error > VideoError

ActionScript class name mx.video.VideoError

The properties of the VideoError class allow you to diagnose error conditions that occur when working with the FLVPlayback component.

The mx.video.VideoError class extends the Error class.

Property summary for the VideoError class

The following table lists the properties of the VideoError class:

Property	Description
VideoError.code	A numeric error code.
VideoError.DELETE_DEFAULT_PLAYER	A number indicating an attempt to delete the default video player.
VideoError.DELETE_DEFAULT_PLAYER	A number indicating an illegal cue point.
VideoError.INVALID_CONTENT_PATH	A number indicating an invalid <code>contentPath</code> value.
VideoError.INVALID_SEEK	A number indicating an invalid seek.
VideoError.INVALID_XML	A number indicating that invalid XML was encountered in an XML file.
VideoError.NO_BITRATE_MATCH	A number indicating that a default FLV file that matches any bit rate could not be found.
VideoError.NO_CONNECTION	A number indicating that the method cannot connect to the server or find the FLV file on the server.
VideoError.NO_CUE_POINT_MATCH	A number indicating that no matching cue point was found.

VideoError.code

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`mx.video.VideoError.code`

Description

The numeric code that identifies the error condition.

Example

The following example displays the error condition in the Output panel:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    trace ("Error code is: " + err.code)
    ...
}
```

VideoError.DELETE_DEFAULT_PLAYER

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

`mx.video.VideoError.DELETE_DEFAULT_PLAYER`

Description

A value of 1007, which occurs if you call the `FLVPlayback.closeVideoPlayer()` method to attempt to close the default video player (number 0). You cannot delete the default video player.

Example

The following code checks for the `DELETE_DEFAULT_PLAYER` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == DELETE_DEFAULT_PLAYER) {
        ...
    }
}
```

See also

[FLVPlayback.activeVideoPlayerIndex](#)

VideoError.ILLEGAL_CUE_POINT

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.ILLEGAL_CUE_POINT
```

Description

A value of 1002, indicating an invalid cue point was found.

Example

The following code checks for the `ILLEGAL_CUE_POINT` error code:

```
try {
    ...
} catch (err:VideoError) {
    if (err.code == ILLEGAL_CUE_POINT) {
        ...
    }
}
```

See also

[FLVPlayback.cuePoint](#)

VideoError.INVALID_CONTENT_PATH

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.INVALID_CONTENT_PATH
```

Description

A value of 1004, indicating an invalid `contentPath` value was found.

Example

The following code checks for the `INVALID_CONTENT_PATH` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == INVALID_CONTENT_PATH) {
        ...
    }
}
```

See also

[FLVPlayback.contentPath](#)

VideoError.INVALID_SEEK

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.INVALID_SEEK
```


Description

A value of 1003, indicating an invalid seek was attempted.

Example

The following code checks for the `INVALID_SEEK` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == INVALID_SEEK) {
        ...
    }
}
```

See also

[FLVPlayback.seek\(\)](#)

VideoError.INVALID_XML

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.INVALID_XML
```

Description

A value of 1005, indicating invalid XML was encountered. An invalid XML error can occur when downloading and parsing a SMIL file. The `VideoError.message` property contains text that describes the precise problem. For more information, see [“Using a SMIL file” on page 712](#).

Example

The following code checks for the `INVALID_XML` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == INVALID_XML) {
        ...
    }
}
```

See also

[FLVPlayback.contentPath](#)

VideoError.NO_BITRATE_MATCH

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.NO_BITRATE_MATCH
```

Description

A value of 1006, which indicates that there is no default FLV file listed that matches any bit rate. Occurs only when downloading and parsing a SMIL file. For more information, see [“Using a SMIL file” on page 712](#).

Example

The following code checks for the `NO_BITRATE_MATCH` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == NO_BITRATE_MATCH) {
        ...
    }
}
```

See also

[FLVPlayback.bitrate](#)

VideoError.NO_CONNECTION

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.NO_CONNECTION
```

Description

A value of 1000, indicating the method cannot connect to the server or find the FLV file on the server.

Example

The following code checks for the `NO_CONNECTION` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == NO_CONNECTION) {
        ...
    }
}
```

VideoError.NO_CUE_POINT_MATCH

Availability

Flash Player 8.

Edition

Flash Professional 8.

Usage

```
mx.video.VideoError.NO_CUE_POINT_MATCH
```

Description

A value of 1001, indicating that no matching cue point was found.

Example

The following code checks for the `NO_CUE_POINT_MATCH` error code:

```
import mx.video.*;

try {
    ...
} catch (err:VideoError) {
    if (err.code == NO_CUE_POINT_MATCH) {
        ...
    }
}
```

See also

[FLVPlayback.findCuePoint\(\)](#)

VideoPlayer class

Inheritance MovieClip > VideoPlayer class

ActionScript Class Name mx.video.VideoPlayer

VideoPlayer extends the MovieClip class and wraps a Video object.

The FLVPlayback class wraps the VideoPlayer class and Macromedia strongly encourages you to use the FLVPlayback class in almost all cases. There is no functionality in the VideoPlayer class that cannot be accessed using the FLVPlayback class.

The VideoPlayer class is included here because it allows you to create a video player with a smaller SWF file. The VideoPlayer class does not allow you to include a skin or playback controls, and it has a smaller API. You cannot find or seek to cue points, for example, although `cuePoint` events will occur.

In addition, the FLVPlayback class automatically interfaces with the NCManager class to access streaming FLV files on a FCS, for example. You interact with the NCManager class when you set the `contentPath` property and when you pass a URL to the `play()` and `load()` methods. If you use the VideoPlayer class by itself, however, you must include the following statement in your ActionScript code to make sure the NCManager class is included:

```
_forceNCManager:mx.video.NCManager;
```

The NCManager class also has an interface class, INCManager, which allows you to replace the NCManager class with a custom class for managing network communications. If you do that, you also need to include the following statement, replacing `NCManager` with the name of the class you have provided:

```
mx.video.VideoPlayer.DEFAULT_INCMANAGER = "mx.video.NCManager";
```

You do not need to add this statement if you are using the default NCManager class.

NOTE

You also can set `DEFAULT_INCMANAGER` to replace the default `mx.video.NCManager` with the `FLVPlayback` component.

To handle multiple streams for multiple bandwidths, NCManager supports a subset of SMIL. For more information, see [“Using a SMIL file” on page 712](#).

This section provides a summary of the VideoPlayer class. You can find detailed documentation of the methods, properties, and events of the VideoPlayer class at www.macromedia.com/go/videoplayer.

Method summary for the VideoPlayer class

The following table lists the methods of the VideoPlayer class:

Method	Description
<code>VideoPlayer.addEventListener()</code>	Creates a listener for a specified event.
<code>VideoPlayer.close()</code>	Closes the video stream and FCS connection.
<code>VideoPlayer.load()</code>	Loads the FLV file but does not begin playing. After resizing (if needed) the FLV file is paused.
<code>VideoPlayer.pause()</code>	Pauses playing the video stream.
<code>VideoPlayer.play()</code>	Begins playing the video stream.
<code>VideoPlayer.removeEventListener()</code>	Removes an event listener.
<code>VideoPlayer.seek()</code>	Seeks to a specified time in the file, given in seconds, with decimal precision up to milliseconds.
<code>VideoPlayer.setScale()</code>	Sets <code>scaleX</code> and <code>scaleY</code> simultaneously.
<code>VideoPlayer.setSize()</code>	Sets <code>width</code> and <code>height</code> simultaneously.
<code>VideoPlayer.stop()</code>	Stops playing the video stream.

Property summary for the VideoPlayer class

The VideoPlayer class has class and instance properties.

Class properties

The following properties occur only for the VideoPlayer class. They are read-only constants that apply to all instances of the VideoPlayer class.

Property	Value	Description
<code>VideoPlayer.BUFFERING</code>	"buffering"	Possible value for the <code>state</code> property. Indicates state entered immediately after <code>play()</code> or <code>load()</code> is called.
<code>VideoPlayer.CONNECTION_ERROR</code>	"connectionError"	Possible value for the <code>state</code> property. Indicates that a connection error occurred.

Property	Value	Description
<code>VideoPlayer.DEFAULT_INCMANAGER</code>	<code>"mx.video.NCManager"</code>	Name of the default (<code>mx.video.NCManager</code>) or custom implementation of the <code>INCManger</code> interface.
<code>VideoPlayer.DISCONNECTED</code>	<code>"disconnected"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file stream is disconnected.
<code>VideoPlayer.LOADING</code>	<code>"loading"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is loading.
<code>VideoPlayer.PAUSED</code>	<code>"paused"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is paused.
<code>VideoPlayer.PLAYING</code>	<code>"playing"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is playing.
<code>VideoPlayer.RESIZING</code>	<code>"resizing"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is resizing.
<code>VideoPlayer.REWINDING</code>	<code>"rewinding"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is rewinding.
<code>VideoPlayer.SEEKING</code>	<code>"seeking"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is seeking.
<code>VideoPlayer.STOPPED</code>	<code>"stopped"</code>	Possible value for the <code>state</code> property. Indicates that the FLV file is stopped.
<code>VideoPlayer.version</code>	<code>x.x.x.xx</code>	A number that is the component's version number.

Instance Properties

The following table lists the instance properties of the `VideoPlayer` class. This set of properties applies to each instance of a `VideoPlayer` class.

Property	Description
<code>VideoPlayer.autoRewind</code>	A Boolean value that, if <code>true</code> , causes the FLV file to rewind to the first frame when play stops.
<code>VideoPlayer.autoSize</code>	A Boolean value that, if <code>true</code> , causes the video to automatically size to the source dimensions.
<code>VideoPlayer.bufferTime</code>	A number that specifies the number of seconds to buffer in memory before beginning to play a video stream.
<code>VideoPlayer.bytesLoaded</code>	A number that indicates the extent of downloading in number of bytes for an HTTP download. Read-only.
<code>VideoPlayer.bytesTotal</code>	A number that specifies the total number of bytes downloaded for an HTTP download. Read-only.
<code>VideoPlayer.connected</code>	A Boolean value that indicates whether the FLV file stream is connected. Read-only.
<code>VideoPlayer.height</code>	A number that specifies the height of the video in pixels.
<code>VideoPlayer.idleTimeout</code>	The amount of time, in milliseconds, before an idle FCS connection (because playing is paused or stopped) is terminated.
<code>VideoPlayer.isLive</code>	A Boolean value that is <code>true</code> if the video stream is live. Not applicable to HTTP download.
<code>VideoPlayer.isRTMP</code>	A Boolean value that is <code>true</code> if the FLV file is streaming from FCS. Read-only.
<code>VideoPlayer.maintainAspectRatio</code>	A Boolean value that, if <code>true</code> , maintains the video aspect ratio.
<code>VideoPlayer.metadata</code>	An object that is a metadata information packet that is received from a call to the <code>onMetaData()</code> callback function, if available. Read-only.
<code>VideoPlayer.ncMgr</code>	An <code>INCMManager</code> object that provides access to an instance of the class implementing <code>INCMManager</code> .
<code>VideoPlayer.playheadTime</code>	A number that is the current playhead time or position, measured in seconds, which can be a fractional value.

Property	Description
<code>VideoPlayer.playheadUpdateInterval</code>	A number that is the amount of time, in milliseconds, between each <code>playheadUpdate</code> event.
<code>VideoPlayer.progressInterval</code>	A number that is the amount of time, in milliseconds, between each <code>progress</code> event.
<code>VideoPlayer.scaleX</code>	A number that specifies the horizontal scale.
<code>VideoPlayer.scaleY</code>	A number that specifies the vertical scale.
<code>VideoPlayer.state</code>	A string that specifies the state of the component. Set with the <code>load()</code> , <code>play()</code> , <code>stop()</code> , <code>pause()</code> and <code>seek()</code> methods. Read-only.
<code>VideoPlayer.stateResponsive</code>	A Boolean value that is <code>true</code> if the state is responsive (that is, if controls can be enabled in the current state). Read-only.
<code>VideoPlayer.totalTime</code>	A number that is the total playing time for the video.
<code>VideoPlayer.transform</code>	An object that provides direct access to the <code>Sound.setTransform()</code> and <code>Sound.getTransform()</code> methods to provide more sound control.
<code>VideoPlayer.url</code>	A string that specifies the URL of the loaded (or loading) stream.
<code>VideoPlayer.videoHeight</code>	A number that specifies the height of the FLV file.
<code>VideoPlayer.videoWidth</code>	A number that specifies the width of the FLV file.
<code>VideoPlayer.visible</code>	A Boolean value that, if <code>true</code> , makes the FLV file visible.
<code>VideoPlayer.volume</code>	A number in the range of 0 to 100 that indicates the volume control setting.
<code>VideoPlayer.width</code>	A number (percentage) that specifies how far a user can move the volume bar handle before an update occurs.
<code>VideoPlayer.x</code>	A number that specifies the horizontal dimension in pixels of the video player.
<code>VideoPlayer.y</code>	A number that specifies the vertical dimension in pixels of the video player.

Event summary for the VideoPlayer class

The following table lists the events of the VideoPlayer class:

Event	Description
<code>VideoPlayer.close</code>	Dispatched when the video stream is closed, whether through timeout or a call to the <code>close()</code> method.
<code>VideoPlayer.complete</code>	Dispatched when playing completes by reaching the end of the FLV file.
<code>VideoPlayer.cuePoint</code>	Dispatched when a cue point is reached.
<code>VideoPlayer.metadataReceived</code>	Dispatched the first time the FLV file metadata is reached.
<code>VideoPlayer.playheadUpdate</code>	Dispatched every .25 seconds while the FLV file is playing.
<code>VideoPlayer.progress</code>	Dispatched every .25 seconds, starting when the <code>load()</code> method is called and ending when all bytes are loaded or there is a network error.
<code>VideoPlayer.ready</code>	Dispatched when the FLV file is loaded and ready to display.
<code>VideoPlayer.resize</code>	Dispatched when the video is resized.
<code>VideoPlayer.rewind</code>	Dispatched when the location of the playhead is moved backward by a call to <code>seek()</code> or when the automatic rewind operation completes.
<code>VideoPlayer.stateChange</code>	Dispatched when the playback state changes.

Using a SMIL file

To handle multiple streams for multiple bandwidths, the `VideoPlayer` class uses a helper class (`NCManager`) that supports a subset of SMIL. SMIL is used to identify the location of the video stream, the layout (width and height) of the FLV file, and the source FLV files that correspond to the different bandwidths. It can also be used to specify the bit rate and duration of the FLV file.

The following example shows a SMIL file that streams multiple bandwidth FLV files from a FCS using RTMP:

```
<smil>
  <head>
    <meta base="rtmp://myserver/mypgm/" >
    <layout>
      <root-layout width="240" height="180" >
    </layout>
  </head>
  <body>
    <switch>
      <video src="myvideo_mdm.flv" system-bitrate="56000"
dur="3:00.1">
      <video src="myvideo_isdn.flv" system-bitrate="128000"
dur="3:00.1">
      <ref src="myvideo_cable.flv" dur="3:00.1"/>
    </switch>
  </body>
</smil>
```

The `<head>` tag may contain the `<meta>` and `<layout>` tags. The `<meta>` tag supports only the `base` attribute, which is used to specify the URL of the streaming video (RTMP from a FCS).

The `<layout>` tag supports only the `root-layout` element, which is used to set the `height` and `width` attributes, and, therefore, determines the size of the window in which the FLV file is rendered. These attributes accept only pixel values, not percentages.

Within the body of the SMIL file, you can either include a single link to a FLV source file or, if you're streaming multiple files for multiple bandwidths from a FCS (as in the previous example), you can use the `<switch>` tag to list the source files.

The `video` and `ref` tags within the `<switch>` tag are synonymous—they both can use the `src` attribute to specify FLV files. Further, each can use the `region`, `system-bitrate`, and `dur` attributes to specify the region, the minimum bandwidth required, and the duration of the FLV file.

Within the `<body>` tag, only one occurrence of either the `<video>`, `<src>`, or `<switch>` tags is allowed.

The following example shows a progressive download for a single FLV file that does not use bandwidth detection:

```
<smil>
  <head>
    <layout>
      <root-layout width="240" height="180" />
    </layout>
  </head>
  <body>
    <video src="myvideo.flv" />
  </body>
</smil>
```

<smil>

Availability

Flash Professional 8.

Usage

```
<smil>
```

...

```
child tags
```

...

```
</smil>
```

Attributes

None.

Child tags

<head>, <body>

Parent tag

None.

Description

Top level tag, which identifies a SMIL file.

Example

The following example shows a SMIL file specifying three FLV files:

```
<smil>
  <head>
    <meta base="rtmp://myserver/mypgm/" >
    <layout>
      <root-layout width="240" height="180" >
    </layout>
  </head>
  <body>
    <switch>
      <video src="myvideo_mdm.flv" system-bitrate="56000"
dur="3:00.1">
      <video src="myvideo_isdn.flv" system-bitrate="128000"
dur="3:00.1">
      <ref src="myvideo_cable.flv" dur="3:00.1"/>
    </switch>
  </body>
</smil>
```

<head>

Availability

Flash Professional 8.

Usage

```
<head>
...
child tags
...
</head>
```

Attributes

None.

Child tags

<meta>, <layout>

Parent tag

<smil>

Description

Supporting the `<meta>` and `<layout>` tags, specifies the location and default layout (height and width) of the source FLV files.

Example

The following example sets the root layout to 240 pixels by 180 pixels:

```
<head>
  <meta base="rtmp://myserver/mypgm/" >
  <layout>
    <root-layout width="240" height="180" >
  </layout>
</head>
```

<meta>

Availability

Flash Professional 8.

Usage

```
<meta/>
```

Attributes

base

Child tags

```
<layout>
```

Parent tag

None.

Description

Contains the `base` attribute which specifies the location (RTMP URL) of the source FLV files.

Example

The following example shows a meta tag for a base location on myserver:

```
<meta base="rtmp://myserver/mypgm/" >
```

<layout>

Availability

Flash Professional 8.

Usage

```
<layout>  
...  
  child tags  
...  
</layout>
```

Attributes

None.

Child tags

```
<root-layout>
```

Parent tag

```
<meta>
```

Description

Specifies the width and height of the FLV file.

Example

The following example specified the layout of 240 pixels by 180 pixels:

```
<layout>  
  <root-layout width="240" height="180" >  
</layout>
```

<root-layout>

Availability

Flash Professional 8.

Usage

```
<root-layout...attributes.../>
```

Attributes

Width, height

Child tags

None.

Parent tag

<layout>

Description

Specifies the width and height of the FLV file.

Example

The following example specified the layout of 240 pixels by 180 pixels:

```
<root-layout width="240" height="180" >
```

<body>

Availability

Flash Professional 8.

Usage

```
<body>
```

```
...
```

```
child tags
```

```
...
```

```
</body>
```

Attributes

None.

Child tags

<video>, <ref>, <switch>

Parent tag

<smil>

Description

Contains the <video>, <ref>, and <switch> tags, which specify the name of the source FLV file, the minimum bandwidth, and the duration of the FLV file. The `system-bitrate` attribute is supported only when using the <switch> tag. Within the <body> tag, only one instance of either <switch>, <video>, or <ref> tags is allowed.

Example

The following example specified three FLV files, two using the `video` tag, and one using the `ref` tag:

```
<body>
  <switch>
    <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">
    <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1">
    <ref src="myvideo_cable.flv" dur="3:00.1"/>
  </switch>
</body>
```

<video>

Availability

Flash Professional 8.

Usage

```
<video...attributes.../>
```

Attributes

`src`, `system-bitrate`, `dur`

Child tags

None

Parent tag

```
<body>
```

Description

Synonymous with the `<ref>` tag. Supports the `src` and `dur` attributes, which specify the name of the source FLV file and its duration. The `dur` attribute supports the full (00:03:00:01) and partial (03:00:01) time formats.

Example

The following example sets the source and duration for a video:

```
<video src="myvideo_mdm.flv" dur="3:00.1">
```

<ref>

Availability

Flash Professional 8.

Usage

```
<ref...attributes.../>
```

Attributes

src, system-bitrate, dur

Child tags

None

Parent tag

<body>

Description

Synonymous with <video> tag. Supports the src and dur attributes, which specify the name of the source FLV file and its duration. The dur attribute supports the full (00:03:00:01) and partial (03:00:01) time formats.

Example

The following example sets the source and duration for a video:

```
<ref src="myvideo_cable.flv" dur="3:00.1"/>
```

<switch>

Availability

Flash Professional 8.

Usage

```
<switch>  
...  
child tags  
...  
</switch/>
```

Attributes

None

Child tags

<video>, <ref>

Parent tag

<body>

Description

Used with either the <video> or <ref> child tags to list the FLV files for multiple bandwidth video streaming. The <switch> tag supports the `system-bitrate` attribute, which specifies the minimum bandwidth as well as the `src` and `dur` attributes.

Example

The following example specified three FLV files, two using the `video` tag, and one using the `ref` tag:

```
<switch>  
  <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">  
  <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1">  
  <ref src="myvideo_cable.flv" dur="3:00.1"/>  
</switch>
```


You can use the Focus Manager class to specify the order in which components receive focus when a user presses the Tab key to navigate in an application. You can also use the Focus Manager to set a button in your document that receives keyboard input when a user presses Enter (Windows) or Return (Macintosh). For example, when users fill out a form, they should be able to tab between fields and press Enter (Windows) or Return (Macintosh) to submit the form.

All components implement Focus Manager support; you don't need to write code to invoke the FocusManager class.

NOTE

Focus Manager support overrides the use of the `on(keyPress)` global handler. Because all components implement Focus Manager, an application that includes components *and* uses the `on(keyPress)` global handler needs to have a `tabIndex` for every control (including components *and* movie clips) set, explicitly (see [“Using Focus Manager” on page 722](#)). Or, preferably, you can add an event listener for a specific Key and the Focus Manager will not override the corresponding event handler. For more information about creating an event listener for a Key, see [“Capturing keypresses” in *Learning ActionScript 2.0 in Flash*](#).

The Focus Manager interacts with the System Manager, which activates and deactivates FocusManager instances as pop-up windows are activated or deactivated. Each modal window has an instance of FocusManager so the components in that window become their own tab set, preventing the user from tabbing into components in other windows.

The Focus Manager recognizes groups of radio buttons (those with a defined `RadioButton.groupName` property) and sets focus to the instance in the group that has a `selected` property that is set to `true`. When the Tab key is pressed, the Focus Manager checks to see if the next object has the same group name as the current object. If it does, it automatically moves focus to the next object with a different group name. Other sets of components that support a `groupName` property can also use this feature.

The Focus Manager handles focus changes caused by mouse clicks. If the user clicks a component, that component is given focus.

NOTE

When testing a script using Focus Manager (Control > Test Movie), select Control > Disable Keyboard Shortcuts in test mode; otherwise, Focus Manager does not appear to work. Also, tabbing and keyboard shortcuts are used by the authoring environment by default. So, if you use test mode, the tab navigation, Enter key, and other key combinations may perform in unexpected ways or appear to fail. Those features should be tested in the Player outside the authoring environment.

Using Focus Manager

The Focus Manager does not automatically assign focus to a component. You must write a script that calls `FocusManager.setFocus()` on a component if you want a component to have focus when an application loads.

NOTE

If you call `FocusManager.setFocus()` to set focus to a component when an application loads, the focus ring does not appear around that component. The component has focus, but the indicator is not present.

To create focus navigation in an application, set the `tabIndex` property on any objects (including buttons) that should receive focus. When a user presses the Tab key, the Focus Manager looks for an enabled object with a `tabIndex` property that is higher than the current value of `tabIndex`. Once the Focus Manager reaches the highest `tabIndex` property, it returns to zero. So, in the following example, the `comment` object (probably a `TextArea` component) receives focus first, and then the `okButton` object receives focus:

```
comment.tabIndex = 1;  
okButton.tabIndex = 2;
```

You can also use the Accessibility panel to assign a tab index value.

If nothing on the Stage has a tab index value, the Focus Manager uses the *depth* (stacking order, or *z-order*). The depth is set up primarily by the order in which components are dragged to the Stage; however, you can also use the Modify > Arrange > Bring to Front/Send to Back commands to determine the final depth.

To create a button that receives focus when a user presses Enter (Windows) or Return (Macintosh), set the `FocusManager.defaultPushButton` property to the instance name of the desired button, as shown here:

```
focusManager.defaultPushButton = okButton;
```

NOTE

The Focus Manager is sensitive to when objects are placed on the Stage (the depth order of objects) and not their relative positions on the Stage. This is different from the way Flash Player handles tabbing.

Using Focus Manager to allow tabbing

You can use the Focus Manager to create a scheme that allows users to press the Tab key to cycle through objects in a Flash application. (Objects in the tab scheme are called *tab targets*.) The Focus Manager examines the `tabEnabled` and `tabChildren` properties of the objects' parents in order to locate the objects.

A movie clip can be either a container of tab targets, a tab target itself, or neither:

Movie clip type	tabEnabled	tabChildren
Container of tab targets	false	true
Tab target	true	false
Neither	false	false

NOTE

This is different from the default Flash Player behavior, in which a container's `tabChildren` property can be undefined.

Consider the following scenario. On the Stage of the main timeline are two text fields (`txt1` and `txt2`) and a movie clip (`mc`) that contains a `DataGrid` component (`grid1`) and another text field (`txt3`). You would use the following code to allow users to press Tab and cycle through the objects in the following order: `txt1`, `txt2`, `grid1`, `txt3`.

NOTE

The `FocusManager` and `TextField` instances are enabled by default.

```

// Let Focus Manager know mc has children;
// this overrides mc.focusEnabled=true;
mc.tabChildren=true;
mc.tabEnabled=false;
// Set the tabbing sequence.
txt1.tabIndex = 1;
txt2.tabIndex = 2;
mc.grid1.tabIndex = 3;
mc.txt3.tabIndex = 4;

// Set initial focus to txt1.
txt1.text = "focus";
focusManager.setFocus(txt1);

```

If your movie clip doesn't have an `onPress` or `onRelease` method or a `tabEnabled` property, it won't be seen by the Focus Manager unless you set `focusEnabled` to `true`. Input text fields are always in the tab scheme unless they are disabled.

If a Flash application is playing in a web browser, the application doesn't have focus until a user clicks somewhere in the application. Also, once a user clicks in the Flash application, pressing `Tab` can cause focus to jump outside the Flash application. To keep tabbing limited to objects inside the Flash application in Flash Player 7 ActiveX control, add the following parameter to the HTML `<object>` tag:

```
<param name="SeamlessTabbing" value="false"/>
```

Creating an application with Focus Manager

The following procedure creates a focus scheme in a Flash application.

To create a focus scheme:

1. Drag the `TextInput` component from the Components panel to the Stage.
2. In the Property inspector, assign it the instance name `comment`.
3. Drag the `Button` component from the Components panel to the Stage.
4. In the Property inspector, assign it the instance name `okButton` and set the label parameter to `OK`.
5. In Frame 1 of the Actions panel, enter the following:

```

comment.tabIndex = 1;
okButton.tabIndex = 2;
focusManager.setFocus(comment);
function click(evt){
    trace(evt.type);
}
okButton.addEventListener("click", this);

```


6. Select Control > Test Movie.
7. Select Control > Disable Keyboard Shortcuts.

The code sets the tab ordering. Although the comment field doesn't have an initial focus ring, it has initial focus, so you can start typing in the comment field without clicking in it. Also, you have to select the Disable Keyboard Shortcuts menu option for focus to work properly in test mode.

Customizing Focus Manager

You can change the color of the focus ring in the Halo theme by changing the value of the `themeColor` style, as in this example:

```
_global.style.setStyle("themeColor", "haloBlue");
```

The Focus Manager uses a `FocusRect` skin for drawing focus. This skin can be replaced or modified and subclasses can override `UIComponent.drawFocus` to draw custom focus indicators.

FocusManager class (API)

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `FocusManager`

ActionScript Class Name `mx.managers.FocusManager`

You can use the Focus Manager to specify the order in which components receive focus when a user presses the Tab key to navigate in an application. You can also use the `FocusManager` class to set a button in your document that receives keyboard input when a user presses Enter (Windows) or Return (Macintosh).

TIP

In a class file that inherits from `UIComponent`, it is not good practice to refer to `_root.focusManager`. Every `UIComponent` instance inherits a `getFocusManager()` method, which returns a reference to the `FocusManager` instance responsible for controlling that component's focus scheme.

Method summary for the FocusManager class

The following table lists the methods of the FocusManager class.

Method	Description
<code>FocusManager.getFocus()</code>	Returns a reference to the object that has focus.
<code>FocusManager.sendDefaultPushButtonEvent()</code>	Sends a <code>click</code> event to listener objects registered to the default push button.
<code>FocusManager.setFocus()</code>	Sets focus to the specified object.

Methods inherited from the UIObject class

The following table lists the methods the FocusManager class inherits from the UIObject class.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `FocusManager` class inherits from the `UIComponent` class.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `FocusManager` class

The following table lists the properties of the `FocusManager` class.

Property	Description
<code>FocusManager.defaultPushButton</code>	The object that receives a <code>click</code> event when a user presses the Return or Enter key.
<code>FocusManager.defaultPushButtonEnabled</code>	Indicates whether keyboard handling for the default push button is turned on (<code>true</code>) or off (<code>false</code>). The default value is <code>true</code> .
<code>FocusManager.enabled</code>	Indicates whether tab handling is turned on (<code>true</code>) or off (<code>false</code>). The default value is <code>true</code> .
<code>FocusManager.nextTabIndex</code>	The next value of the <code>tabIndex</code> property.

Properties inherited from the `UIObject` class

The following table lists the properties the `FocusManager` class inherits from the `UIObject` class.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.

Property	Description
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `FocusManager` class inherits from the `UIComponent` class.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `FocusManager` class

There are no events exclusive to the `FocusManager` class.

Events inherited from the `UIObject` class

The following table lists the events the `FocusManager` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the FocusManager class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

FocusManager.defaultPushButton

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.defaultPushButton
```

Description

Property; specifies the default push button for an application. When the user presses Enter (Windows) or Return (Macintosh), the listeners of the default push button receive a `click` event. The default value is `undefined` and the data type of this property is `object`.

The Focus Manager uses the emphasized style declaration of the `SimpleButton` class to visually indicate the current default push button.

The value of the `defaultPushButton` property is always the button that has focus. Setting the `defaultPushButton` property does not give initial focus to the default push button. If there are several buttons in an application, the button that currently has focus receives the `click` event when Enter or Return is pressed. If some other component has focus when Enter or Return is pressed, the `defaultPushButton` property is reset to its original value.

Example

The following code sets the default push button to the `OKButton` instance:

```
focusManager.defaultPushButton = OKButton;
```

See also

[FocusManager.defaultPushButtonEnabled](#),
[FocusManager.sendDefaultPushButtonEvent\(\)](#)

FocusManager.defaultPushButtonEnabled

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
focusManager.defaultPushButtonEnabled
```

Description

Property; a Boolean value that determines if keyboard handling of the default push button is turned on (`true`) or not (`false`). Setting `defaultPushButtonEnabled` to `false` allows a component to receive the Return or Enter key and handle it internally. You must re-enable default push button handling by watching the component's `onKillFocus()` method (see `onKillFocus` (MovieClip.onKillFocus handler) in *ActionScript 2.0 Language Reference*) or `focusOut` event. The default value is `true`.

This property is for use by advanced component developers.

Example

The following code disables default push button handling:

```
focusManager.defaultPushButtonEnabled = false;
```

FocusManager.enabled

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
focusManager.enabled
```

Description

Property; a Boolean value that determines if tab handling is turned on (`true`) or not (`false`) for a particular group of focus objects. (For example, another pop-up window could have its own Focus Manager.) Setting `enabled` to `false` allows a component to receive the tab handling keys and handle them internally. You must re-enable the Focus Manager handling by watching the component's `onKillFocus()` method (see `onKillFocus` (`MovieClip.onKillFocus` handler) in *ActionScript 2.0 Language Reference*) or `focusOut` event. The default value is `true`.

Example

The following code disables tabbing:

```
focusManager.enabled = false;
```

FocusManager.getFocus()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.getFocus()
```

Parameters

None.

Returns

A reference to the object that has focus.

Description

Method; returns a reference to the object that currently has focus.

Example

The following code sets the focus to `myOKButton` if the object that currently has focus is `myInputText`:

```
if (focusManager.getFocus() == myInputText)
{
    focusManager.setFocus(myOKButton);
}
```

See also

[FocusManager.setFocus\(\)](#)

FocusManager.nextTabIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`FocusManager.nextTabIndex`

Description

Property; the next available tab index number. Use this property to dynamically set an object's `tabIndex` property.

Example

The following code gives the `mycheckbox` instance the next highest `tabIndex` value:

```
mycheckbox.tabIndex = focusManager.nextTabIndex;
```

See also

[UIComponent.tabIndex](#)

FocusManager.sendDefaultPushButtonEvent()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.sendDefaultPushButtonEvent()
```

Parameters

None.

Returns

Nothing.

Description

Method; sends a `click` event to listener objects registered to the default push button. Use this method to programmatically send a `click` event.

Example

The following code triggers the default push button `click` event and fills in the user name and password fields when a user selects the `CheckBox` instance `chb` (the check box would be labeled “Automatic Login”):

```
name_txt.tabIndex = 1;
password_txt.tabIndex = 2;
chb.tabIndex = 3;
submit_ib.tabIndex = 4;

focusManager.defaultPushButton = submit_ib;

chbObj = new Object();
chbObj.click = function(o){
    if (chb.selected == true){
        name_txt.text = "Jody";
        password_txt.text = "foobar";
        focusManager.sendDefaultPushButtonEvent();
    } else {
        name_txt.text = "";
        password_txt.text = "";
    }
}
chb.addEventListener("click", chbObj);

submitObj = new Object();
submitObj.click = function(o){
    if (password_txt.text != "foobar"){
        trace("error on submit");
    } else {
        trace("Yeah! sendDefaultPushButtonEvent worked!");
    }
}
submit_ib.addEventListener("click", submitObj);
```

See also

[FocusManager.defaultPushButton](#)

FocusManager.setFocus()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.setFocus(object)
```

Parameters

object A reference to the object to receive focus.

Returns

Nothing.

Description

Method; sets focus to the specified object. If the object to which you want to set focus is not on the main timeline, use the following code:

```
_root.focusManager.setFocus(object);
```

Example

The following code sets focus to myOKButton:

```
focusManager.setFocus(myOKButton);
```

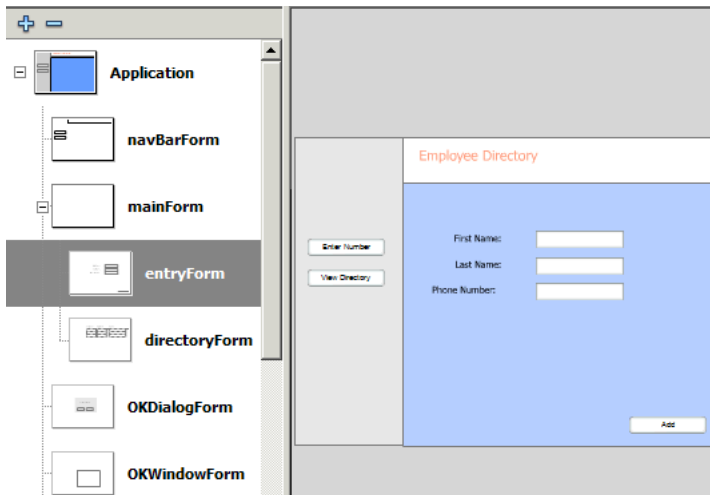
See also

[FocusManager.getFocus\(\)](#)

The Form class provides the runtime behavior of forms that you create in the Screen Outline pane in Flash. For an overview of working with screens, see “Working with Screens (Flash Professional Only)” in *Using Flash*.

Using the Form class (Flash Professional only)

Forms function as containers for graphic objects—user interface elements in an application, for example—as well as application states. You can use the Screen Outline pane to visualize the different states of an application that you’re creating, where each form is a different application state. For example, the following illustration shows the Screen Outline pane for an example application designed using forms.



Screen Outline view of sample form application

This illustration shows the outline for a sample application called Employee Directory, which consists of several forms. The form named `entryForm` (selected in the above illustration) contains several user interface objects, including input text fields, labels, and a push button. The developer can easily present this form to the user by toggling its visibility (using the `Form.visible` property), while simultaneously toggling the visibility of other forms, as well.

Using the Behaviors panel you can also attach behaviors and controls to forms. For more information about adding transitions and controls to screens, see “Creating controls and transitions for screens with behaviors (Flash Professional only)” in *Using Flash*.

Because the Form class extends the Loader class, you can easily load external content (a SWF or JPEG file) into a form. For example, the contents of a form could be a separate SWF file, which itself might contain forms. In this way, you can make your form applications modular, which makes maintaining the applications easier, and also reduces initial download time. For more information, see “Loading external content into screens (Flash Professional only)” on page 1072.

Form parameters

You can set the following authoring parameters for each Form instance in the Property inspector or in the Component inspector:

autoload indicates whether the content specified by the `contentPath` parameter should load automatically (`true`), or wait to load until the `Loader.load()` method is called (`false`). The default value is `true`.

contentPath specifies the contents of the form. This can be the linkage identifier of a movie clip or an absolute or relative URL for a SWF or JPEG file to load into the slide. By default, loaded content is clipped to fit the slide.

visible specifies whether the form is visible (`true`) or not (`false`) when it first loads.

Form class (Flash Professional only)

Inheritance MovieClip > UIObject class > UIComponent class > View > Loader component > Screen class (Flash Professional only) > Form

ActionScript Class Name mx.screens.Form

The Form class provides the runtime behavior of forms that you create in the Screen Outline pane in Flash.

Method summary for the Form class

The following table lists methods of the Form class.

Method	Description
<code>Form.getChildForm()</code>	Returns the child form at a specified index.

Methods inherited from the UIObject class

The following table lists the methods the Form class inherits from the UIObject class. When calling these methods from the Form object, use the syntax *formInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Form class inherits from the UIComponent class. When calling these methods from the Form object, use the syntax *formInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the Loader class

The following table lists the methods the Form class inherits from the Loader class. When calling these methods from the Form object, use the syntax *formInstance.methodName*.

Method	Description
Loader.load()	Loads the content specified by the <code>contentPath</code> property.

Methods inherited from the Screen class

The following table lists the methods the Form class inherits from the Screen class. When calling these methods from the Form object, use the syntax *formInstance.methodName*.

Method	Description
Screen.getChildScreen()	Returns the child screen of this screen at a particular index.

Property summary for the Form class

The following table lists the properties that are exclusive to the Form class.

Property	Description
Form.currentFocusedForm	Read-only; returns the form that contains the global current focus.
Form.indexInParentForm	Read-only; returns the index (zero-based) of this form in its parent's list of subforms.
Form.numChildForms	Read-only; returns the number of child forms that this form contains.
Form.parentIsForm	Read-only; specifies whether the parent object of this form is also a form.
Form.parentForm	Read-only; reference to the form's parent form.
Form.rootForm	Read-only; returns the root of the form tree, or subtree, that contains the form.
Form.visible	Specifies whether the form is visible when its parent form, slide, movie clip, or SWF file is visible.

Properties inherited from the UIObject class

The following table lists the properties the Form class inherits from the UIObject class. When accessing these properties from the Form object, use the syntax

formInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Form class inherits from the UIComponent class.

When accessing these properties from the Form object, use the syntax

formInstance.propertyName.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the Loader class

The following table lists the properties the Form class inherits from the Loader class. When accessing these properties from the Form object, use the syntax

formInstance.propertyName.

Property	Description
<code>Loader.autoLoad</code>	A Boolean value that indicates whether the content loads automatically (<code>true</code>) or you must call <code>Loader.load()</code> (<code>false</code>).
<code>Loader.bytesLoaded</code>	A read-only property that indicates the number of bytes that have been loaded.
<code>Loader.bytesTotal</code>	A read-only property that indicates the total number of bytes in the content.
<code>Loader.content</code>	A reference to the content of the loader. This property is read-only.
<code>Loader.contentPath</code>	A string that indicates the URL of the content to be loaded.
<code>Loader.percentLoaded</code>	A number that indicates the percentage of loaded content. This property is read-only.
<code>Loader.scaleContent</code>	A Boolean value that indicates whether the content scales to fit the loader (<code>true</code>), or the loader scales to fit the content (<code>false</code>).

Properties inherited from the Screen class

The following table lists the properties that the Form class inherits from the Screen class.

When accessing these properties from the Form object, use the syntax

formInstance.propertyName.

Property	Description
<code>Screen.currentFocusedScreen</code>	Read-only; returns the screen that contains the global current focus.
<code>Screen.indexInParent</code>	Read-only; returns the screen's index (zero-based) in its parent screen's list of child screens.
<code>Screen.numChildScreens</code>	Read-only; returns the number of child screens contained by the screen.
<code>Screen.parentIsScreen</code>	Read-only; returns a Boolean (<code>true</code> or <code>false</code>) value that indicates whether the screen's parent object is itself a screen.
<code>Screen.rootScreen</code>	Read-only; returns the root screen of the tree or subtree that contains the screen.

Event summary for the Form class

There are no events exclusive to the Form class.

Events inherited from the UIObject class

The following table lists the events the Form class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Form class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the Loader class

The following table lists the events the Form class inherits from the Loader class.

Event	Description
<code>Loader.complete</code>	Triggered when the content finished loading.
<code>Loader.progress</code>	Triggered while content is loading.

Events inherited from the Screen class

The following table lists the events the Form class inherits from the Screen class.

Event	Description
Screen.allTransitionsInDone	Broadcast when all “in” transitions applied to a screen have finished.
Screen.allTransitionsOutDone	Broadcast when all “out” transitions applied to a screen have finished.
Screen.mouseDown	Broadcast when the mouse button was pressed over an object (shape or movie clip) directly owned by the screen.
Screen.mouseDownSomewhere	Broadcast when the mouse button was pressed somewhere on the Stage, but not necessarily on an object owned by this screen.
Screen.mouseMove	Broadcast when the mouse is moved while over a screen.
Screen.mouseOut	Broadcast when the mouse is moved from inside the screen to outside it.
Screen.mouseOver	Broadcast when the mouse is moved from outside this screen to inside it.
Screen.mouseUp	Broadcast when the mouse button was released over an object (shape or movie clip) directly owned by the screen.
Screen.mouseUpSomewhere	Broadcast when the mouse button was released somewhere on the Stage, but not necessarily over an object owned by this screen.

Form.currentFocusedForm

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mx.screens.Form.currentFocusedForm
```

Description

Property (read-only); returns the Form object that contains the global current focus. The actual focus may be on the form itself, or on a movie clip, text object, or component inside that form. May be `null` if there is no current focus.

Example

The following code, attached to a button (not shown), displays the name of the form with the current focus.

```
trace("The form with the current focus is: " +  
    mx.screens.Form.currentFocusedForm);
```

Form.getChildForm()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myForm.getChildForm(childIndex)
```

Parameters

childIndex A number that indicates the zero-based index of the child form to return.

Returns

A Form object.

Description

Method; returns the child form of *myForm* whose index is *childIndex*.

Example

The following example is displayed in the Output panel the names of all the child Form objects belonging to the root Form object named `application`.

```
for (var i:Number = 0; i < _root.application.numChildForms; i++) {  
    var childForm:mx.screens.Form = _root.application.getChildForm(i);  
    trace(childForm._name);  
}
```

See also

[Form.numChildForms](#)

Form.indexInParentForm

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myForm.indexInParentForm

Description

Property (read-only); contains the zero-based index of *myForm* in its parent's list of child forms. If the parent object of *myForm* is a screen but not a form (for example, if it is a slide), `indexInParentForm` is always 0.

Example

```
var myIndex:Number = myForm.indexInParent;  
if (myForm == myForm._parent.getChildForm(myIndex)) {  
    trace("I'm where I should be");  
}
```

See also

[Form.getChildForm\(\)](#)

Form.numChildForms

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myForm.numChildForms

Description

Property (read-only); the number of child forms contained by *myForm* that are derived directly from the class `mx.screens.Form`. This property does not include any slides that are contained by *myForm*; it contains only forms.

NOTE

When using a custom ActionScript 2.0 class that extends `mx.screens.Form`, the form isn't counted in the `numChildForms` property.

Example

The following code iterates over all the child forms contained in *myForm* and displays their names in the Output panel.

```
var howManyKids:Number = myForm.numChildForms;
for(i=0; i<howManyKids; i++) {
    var childForm = myForm.getChildForm(i);
    trace(childForm._name);
}
```

See also

[Form.getChildForm\(\)](#)

Form.parentIsForm

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`myForm.parentIsForm`

Description

Property (read-only): returns a Boolean value indicating whether the specified form's parent object is also a form (`true`) or not (`false`). If this property is `false`, *myForm* is at the root of its form hierarchy.

Example

```
if (myForm.parentIsForm) {
    trace("I have "+myForm._parent.numChildScreens+" sibling screens");
} else {
    trace("I am the root form and have no siblings");
}
```

Form.parentForm

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myForm.parentForm

Description

Property (read-only): a reference to the form's parent form.

Example

The following example code resides on a screen named *myForm* that is a child of the default *form1* screen created when you select Flash Form Application from the New Document dialog box.

```
onClipEvent(keyDown){
    var parentForm:mx.screens.Form = this.parentForm;
    trace(parentForm);
}
// output: _level0.application.form1
```

Form.rootForm

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myForm.rootForm

Description

Property (read-only); returns the form at the top of the form hierarchy that contains *myForm*. If *myForm* is contained by an object that is not a form (that is, a slide), this property returns *myForm*.

Example

In the following example, a reference to the root form of `myForm` is placed in a variable named `root`. If the value assigned to `root` refers to `myForm`, then `myForm` is at the top of its form tree.

```
var root:mx.screens.Form = myForm.rootForm;
if(root == myForm) {
    trace("myForm is the top form in its tree");
}
```

Form.visible

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`myForm.visible`

Description

Property; determines whether `myForm` is visible when its parent form, slide, movie clip, or SWF file is visible. You can also set this property using the Property inspector in the Flash authoring environment.

When this property is set to `true`, `myForm` receives a `reveal` event; when set to `false`, `myForm` receives a `hide` event. You can attach transitions to forms that execute when a form receives one of these events. For more information on adding transitions to screens, see “Creating controls and transitions for screens with behaviors (Flash Professional only)” in *Using Flash*.

Example

The following code, on a timeline frame, sets the `visible` property of the form that contains the button to `false`.

```
btnOk.addEventListener("click", btnOkClick);
function btnOkClick(eventObj:Object):Void {
    eventObj.target._parent.visible = false;
}
```


Iterator interface (Flash Professional only)

ActionScript Class Name mx.utils.Iterator

The Iterator interface lets you step through the objects that a collection contains.

Method summary for the Iterator interface

The following table lists the methods of the Iterator interface.

Method	Description
<code>Iterator.hasNext()</code>	Indicates whether the iterator has more items.
<code>Iterator.next()</code>	Returns the next item in the iteration.

Iterator.hasNext()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
iterator.hasNext()
```

Returns

A Boolean value that indicates whether there are (`true`) or are not (`false`) more instances in the iterator.

Description

Method; indicates whether there are more instances in the iterator. You typically use this method in a `while` statement when looping through an iterator.

Example

The following example uses the `hasNext()` method to control looping through the iterator of items in a collection:

```
on (click) {
    var myColl:mx.utils.Collection;
    myColl = _parent.thisShelf.MyCompactDisks;
    var itr:mx.utils.Iterator = myColl.getIterator();
    while (itr.hasNext()) {
        var cd:CompactDisk = CompactDisk(itr.next());
        var title:String = cd.Title;
        var artist:String = cd.Artist;
        trace("Title: "+title+" Artist: "+artist);
    }
}
```

Iterator.next()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
iterator.next()
```

Returns

An object that is the next item in the iterator.

Description

Method; returns an instance of the next item in the iterator. You must cast this instance to the correct type.

Example

The following example uses the `next()` method to access the next item in a collection:

```
on (click) {
    var myColl:mx.utils.Collection;
    myColl = _parent.thisShelf.MyCompactDisks;
    var itr:mx.utils.Iterator = myColl.getIterator();
    while (itr.hasNext()) {
        var cd:CompactDisk = CompactDisk(itr.next());
        var title:String = cd.Title;
        var artist:String = cd.Artist;
        trace("Title: "+title+" Artist: "+artist);
    }
}
```

A Label component is a single line of text. You can specify that a label be formatted with HTML. You can also control the alignment and size of a label. Label components don't have borders, cannot be focused, and don't broadcast any events.

A live preview of each Label instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. The label doesn't have a border, so the only way to see its live preview is to set its text parameter. The autoSize parameter is not supported in live preview.

When you add the Label component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.LabelAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 19, "Creating Accessible Content," in *Using Flash*.

Using the Label component

Use a Label component to create a text label for another component in a form, such as a "Name:" label to the left of a TextInput field that accepts a user's name. If you're building an application using components based on version 2 of the Macromedia Component Architecture, it's a good idea to use a Label component instead of a plain text field because you can use styles to maintain a consistent look and feel.

If you want to rotate a Label component, you must embed the fonts. See ["Using styles with the Label component"](#) on page 753.

Label parameters

You can set the following authoring parameters for each Label component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

autoSize indicates how the label is sized and aligned to fit the text. The default value is `none`. The parameter can have any of the following four values:

- `none`, which specifies that the label is not resized or aligned to fit the text.
- `left`, which specifies that the right and bottom sides of the label are resized to fit the text. The left and top sides are not resized.
- `center`, which specifies that the left and right sides of the label resize to fit the text. The horizontal center of the label stays anchored at its original horizontal center position.
- `right`, which specifies that the left and bottom sides of the label are resized to fit the text. The top and right side are not resized.

NOTE

The `autoSize` property of the Label component is different from the `autoSize` property of the built-in ActionScript TextField object.

html indicates whether the label is formatted with HTML (`true`) or not (`false`). If this parameter is set to `true`, a label cannot be formatted with styles, but you can format the text as HTML using the `font` tag. The default value is `false`.

text indicates the text of the label; the default value is `Label`.

You can set the following additional parameters for each Label component instance in the Component inspector (Window > Component Inspector):

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to set additional options for Label instances using its methods, properties, and events. For more information, see [“Label class” on page 755](#).

Creating an application with the Label component

The following procedure explains how to add a Label component to an application while authoring. In this example, the label is beside a combo box with dates in a shopping cart application.

To create an application with the Label component:

1. Drag a Label component from the Components panel to the Stage.
2. In the Component inspector, enter **Expiration Date** for the label parameter.

To create a Label component instance using ActionScript:

1. Drag the Label component from the Components panel to the current document's library.
This adds the component to the library, but doesn't make it visible in the application.
2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.Label, "my_label", 1);  
my_label.text = "Hello World";
```

This script uses the method `UIObject.createClassObject()` to create the Label instance.

3. Select Control > Test Movie.

Customizing the Label component

You can transform a Label component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. You can also set the `autoSize` authoring parameter; setting this parameter doesn't change the bounding box in the live preview, but the label is resized. For more information, see [“Label parameters” on page 752](#). At runtime, use the `setSize()` method (see `UIObject.setSize()` or `Label.autoSize`).

Using styles with the Label component

You can set style properties to change the appearance of a label instance. All text in a Label component instance must share the same style. For example, you can't set the `color` style to "blue" for one word in a label and to "red" for the second word in the same label.

If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties. For more information about styles, see [“Using styles to customize component color and text” in *Using Components*](#).

A Label component supports the following styles:

Style	Theme	Description
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is <code>10</code> .
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

Using skins with the Label component

The Label component does not have any visual elements to skin.

Label class

Inheritance MovieClip > [UIObject class](#) > Label

ActionScript Class Name mx.controls.Label

The properties of the Label class allow you at runtime to specify text for the label, indicate whether the text can be formatted with HTML, and indicate whether the label auto-sizes to fit the text.

Setting a property of the Label class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

When you access the values of label properties, make sure the component has finished loading before you try to access the desired property. Consider the following example:

```
var listenerObject:Object = new Object();
listenerObject.load = function(){
    trace(label.width);
};
label.addEventListener("load", listenerObject);
```

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Label.version);
```

NOTE

The code `trace(myLabelInstance.version);` returns `undefined`.

Method summary for the Label class

There are no methods exclusive to the Label class.

Methods inherited from the UIObject class

The following table lists the methods the Label class inherits from the UIObject class. When calling these methods from the Label object, use the form `labelInstance.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.

Method	Description
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the Label class

The following table lists properties of the Label class.

Property	Description
<code>Label.autoSize</code>	A string that indicates how a label sizes and aligns to fit the value of its <code>text</code> property. There are four possible values: "none", "left", "center", and "right". The default value is "none".
<code>Label.html</code>	A Boolean value that indicates whether a label can be formatted with HTML (<code>true</code>) or not (<code>false</code>).
<code>Label.text</code>	The text on the label.

Properties inherited from the UIObject class

The following table lists the properties the Label class inherits from the UIObject class. When you access these properties, use the form `labelInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the Label class

There are no events exclusive to the Label class.

Events inherited from the UIObject class

The following table lists the events the Label class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Label.autoSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

LabelInstance.autoSize

Description

Property; a string that indicates how a label is sized and aligned to fit the value of its `text` property. There are four possible values: "none", "left", "center", and "right". The default value is "none".

- `none` The label is not resized or aligned to fit the text.
- `left` The right and bottom sides of the label are resized to fit the text. The left and top sides are not resized.
- `center` The left and right sides of the label resize to fit the text. The horizontal center of the label stays anchored at its original horizontal center position.
- `right` The left and bottom sides of the label are resized to fit the text. The top and right sides are not resized.

NOTE

The `autoSize` property of the `Label` component is different from the `autoSize` property of the built-in `ActionScript TextField` object.

Example

In following example, the label instance `my_label` resizes the left and bottom sides of the label to fit all the text:

```
my_label.text = "A really long label with Label.autoSize set";  
my_label.autoSize = "right";
```

Label.html

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

LabelInstance.html

Description

Property; a Boolean value that indicates whether the label can be formatted with HTML (*true*) or not (*false*). The default value is *false*. Label components with the *html* property set to *true* cannot be formatted with styles.

To retrieve plain text from HTML-formatted text, set the *HTML* property to *false* and then access the *text* property. This removes the HTML formatting, so you may want to copy the label text to an offscreen Label or TextArea component before you retrieve the plain text.

Example

The following example sets the *html* property to *true* so the label can be formatted with HTML. The *text* property is then set to a string that includes HTML formatting.

```
my_label.html = true;  
my_label.text = "The <b>Royal</b> Nonesuch";  
my_label.autoSize = "right";
```

The word “Royal” is displayed in bold.

Label.text

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

LabelInstance.text

Description

Property; the text of a label. The default value is "Label".

Example

The following code sets the *text* property of the Label instance *my_label* and sends the value to the Output panel:

```
my_label.text = "The Royal Nonesuch";  
trace(my_label.text);
```


The List component is a scrollable single- or multiple-selection list box. A list can also display graphics, including other components. You add the items displayed in the list by using the Values dialog box that appears when you click in the labels or data parameter fields. You can also use the `List.addItem()` and `List.addItemAt()` methods to add items to the list.

The List component uses a zero-based index, where the item with index 0 is the top item displayed. When adding, removing, or replacing list items using the List class methods and properties, you may need to specify the index of the list item.

The list receives focus when you click it or tab to it, and you can then use the following keys to control it:

Key	Description
Alphanumeric keys	Jump to the next item that has <code>Key.getAscii()</code> as the first character in its label.
Control	Toggle key that allows multiple noncontiguous selections and deselections.
Down Arrow	Selection moves down one item.
Home	Selection moves to the top of the list.
Page Down	Selection moves down one page.
Page Up	Selection moves up one page.
Shift	Allows for contiguous selection.
Up Arrow	Selection moves up one item.

NOTE

The page size used by the Page Up and Page Down keys is one less than the number of items that fit in the display. For example, paging down through a ten-line drop-down list shows items 0-9, 9-18, 18-27, and so on, with one item overlapping per page.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each List instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring.

When you add the List component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.ListAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the List component

You can set up a list so that users can make either single or multiple selections. For example, a user visiting an e-commerce website needs to select which item to buy. There are 30 items, and the user scrolls through a list and selects one by clicking it.

You can also design a list that uses custom movie clips as rows so you can display more information to the user. For example, in an e-mail application, each mailbox could be a List component and each row could have icons to indicate priority and status.

Understanding the design of the List component

When you design an application with the List component, or any component that extends the List class, it is helpful to understand how the list was designed. The following are some fundamental assumptions and requirements that Macromedia used when developing the List class:

- Keep it small, fast, and simple.
 - Don't make something more complicated than absolutely necessary. This was the prime design directive. Most of the requirements listed below are based on this directive.
- Lists have uniform row heights.
 - Every row must be the same height; the height can be set during authoring or at runtime.
- Lists must scale to thousands of records.

- Lists don't measure text.

This restriction has the most potential ramifications. Because a list must scale to thousands of records, any one of which could contain an unusually long string, it shouldn't grow to fit the largest string of text within it, or add a horizontal scroll bar in "auto" mode. Also, measuring thousands of strings would be too intensive. The compromise is the `maxHPosition` property, which, when `vScrollPolicy` is set to "on", gives the list extra buffer space for scrolling.

If you know you're likely to deal with long strings, turn `hScrollPolicy` to "on", and add a 200-pixel `maxHPosition` value to your `List` or `Tree` component. A user is more or less guaranteed to be able to scroll to see everything. The `DataGrid` component, however, does support "auto" as an `hScrollPolicy` value, because it measures columns (which are the same width per item), not text.

The fact that lists don't measure text also explains why lists have uniform row heights. Sizing individual rows to fit text would require intensive measuring. For example, if you wanted to accurately show the scroll bars on a list with nonuniform row height, you'd need to premeasure every row.

- Lists perform worse as a function of their visible rows.

Although lists can display 5000 records, they can't render 5000 records at once. The more visible rows (specified by the `rowCount` property) you have on the Stage, the more work the list must do to render. Limiting the number of visible rows, if at all possible, is the best solution.

- Lists aren't tables.

For example, `DataGrid` components, which extend the `List` class, are intended to provide an interface for many records. They're not designed to display complete information; they're designed to display enough information so that users can drill down to see more. The message view in Microsoft Outlook is a prime example. You don't read the entire e-mail in the grid; the mail would be difficult to read and the client would perform terribly. Outlook displays enough information so that a user can drill into the post to see the details.

List parameters

You can set the following authoring parameters for each List component instance in the Property inspector or in the Component inspector:

data is an array of values that populate the data of the list. The default value is [] (an empty array). There is no equivalent runtime property.

labels is an array of text values that populate the label values of the list. The default value is [] (an empty array). There is no equivalent runtime property.

multipleSelection is a Boolean value that indicates whether you can select multiple values (*true*) or not (*false*). The default value is *false*.

rowHeight indicates the height, in pixels, of each row. The default value is 20. Setting a font does not change the height of a row.

You can write ActionScript to set additional options for List instances using its methods, properties, and events. For more information, see [“List class” on page 770](#).

Creating an application with the List component

The following procedure explains how to add a List component to an application while authoring. In this example, the list is a sample with three items.

To add a simple List component to an application:

1. Drag a List component from the Components panel to the Stage.
2. Select the Free Transform tool and resize the component to fit your application.
3. In the Property inspector, do the following:
 - Enter the instance name **my_list**.
 - Enter **Item1**, **Item2**, and **Item3** for the labels parameter.
 - Enter **item1.html**, **item2.html**, **item3.html** for the data parameter.
4. Select Control > Test Movie to see the list with its items.
5. Return to the authoring environment, insert a new layer, and name it **actions**.
6. Add the following ActionScript to Frame 1 of the actions layer.

```
my_list.change = function(evt:Object) {
    getURL(evt.target.selectedItem.data, "_blank");
};
my_list.addEventListener("change", my_list);
```


To populate a List instance with a data provider:

1. Drag a List component from the Components panel to the Stage.
2. Select the Free Transform tool and resize the component to fit your application.
3. In the Property inspector, enter the instance name **my_list**.
4. Select Frame 1 of the Timeline and, in the Actions panel, enter the following:

```
my_list.dataProvider = myDP;
```

If you have defined a data provider named `myDP`, the list fills with data. (For more information about data providers, see [List.dataProvider](#).)

5. Select Control > Test Movie to see the list with its items.

To use a List component to control a movie clip instance

1. Drag a List component from the Components panel to the Stage.
2. Select the Free Transform tool and resize the component to fit your application.
3. In the Property inspector, enter the instance name **my_list**.
4. Create a movie clip on the Stage and give it the instance name **my_mc**.
5. Open the movie clip in symbol-editing mode and add some animation.
6. Insert a new layer and name it **actions**.
7. Add the following ActionScript to Frame 1 of the actions layer.

```
my_list.addItem({label:"play", data:"play"});
my_list.addItem({label:"stop", data:"stop"});
var listHandler:Object = new Object();
listHandler.change = function(evt:Object) {
    switch (evt.target.selectedItem.data) {
        case "play" :
            my_mc.play();
            break;
        case "stop" :
            my_mc.stop();
            break;
        default :
            trace("unhandled event: "+evt.target.selectedItem.data);
            break;
    }
};
my_list.addEventListener("change", listHandler);
```

8. Select Control > Test Movie to use the list to stop and play the `my_mc` movie clip instance.

To create a List component instance using ActionScript:

1. Drag the List component from the Components panel to the library.

This adds the component to the library, but doesn't make it visible in the application.

2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.List, "my_list", 1);  
my_list.addItem({label:"One", data:dt1});  
my_list.addItem({label:"Two", data:dt2});
```

This script uses the method `UIObject.createClassObject()` to create the List instance.

3. Select Control > Test Movie.

Customizing the List component

You can transform a List component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `List.setSize()` method (see `UIObject.setSize()`).

When a list is resized, the rows of the list shrink horizontally, clipping any text within them. Vertically, the list adds or removes rows as needed. Scroll bars position themselves automatically. For more information about scroll bars, see [“ScrollPane component” on page 1093](#).

Using styles with the List component

You can set style properties to change the appearance of a List component.

A List component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>alternatingRowColors</code>	Both	Specifies colors for rows in an alternating pattern. The value can be an array of two or more colors, for example, <code>0xFF00FF</code> , <code>0xCC6699</code> , and <code>0x996699</code> . Unlike single-value color styles, <code>alternatingRowColors</code> does not accept color names; the values must be numeric color codes. By default, this style is not set and <code>backgroundColor</code> is used in its place for all rows.
<code>backgroundColor</code>	Both	The background color of the list. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to "false". The default value is <code>0xDDDDDD</code> (medium gray).
<code>borderStyle</code>	Both	The List component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" on page 1063 . The default border style is "inset".
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".

Style	Theme	Description
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>defaultIcon</code>	Both	The name of the default icon to display on each row. The default value is <code>undefined</code> , which means no icon is displayed.
<code>repeatDelay</code>	Both	The number of milliseconds of delay between when a user first presses a button in the scrollbar and when the action begins to repeat. The default value is 500, half a second.
<code>repeatInterval</code>	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button in the scrollbar. The default value is 35.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>0xE3FFD6</code> (bright green) with the Halo theme and <code>0xA9A9A9</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionColor</code>	Both	The background color of a selected row. The default value is <code>0xCDFFC1</code> (light green) with the Halo theme and <code>0xE0E0E0</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal to selected state or back from selected to normal, in milliseconds. The default value is 200.

Style	Theme	Description
<code>selectionDisabledColor</code>	Both	The background color of a selected row. The default value is a <code>0xDDDDDD</code> (medium gray). Because the default value for this property is the same as the default for <code>backgroundDisabledColor</code> , the selection is not visible when the component is disabled unless one of these style properties is changed.
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. This applies only for the transition from a normal to a selected state. The default equation uses a sine in/out formula. For more information, see “Customizing component animations” in <i>Using Components</i> .
<code>textRollOverColor</code>	Both	The color of text when the pointer rolls over it. The default value is <code>0x2B333C</code> (dark gray). This style is important when you set <code>rollOverColor</code> , because the two settings must complement each other so that text is easily viewable during a rollover.
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two settings must complement each other so that text is easily viewable while selected.
<code>useRollOver</code>	Both	Determines whether rolling over a row activates highlighting. The default value is <code>true</code> .

Setting styles for all List components in a document

The `List` class inherits from the `ScrollSelectList` class. The default class-level style properties are defined on the `ScrollSelectList` class, which the `Menu` component and all `List`-based components extend. You can set new default style values on this class directly, and the new settings are reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the `List` and `List`-based components only, you can create a new `CSSStyleDeclaration` instance and store it in `_global.styles.List`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.List == undefined) {
    _global.styles.List = new CSSStyleDeclaration();
}
_global.styles.List.setStyle("backgroundColor", 0xFF00AA);
```

When creating a new class-level style declaration, you lose all default values provided by the `ScrollSelectList` declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.List;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

To provide styles for the `List` component but not for components that extend `List` (`DataGrid` and `Tree`), you must provide class-level style declarations for these subclasses.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
    _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFFFFFFFF);
if (_global.styles.Tree == undefined) {
    _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFFFFFFFF);
```

For more information about class-level styles, see “Setting styles for a component class” in *Using Components*.

Using skins with the List component

The `List` component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see “[RectBorder class](#)” on page 1063 and see “[Using skins with the UIScrollBar component](#)” on page 1394.

List class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `ScrollSelectList` > `List`

ActionScript Class Name `mx.controls.List`

The `List` component is composed of three parts: items, rows, and a data provider.

An *item* is an ActionScript object used for storing the units of information in the list. A list can be thought of as an array; each indexed space of the array is an item. An item is an object that typically has a `label` property that is displayed and a `data` property that is used for storing data.

A *row* is a component that is used to display an item. Rows are either supplied by default by the list (the `SelectableRow` class is used), or you can supply them, usually as a subclass of the `SelectableRow` class. The `SelectableRow` class implements the `CellRenderer` API, which is the set of properties and methods that allow the list to manipulate each row and send data and state information (for example, size, selected, and so on) to the row for display.

A data provider is a data model of the list of items in a list. Any array in the same frame as a list is automatically given methods that let you manipulate data and broadcast changes to multiple views. You can build an `Array` instance or get one from a server and use it as a data model for multiple lists, combo boxes, data grids, and so on. The `List` component has methods that proxy to its data provider (for example, `addItem()` and `removeItem()`). If no external data provider is provided to the list, these methods create a data provider instance automatically, which is exposed through `List.dataProvider`.

To add a `List` component to the tab order of an application, set its `tabIndex` property (see `UIComponent.tabIndex`). The `List` component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in *Using Components*.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.List.version);
```

NOTE

The code `trace(myListInstance.version);` returns `undefined`.

Method summary for the `List` class

The following table lists methods of the `List` class.

Method	Description
<code>List.addItem()</code>	Adds an item to the end of the list.
<code>List.addItemAt()</code>	Adds an item to the list at the specified index.
<code>List.getItemAt()</code>	Returns the item at the specified index.
<code>List.removeAll()</code>	Removes all items from the list.
<code>List.removeItemAt()</code>	Removes the item at the specified index.
<code>List.replaceItemAt()</code>	Replaces the item at the specified index with another item.
<code>List.setPropertiesAt()</code>	Applies the specified properties to the specified item.

Method	Description
<code>List.sortItems()</code>	Sorts the items in the list according to the specified compare function.
<code>List.sortItemsBy()</code>	Sorts the items in the list according to a specified property.

Methods inherited from the UIObject class

The following table lists the methods the List class inherits from the UIObject class. When calling these methods, use the form *listInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the List class inherits from the UIComponent class. When calling these methods, use the form *listInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the List class

The following table lists properties of the List class.

Property	Description
List.cellRenderer	Assigns the class or symbol to use to display each row of the list.
List.dataProvider	The source of the list items.
List.hPosition	The horizontal position of the list.
List.hScrollPolicy	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
List.iconField	A field in each item to be used to specify icons.
List.iconFunction	A function that determines which icon to use.
List.labelField	Specifies a field of each item to be used as label text.
List.labelFunction	A function that determines which fields of each item to use for the label text.
List.length	The number of items in the list. This property is read-only.
List.maxHPosition	The number of pixels the list can scroll to the right, when List.hScrollPolicy is set to "on".
List.multipleSelection	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
List.rowCount	The number of rows that are at least partially visible in the list.
List.rowHeight	The pixel height of every row in the list.
List.selectable	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).
List.selectedIndex	The index of a selection in a single-selection list.
List.selectedIndices	An array of the selected items in a multiple-selection list.
List.selectedItem	The selected item in a single-selection list. This property is read-only.
List.selectedItems	The selected item objects in a multiple-selection list. This property is read-only.
List.vPosition	The topmost visible item of the list.
List.vScrollPolicy	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Properties inherited from the UIObject class

The following table lists the properties the List class inherits from the UIObject class. When accessing these properties, use the form *listInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the List class inherits from the UIComponent class. When accessing these properties, use the form *listInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the List class

The following table lists events that of the List class.

Event	Description
<code>List.change</code>	Broadcast whenever user interaction causes the selection to change.
<code>List.itemRollOut</code>	Broadcast when the pointer rolls over and then off of list items.
<code>List.itemRollOver</code>	Broadcast when the pointer rolls over list items.
<code>List.scroll</code>	Broadcast when a list is scrolled.

Events inherited from the UIObject class

The following table lists the events the List class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the List class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

List.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.addItem(label[, data])
```

```
listInstance.addItem(itemObject)
```

Parameters

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any data type.

itemObject An item object that usually has `label` and `data` properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list.

In the first usage example, an item object is always created with the specified `label` property, and, if specified, the `data` property.

The second usage example adds the specified item object.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components are updated as well.

Example

Both of the following lines of code add an item to the `my_list` instance. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;
```

```
my_list.addItem("this is an Item");
```

```
my_list.addItem({label:"Gordon", age:"very old", data:123});
```

List.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.addItemAt(index, label[, data])
```

```
listInstance.addItemAt(index, itemObject)
```

Parameters

index A number greater than or equal to 0 that indicates the position of the item.

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any data type.

itemObject An item object that usually has *label* and *data* properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the position specified by the *index* parameter.

In the first usage example, an item object is always created with the specified *label* property, and, if specified, the *data* property.

The second usage example adds the specified item object.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components are updated as well.

Example

The following example adds an item to the first index position, which is the second item in the list. To try this code, drag a List component to the Stage and give it the instance name **my_list**. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;  
  
my_list.addItem("this is an Item");  
my_list.addItem({label:"Gordon", age:"very old", data:123});  
my_list.addItemAt(1, {label:"Red", data:0xFF0000});
```

List.cellRenderer

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.cellRenderer
```

Description

Property; assigns the cell renderer to use for each row of the list. This property must be a class object reference or a symbol linkage identifier. Any class used for this property must implement the [CellRenderer API](#).

Example

The following example uses a linkage identifier to set a new cell renderer:

```
my_list.cellRenderer = "ComboBoxCell";
```

List.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();  
listenerObject.change = function(eventObject:Object) {  
    // Your code here.  
};  
listInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {  
    // Your code here.  
}
```

Description

Event; broadcast to all registered listeners when the selected index of the list changes as a result of user interaction.

The first usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class \(API\)” on page 500](#).

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a List instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the List instance `my_list`, sends “_level0.my_list” to the Output panel:

```
on (change) {  
    trace(this);  
}
```

Example

The following example adds three items to the List component. Changing the selected value of the list causes the value of the newly selected item to be displayed in the Output panel. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;  
  
my_list.addItem({data:'flash', label:'Flash'});  
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});  
my_list.addItem({data:'coldfusion', label:'ColdFusion'});  
  
// Create listener object.  
var listListener:Object = new Object();  
listListener.change = function(evt_obj:Object) {  
    trace("Value changed to: " + evt_obj.target.value);  
}  
// Add listener.  
my_list.addEventListener("change", listListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

List.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.dataProvider

Description

Property; the data model for items viewed in a list. The value of this property can be an array or any object that implements the DataProvider API. The default value is []. For more information, see [“DataProvider API” on page 317](#).

The List component, like other data-aware components, adds methods to the Array object’s prototype so that they conform to the DataProvider API. Therefore, any array that exists at the same time as a list automatically has all the methods (`addItem()`, `getItemAt()`, and so on) it needs to be the data model for the list, and can be used to broadcast model changes to multiple components.

If the array contains objects, the `List.labelField` or `List.labelFunction` properties are accessed to determine what parts of the item to display. The default value is "label", so if a label field exists, it is chosen for display; if it doesn’t exist, a comma-separated list of all fields is displayed.

NOTE

If the array contains strings at each index, and not objects, the list is not able to sort the items and maintain the selection state. Any sorting causes the selection to be lost.

Any instance that implements the DataProvider API can be a data provider for a List component. This includes Flash Remoting recordsets, Firefly data sets, and so on.

Example

The following example uses an array of strings to populate the list:

```
my_list.dataProvider = ["Ground Shipping", "2nd Day Air", "Next Day Air"];
```


This example creates a data provider array and assigns it to the `dataProvider` property, as in the following:

```
var myDP_array:Array = new Array();
my_list.dataProvider = myDP_array;

var accounts_array:Array = new Array();
accounts_array.push({name:"checkings", accountID:12345});
accounts_array.push({name:"savings", accountID:67890});

for (var i:Number = 0; i < accounts_array.length; i++) {
    // These changes to the data provider will be broadcast to the list.
    myDP_array.addItem({label:accounts_array[i].name,
        data:accounts_array[i].accountID});
}
```

List.getItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
ListInstance.getItemAt(index)
```

Parameters

index A number greater than or equal to 0, and less than `List.length`. It specifies the index of the item to retrieve.

Returns

The indexed item object; undefined if the index is out of range.

Description

Method; retrieves the item at the specified index. This method gets the data item either from an array, DataProvider, or from a data object created with `CellRenderer.setValue()`.

Example

The following code displays the label of the item at index position 2. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});

trace(my_list.getItemAt(2).label);
```

List.hPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.hPosition

Description

Property; scrolls the list horizontally to the number of pixels specified. You can't set `hPosition` unless the value of `hScrollPolicy` is "on" and the list has a `maxHPosition` that is greater than 0.

Example

The following code displays the current value of `hPosition` whenever the list instance is scrolled horizontally. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 50;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});
```

```
var listListener:Object = new Object();
listListener.scroll = function (evt_obj:Object) {
    trace("my_list.hPosition = " + my_list.hPosition);
}
my_list.addEventListener("scroll", listListener);
```

List.hScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.hScrollPolicy

Description

Property; a string that determines whether the horizontal scroll bar is displayed; the value can be "on" or "off". The default value is "off". The horizontal scroll bar does not measure text; you must set a maximum horizontal scroll position (see [List.maxHPosition](#)).

NOTE

`List.hScrollPolicy` does not support the value "auto".

Example

The following code enables the list to scroll horizontally up to 200 pixels. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 200;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});
```

See also

[List.hPosition](#), [List.maxHPosition](#)

List.iconField

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.iconField

Description

Property; specifies the name of a field to be used as an icon identifier. If the field has a value of undefined, the default icon specified by the defaultIcon style is used. If the defaultIcon style is undefined, no icon is used.

Example

The following example sets the iconField property to the icon property of each item. To try this code, drag a List component to the Stage, give it the instance name **my_list**, and create three symbols with the instance names **flash**, **dreamweaver**, and **cf** respectively. Add the following code to Frame 1 in the timeline:

```
/**
 * Requires:
 * - List component on Stage (instance name: my_list)
 * - MovieClip/Graphic symbol in the Library with Linkage ID of "flash"
 * - MovieClip/Graphic symbol in the Library with Linkage ID of "dreamweaver"
 * - MovieClip/Graphic symbol in the Library with Linkage ID of "cf"
 */

var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash", icon:"flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver",
    icon:"dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion", icon:"cf"});

my_list.iconField = "icon";
```

See also

[List.iconFunction](#)

List.iconFunction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.iconFunction

Description

Property; specifies a function that determines which icon each row uses to display its item. This function receives a parameter, *item*, which is the item being rendered, and must return a string representing the icon's symbol identifier.

Example

The following example adds icons that indicate whether a file is an image or a text document. If the `data.fileExtension` field contains a value of "jpg" or "gif", the icon used is "pictureIcon", and so on.

```
my_list.iconFunction = function(item:Object):String {
    var type:String = item.data.fileExtension;
    if (type == "jpg" || type == "gif") {
        return "pictureIcon";
    } else if (type == "doc" || type == "txt") {
        return "docIcon";
    }
}
```

The following example sets the `iconField` property to the `icon` property of each item. To try this code, drag a `List` component to the Stage, give it the instance name `my_list`, and create three symbols with the instance names `flash`, `dreamweaver`, and `cf` respectively. Add the following code to Frame 1 in the timeline:

```
/**
 * Requires:
 * - List component on Stage (instance name: my_list)
 * - MovieClip/Graphic symbol in the Library with Linkage ID of "flashIcon"
 */

var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash"});
```

```
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.iconFunction = function(item:Object):String {
    if (item.data == "flash") {
        // Put icon next to list item with the data "flash".
        return "flashIcon";
    }
};
my_list.iconField = "icon";
```

List.itemRollOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.itemRollOut = function(eventObject:Object) {
    // Your code here.
};
listInstance.addEventListener("itemRollOut", listenerObject);
```

Usage 2:

```
on (itemRollOut) {
    // Your code here.
}
```

Event object

In addition to the standard properties of the event object, the `itemRollOut` event has an `index` property, which specifies the number of the item that was rolled out.

Description

Event; broadcast to all registered listeners when the pointer rolls over and then off of list items.

The first usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, `itemRollOut`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class \(API\)” on page 500](#).

The second usage example uses an `on()` handler and must be attached directly to a List instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the List instance `my_list`, sends “_level0.my_list” to the Output panel:

```
on (itemRollOut) {
    trace(this);
}
```

Example

The following example sends a message to the Output panel that indicates which item index number has been rolled over:

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

// Create listener object.
var listListener:Object = new Object();
listListener.itemRollOut = function(evt_obj:Object) {
    trace("Item #" + evt_obj.index + " has been rolled out.");
};

// Add listener.
my_list.addEventListener("itemRollOut", listListener);
```

See also

[List.itemRollOver](#)

List.itemRollOver

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.itemRollOver = function(eventObject:Object) {
    // Your code here.
};
listInstance.addEventListener("itemRollOver", listenerObject);
```

Usage 2:

```
on (itemRollOver) {
    // Your code here.
}
```

Event object

In addition to the standard properties of the event object, the `itemRollOver` event has an `index` property that specifies the number of the item that was rolled over.

Description

Event; broadcast to all registered listeners when the list items are rolled over.

The first usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, `itemRollOver`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class \(API\)” on page 500](#).

The second usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `my_list`, sends “`_level0.my_list`” to the Output panel:

```
on (itemRollOver) {
    trace(this);
}
```

Example

The following example sends a message to the Output panel that indicates which item index number has been rolled over:

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

//Create listener object.
var listListener:Object = new Object();
listListener.itemRollOver = function (evt_obj:Object) {
    trace("Item #" + evt_obj.index + " has been rolled over.");
};

//Add listener.
my_list.addEventListener("itemRollOver", listListener);
```

See also

[List.itemRollOut](#)

List.labelField

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.labelField
```

Description

Property; specifies a field in each item to be used as display text. This property takes the value of the field and uses it as the label. The default value is "label".

Example

The following example sets the `labelField` property to be the "name" field of each item. "Nina" would display as the label for the item added in the second line of code:

```
var my_list:mx.controls.List;

my_list.labelField = "name";
my_list.addItem({name: "Nina", age: 25});
```

See also

[List.labelFunction](#)

List.labelFunction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.labelFunction

Description

Property; specifies a function that determines which field (or field combination) of each item to display. This function receives one parameter, *item*, which is the item being rendered, and must return a string representing the text to display.

Example

The following example makes the label display some formatted details of the items:

```
var my_list:mx.controls.List;

my_list.setSize(300, 100);

// Define how list data will be displayed.
my_list.labelFunction = function(item_obj:Object):String {
    var label_str:String = item_obj.label + " - Code is: " + item_obj.data;
    return label_str;
}

// Add data to list.
my_list.addItem({data:"f", label:"Flash"});
my_list.addItem({data:"d", label:"Dreamweaver"});
my_list.addItem({data:"c", label:"ColdFusion"});
```

See also

[List.labelField](#)

List.length

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.length

Description

Property (read-only); the number of items in the list.

Example

The following example displays the number of items currently in the list's data provider:

```
var my_list:mx.controls.List;

// Add data to list.
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var listLength_num:Number = my_list.length;
trace("Length of List: " + listLength_num);
```

List.maxHPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.maxHPosition

Description

Property; specifies the number of pixels the list can scroll when `List.hScrollPolicy` is set to "on". The list doesn't precisely measure the width of text that it contains. You must set `maxHPosition` to indicate the amount of scrolling that the list requires. The list does not scroll horizontally if this property is not set.

Example

The following example creates a list with 200 pixels of horizontal scrolling:

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 200;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
```

See also

[List.hScrollPolicy](#)

List.multipleSelection

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`listInstance.multipleSelection`

Description

Property; indicates whether multiple selections are allowed (`true`) or only single selections are allowed (`false`). The default value is `false`.

Example

The following example tests to determine whether multiple items can be selected, and if so, displays instructions in a label component. To try this code, drag a List component to the Stage and give it the instance name **my_list**. Next, drag a Label component on to the Stage and give it the instance name **my_label**. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.multipleSelection = true;

if (my_list.multipleSelection) {
    my_label.text = "Hold down Control or Shift to select multiple items";
    my_label.autoSize = "left";
}
```

List.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.removeAll()
```

Returns

Nothing.

Description

Method; removes all items in the list.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components are updated as well.

Example

The following code clears all items in a List component when a button is clicked. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Next, drag a Button component to the Stage and give it the instance name `remove_button`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;
var remove_button:mx.controls.Button;

remove_button.label = "Remove";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    my_list.removeAll();
    evt_obj.target.enabled = false;
}
remove_button.addEventListener("click", buttonListener);
```

List.removeItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.removeItemAt(index)
```

Parameters

index A number that indicates the position of the item. The value must be greater than 0 and less than `List.length`.

Returns

An object; the removed item (undefined if no item exists).

Description

Method; removes the item at the specified index position. The list indices after the specified index collapse by one.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components are updated as well.

Example

The following code clears the selected item in a List component when a button is clicked. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Next, drag a Button component to the Stage and give it the instance name `remove_button`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;
var remove_button:mx.controls.Button;

remove_button.label = "Remove";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    if (my_list.selectedIndex != undefined) {
        my_list.removeItemAt(my_list.selectedIndex);
    }
}
remove_button.addEventListener("click", buttonListener);
```

List.replaceItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.replaceItemAt(index, label[, data])
```

```
listInstance.replaceItemAt(index, itemObject)
```

Parameters

index A number greater than 0 and less than `List.length` that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any type.

itemObject An object to use as the item, usually containing `label` and `data` properties.

Returns

Nothing.

Description

Method; replaces the content of the item at the specified index.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components are updated as well.

Example

The following example replaces the item at the currently selected position. To try this code, drag a List component to the Stage and give it the instance name `my_list`. Next, drag a Button component to the Stage and give it the instance name `replace_button`. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;
var replace_button:mx.controls.Button;

replace_button.label = "Replace";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    if (my_list.selectedIndex != undefined) {
        my_list.replaceItemAt(my_list.selectedIndex, {data:"flex",
            label:"Flex"});
    }
}
replace_button.addEventListener("click", buttonListener);
```


List.rowCount

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.rowCount

Description

Property; the number of rows that are at least partially visible in the list. This is useful if you've scaled a list by pixel and need to count its rows. Conversely, setting the number of rows guarantees that an exact number of rows is displayed, without a partial row at the bottom.

The code `my_list.rowCount = num` is equivalent to the code `my_list.setSize(my_list.width, h)` (where `h` is the height required to display `num` items).

The default value is based on the height of the list as set during authoring, or as set by the `List.setSize()` method (see [UIObject.setSize\(\)](#)).

Example

The following example discovers the number of visible items in a list:

```
var rowCount = my_list.rowCount;
```

The following example makes the list display four items:

```
my_list.rowCount = 4;
```

The following example removes a partial row at the bottom of a list, if there is one:

```
my_list.rowCount = my_list.rowCount;
```

The following example sets a list to the smallest number of rows it can fully display:

```
my_list.rowCount = 1;  
trace("my_list has " + my_list.rowCount + " rows");
```

The following example resizes the list using the `setSize()` method and then sets a row count of 8 items:

```
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.setSize(200, 30);
my_list.rowCount = 8;
trace("my_list has " + my_list.rowCount + " rows.");
```

List.rowHeight

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ListInstance.rowHeight

Description

Property; the height, in pixels, of every row in the list. The font settings do not make the rows grow to fit, so setting the `rowHeight` property is the best way to make sure items are fully displayed. The default value is 20.

Example

The following example sets each row to 30 pixels:

```
my_list.rowHeight = 30;
```

The following example sets the row height for each row to 30 pixels and then resizes the list to match the total number of items it contains:

```
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.rowHeight = 30;
my_list.rowCount = my_list.length;
```

List.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
    // Your code here.
};
listInstance.addEventListener("scroll", listenerObject);
```

Usage 2:

```
on (scroll) {
    // Your code here.
}
```

Event object

Along with the standard event object properties, the `scroll` event has one additional property, `direction`. It is a string with two possible values, "horizontal" or "vertical". For a `ComboBox` scroll event, the value is always "vertical".

Description

Event; broadcast to all registered listeners when a list is scrolled.

The first usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class \(API\)” on page 500](#).

The second usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `my_list`, sends “_level0.my_list” to the Output panel:

```
on (scroll) {  
    trace(this);  
}
```

Example

The following example sends the direction and position of the list every time the list items are scrolled:

```
var my_list:mx.controls.List;  
  
my_list.rowCount = 2;  
for (var i:Number = 0; i < 10; i++) {  
    my_list.addItem({data:i, label:"Item #" + i});  
}  
  
var listListener:Object = new Object();  
listListener.scroll = function(evt_obj:Object) {  
    trace("list scrolled (direction:" + evt_obj.direction + ", position:" +  
        evt_obj.position + ")");  
};  
my_list.addEventListener("scroll", listListener);
```

List.selectable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectable

Description

Property; a Boolean value that indicates whether the list is selectable (`true`) or not (`false`). The default value is `true`.

Example

The following example prevents users from selecting items in the list by setting the selectable property to `false`:

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.selectable = false;
```

List.selectedIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectedIndex

Description

Property; the selected index of a single-selection list. The value is `undefined` if nothing is selected; the value is equal to the last item selected if there are multiple selections. If you assign a value to `selectedIndex`, any current selection is cleared and the indicated item is selected.

Using the `selectedIndex` property to change selection doesn't dispatch a change event. To dispatch the change event, use the following code:

```
my_list.dispatchEvent({type:"change", target:my_list});
```

Example

The following example selects the first item in a list by default and displays the index of the currently selected whenever the user selects a new item:

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

// Select first item by default.
my_list.selectedIndex = 0;

var listListener:Object = new Object();
listListener.change = function(evt_obj:Object) {
    trace("selectedIndex = " + evt_obj.target.selectedIndex);
}
my_list.addEventListener("change", listListener);
```

See also

[List.selectedIndex](#), [List.selectedItem](#), [List.selectedItems](#)

List.selectedIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectedIndex

Description

Property; an array of indices of the selected items. Assigning this property replaces the current selection. Setting `selectedIndex` to a zero-length array (or undefined) clears the current selection. The value is undefined if nothing is selected.

The `selectedIndex` property reflects the order in which the items were selected. If you click the second item, then the third item, and then the first item, `selectedIndex` returns `[1,2,0]`.

Example

The following example retrieves the selected indices:

```
var selIndices:Array = my_list.selectedIndices;
```

The following example selects four items:

```
var my_array = new Array (1, 4, 5, 7);  
my_list.selectedIndices = my_array;
```

The following example selects two list items by default and displays their label property in the Output panel:

```
my_list.multipleSelection = true;  
  
my_list.addItem({data:"flash", label:"Flash"});  
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});  
my_list.addItem({data:"coldfusion", label:"ColdFusion"});  
  
my_list.selectedIndices = [0, 2];  
  
var numSelected:Number = my_list.selectedIndices.length;  
for (var i:Number = 0; i < numSelected; i++) {  
    trace("selectedIndices[" + i + "] = "+  
        my_list.getItemAt(my_list.selectedIndices[i]).label);  
}
```

See also

[List.selectedIndex](#), [List.selectedItem](#), [List.selectedItems](#)

List.selectedItem

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
ListInstance.selectedItem
```

Description

Property (read-only); an item object in a single-selection list. (In a multiple-selection list with multiple items selected, `selectedItem` returns the item that was most recently selected.) If there is no selection, the value is undefined.

Example

The following example displays the selected label:

```
trace(my_list.selectedItem.label);
```

The following example displays the contents of a selected whenever the user selects a new item from the list:

```
my_list.multipleSelection = true;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

//Create listener object.
var listListener:Object = new Object();
listListener.change = function(evt_obj:Object) {
    // Display each property of the object.
    var tempStr:String = "[object]";
    for (var i:String in evt_obj.target.selectedItem) {
        tempStr += " " + i + ":'" + evt_obj.target.selectedItem[i]+"'";
    }
    tempStr += "]";
    trace(tempStr);
};
// Add listener.
my_list.addEventListener("change", listListener);
```

See also

[List.selectedIndex](#), [List.selectedIndices](#), [List.selectedItems](#)

List.selectedItems

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.selectedItems
```

Description

Property (read-only); an array of the selected item objects. In a multiple-selection list, `selectedItems` lets you access the set of items selected as item objects.

Example

The following example retrieves an array of selected item objects:

```
var myObjArray:Array = my_list.selectedItems;
```

The following example displays two List instances on the Stage. When a user selects an item from the first list, the selected item is copied to the second list. To try this code, you must add a copy of the List component to the library of the current document. Add the following code to Frame 1 in the timeline:

```
this.createClassObject(mx.controls.List, "my_list", 10,
    {multipleSelection:true});
my_list.setSize(200, 100);

this.createClassObject(mx.controls.List, "selectedItems_list", 20,
    {selectable:false});
selectedItems_list.setSize(200, 100);
selectedItems_list.move(0, 110);

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var listListener:Object = new Object();
listListener.change = function(evt_obj:Object) {
    trace("You have selected " + my_list.selectedItems.length + " items.");
    selectedItems_list.dataProvider = my_list.selectedItems;
}
my_list.addEventListener("change", listListener);
```

See also

[List.selectedIndex](#), [List.selectedItem](#), [List.selectedIndices](#)

List.setPropertiesAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.setPropertiesAt(index, styleObj)
```

Parameters

index A number greater than 0 or less than `List.length` indicating the index of the item to change.

styleObj An object that enumerates the properties and values to set.

Returns

Nothing.

Description

Method; applies the specified properties to the specified item. The supported properties are `icon` and `backgroundColor`.

Example

The following example changes the background color of the third item to red and gives it an icon. To try this code, drag a `List` component to the Stage and give it the instance name `my_list`. Next, add a `MovieClip/Graphic` symbol to the library with a linkage identifier of “file”. Add the following code to Frame 1 in the timeline:

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.setPropertiesAt(2, {backgroundColor:0xFF0000, icon: "file"});
```

List.sortItems()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.sortItems(compareFunc)
```

Parameters

compareFunc A reference to a function. This function is used to compare two items to determine their sort order.

For more information, see `sort` (`Array.sort` method) in *ActionScript 2.0 Language Reference*.

Returns

Nothing.

Description

Method; sorts the items in the list by using the function specified in the *compareFunc* parameter.

Example

The following example sorts the items according to uppercase labels. Note that the `a` and `b` parameters that are passed to the function are items that have `label` and `data` properties.

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.sortItems(uppercaseFunc);
function uppercaseFunc(a:Object, b:Object):Boolean {
    return (a.label.toUpperCase() > b.label.toUpperCase());
}
```

See also

[List.sortItemsBy\(\)](#)

List.sortItemsBy()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.sortItemsBy(fieldName, optionsFlag)
```

```
listInstance.sortItemsBy(fieldName, order)
```

Parameters

fieldName A string that specifies the name of the field to use for sorting. This value is usually "label" or "data".

order A string that specifies whether to sort the items in ascending order ("ASC") or descending order ("DESC").

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or resort it repeatedly.

The following are possible values for *optionsFlag*:

- `Array.DECENDING`, which sorts from highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
my_array.sort (Array.NUMERIC | Array.DECENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the list in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but you can specify any primitive data value.

This is the fastest way to sort data in a component. It also maintains the component's selection state. The `sortItemsBy()` method is fast because it doesn't run any ActionScript while sorting. The `sortItems()` method needs to run an ActionScript compare function, and is therefore slower.

Example

The following code sorts the items in the list in ascending order using the labels of the list items:

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.sortItemsBy("label", "ASC");
```

See also

[List.sortItems\(\)](#)

List.vPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.vPosition

Description

Property; sets the topmost visible item of the list. If you set this property to an index number that doesn't exist, the list scrolls to the nearest index. The default value is 0.

Example

The following example displays the current value of the `vPosition` of the list whenever the contents of the list are scrolled:

```
my_list.setSize(200, 60);
my_list.rowCount = 4;
my_list.vPosition = 2;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"flex", label:"Flex"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"fireworks", label:"Fireworks"});
my_list.addItem({data:"contribute", label:"Contribute"});
my_list.addItem({data:"breeze", label:"Breeze"});

var listListener:Object = new Object();
listListener.scroll = function(evt_obj:Object) {
    trace("my_list.vPosition = " + my_list.vPosition);
}
my_list.addEventListener("scroll", listListener);
```

List.vScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

listInstance.vScrollPolicy

Description

Property; a string that determines whether the list supports vertical scrolling. The value of this property can be "on", "off" or "auto". The value "auto" causes a scroll bar to appear when needed.

Example

The following example disables the vertical scroll bar for a list:

```
var my_list:mx.controls.List;

my_list.setSize(200, 60);
my_list.rowCount = 4;
my_list.vScrollPolicy = "off";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"flex", label:"Flex"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"fireworks", label:"Fireworks"});
my_list.addItem({data:"contribute", label:"Contribute"});
my_list.addItem({data:"breeze", label:"Breeze"});

var listListener:Object = new Object();
listListener.scroll = function(evt_obj:Object) {
    trace("my_list.vPosition = " + my_list.vPosition);
}
my_list.addEventListener("scroll", listListener);
```

You can still create scrolling by using [List.vPosition](#), or by using the mouse or keyboard.

See also

[List.vPosition](#)

The Loader component is a container that can display a SWF or JPEG file (but not *progressive* JPEG files). You can scale the contents of the loader or resize the loader itself to accommodate the size of the contents. By default, the contents are scaled to fit the loader. You can also load content at runtime and monitor loading progress (although after content has been loaded once it is cached, so the progress jumps to 100% quickly).

A Loader component can't receive focus. However, content loaded into the Loader component can accept focus and have its own focus interactions. For more information about controlling focus, see “[FocusManager class](#)” on page 721 or “Creating custom focus navigation” in *Using Components*.

A live preview of each Loader instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

You can use the Accessibility panel to make Loader component content accessible to screen readers. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the Loader component

You can use a loader whenever you need to retrieve content from a remote location and pull it into a Flash application. For example, you could use a loader to add a company logo (JPEG file) to a form. You could also use a loader to leverage Flash work that has already been completed. For example, if you had already built a Flash application and wanted to expand it, you could use the loader to pull the old application into a new application, perhaps as a section of a tab interface. In another example, you could use the loader component in an application that displays photos. Use `Loader.load()`, `Loader.percentLoaded`, and `Loader.complete` to control the timing of the image loads and display progress bars to the user during loading.

If you load certain version 2 Macromedia Component Architecture components into a SWF file or into the Loader component, the components may not work correctly. These components include the following: Alert, ComboBox, DateField, Menu, MenuBar, and Window.

Use the `_lockroot` property when calling `loadMovie()` or loading into the Loader component. If you're using the Loader component, add the following code:

```
myLoaderComponent.content._lockroot = true;
```

If you're using a movie clip with a call to `loadMovie()`, add the following code:

```
myMovieClip._lockroot = true;
```

If you don't set `_lockroot` to `true` in the loader movie clip, the loader only has access to its own library, but not the library in the loaded movie clip.

Flash Player 7 supports the `_lockroot` property. For information about this property, see [_lockroot \(MovieClip._lockroot property\)](#) in *ActionScript 2.0 Language Reference*.

Components such as Loader, ScrollPane and Window have events to determine when content has finished loading. So, if you want to set properties on the content of a Loader, ScrollPane, or Window, add the property statement within a "complete" event handler, as shown in the following example:

```
loadtest = new Object();
loadtest.complete = function(eventObject){
    content_mc._rotation= 45;
}
my_loader.addEventListener("complete", loadtest)
```

For more information, see ["Loader.complete" on page 823](#).

Loader parameters

You can set the following authoring parameters for each Loader component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

autoload indicates whether the content should load automatically (`true`), or wait to load until the `Loader.load()` method is called (`false`). The default value is `true`.

contentPath an absolute or relative URL indicating the file to load into the loader. A relative path must be relative to the SWF file loading the content. The URL must be in the same subdomain as the URL where the Flash content currently resides. For use in Flash Player or in test mode, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive specifications. The default value is `undefined` until the load starts.

The Loader can load content from other domains, *if* you have policy files in those domains. See “Allowing cross-domain data loading” in *Learning ActionScript 2.0 in Flash*.

scaleContent indicates whether the content scales to fit the loader (`true`), or the loader scales to fit the content (`false`). The default value is `true`.

You can set the following additional parameters for each Loader component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject` and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to set additional options for Loader instances using its methods, properties, and events. For more information, see “[Loader class](#)” on page 817.

Creating an application with the Loader component

The following procedure explains how to add a Loader component to an application while authoring. In this example, the loader loads a logo JPEG from an imaginary URL.

To create an application with the Loader component:

1. Drag a Loader component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name **flower**.
3. Select the loader on the Stage and in the Component inspector, and enter `http://www.flash-mx.com/images/image1.jpg` for the `contentPath` parameter.

To create a Loader component instance using ActionScript:

1. Drag the Loader component from the Components panel to the library.
2. Select the first frame in the main Timeline, open the Actions panel, and enter the following code:

```
this.createClassObject(mx.controls.Loader, "my_loader", 1);  
my_loader.contentPath = "http://www.flash-mx.com/images/image1.jpg";
```

This script uses the method “[UIObject.createClassObject\(\)](#)” on page 1362 to create the Loader instance.

3. Select Control > Test Movie.

Customizing the Loader component

You can transform a Loader component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

The sizing behavior of the Loader component is controlled by the `scaleContent` property. When `scaleContent` is `true`, the content is scaled to fit within the bounds of the loader (and is rescaled when [UIObject.setSize\(\)](#) is called). When `scaleContent` is `false`, the size of the component is fixed to the size of the content and [UIObject.setSize\(\)](#) has no effect.

Using styles with the Loader component

The Loader component uses the following styles.

Style	Theme	Description
<code>borderStyle</code>	Both	The Loader component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063 . The default border style is <code>"none"</code> .

For example:

```
my_ldr.setStyle("backgroundColor", 0xEEEEEE);
```

For more information, see “Using styles to customize component color and text” in *Using Components*.

Using skins with the Loader component

The Loader component uses an instance of `RectBorder` for its border (see [“RectBorder class” on page 1063](#)).

Loader class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > Loader

ActionScript Class Name mx.controls.Loader

The properties of the Loader class let you set content to load and monitor its loading progress at runtime.

Setting a property of the Loader class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Loader.version);
```

NOTE

The code `trace(myLoaderInstance.version);` returns `undefined`.

Method summary for the Loader class

The following table lists the method of the Loader class.

Method	Description
Loader.load()	Loads the content specified by the <code>contentPath</code> property.

Methods inherited from the UIObject class

The following table lists the methods the Loader class inherits from the UIObject class. When calling these methods from the Loader object, use the form `LoaderInstance.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.

Method	Description
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Loader class inherits from the UIComponent class. When calling these methods from the Loader object, use the form *LoaderInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the Loader class

The following table lists properties of the Loader class.

Property	Description
<code>Loader.autoLoad</code>	A Boolean value that indicates whether the content loads automatically (<code>true</code>) or you must call <code>Loader.load()</code> (<code>false</code>).
<code>Loader.bytesLoaded</code>	A read-only property that indicates the number of bytes that have been loaded.
<code>Loader.bytesTotal</code>	A read-only property that indicates the total number of bytes in the content.
<code>Loader.content</code>	A reference to the content of the loader. This property is read-only.
<code>Loader.contentPath</code>	A string that indicates the URL of the content to be loaded.
<code>Loader.percentLoaded</code>	A number that indicates the percentage of loaded content. This property is read-only.
<code>Loader.scaleContent</code>	A Boolean value that indicates whether the content scales to fit the loader (<code>true</code>), or the loader scales to fit the content (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the Loader class inherits from the UIObject class.

When accessing these properties from the Loader object, use the form

LoaderInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Loader class inherits from the UIComponent class.

When accessing these properties from the Loader object, use the form

LoaderInstance.propertyName.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the Loader class

The following table lists events of the Loader class.

Event	Description
<code>Loader.complete</code>	Triggered when the content finished loading.
<code>Loader.progress</code>	Triggered while content is loading.

Events inherited from the UIObject class

The following table lists the events the Loader class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Loader class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Loader.autoLoad

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.autoLoad

Description

Property; a Boolean value that indicates whether to automatically load the content (`true`), or wait until `Loader.load()` is called (`false`). The default value is `true`.

Example

The following code sets up the loader component to wait for a `Loader.load()` call:

```
loader.autoLoad = false;
```

Loader.bytesLoaded

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.bytesLoaded

Description

Property (read-only); the number of bytes of content that have been loaded. The default value is 0 until content begins loading.

Example

With a `Loader` component and a `ProgressBar` component in the library of the current document, the following code creates progress bar and loader instances. It then creates a listener object with a `progress` event handler that shows the progress of the load. The listener is registered with the `my_loader` instance.

When you create an instance with `createClassObject()`, you have to position it on the Stage with `move()`. See [UIObject.move\(\)](#).

```
import mx.controls.Loader;
import mx.controls.ProgressBar;

System.security.allowDomain("http://www.flash-mx.com");

this.createClassObject(Loader, "my_ldr", 10);
this.createClassObject(ProgressBar, "my_pb", 20, {source:"my_ldr"});

my_ldr.move(1, 50);
my_pb.move(1, 1);

var loaderListener:Object = new Object();
loaderListener.progress = function(evt_obj:Object) {
    // evt_obj.target is the component that generated the progress event,
    // that is, the loader.
    my_pb.setProgress(my_ldr.bytesLoaded, my_ldr.bytesTotal);
    // Show progress.
};
my_ldr.addEventListener("progress", loaderListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

Loader.bytesTotal

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.bytesTotal

Description

Property (read-only); the size of the content, in bytes. The default value is 0 until content begins loading.

Example

The following code creates a progress bar and a Loader component. It then creates a load listener object with a progress event handler that shows the progress of the load. The listener is registered with the `my_ldr` instance, as follows:

```
import mx.controls.Loader;
import mx.controls.ProgressBar;
this.createClassObject(ProgressBar, "my_pb", 998);
this.createClassObject(Loader, "my_ldr", 999);
my_pb.move(1, 1);
my_ldr.move(1, 50);
my_pb.source = "my_ldr";
var loadListener:Object = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // that is, the loader.
    my_pb.setProgress(my_ldr.bytesLoaded, my_ldr.bytesTotal); // Show
    progress.
}
my_ldr.addEventListener("progress", loadListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

See also

[Loader.bytesLoaded](#)

Loader.complete

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObj:Object){
    // ...
};
loaderInstance.addEventListener("complete", listenerObject);
```

Usage 2:

```
on (complete) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the content finishes loading.

The first usage example uses a dispatcher/listener event model. A component instance (*loaderInstance*) dispatches an event (in this case, *complete*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a Loader instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the Loader instance `myLoaderComponent`, sends “`_level0.myLoaderComponent`” to the Output panel:

```
on (complete) {
    trace(this);
}
```

Example

The following example creates a Loader component, `my_ldr`, and then defines a listener object for a `complete` event. The example loads an image from a web page; when loading is complete, the listener displays a message in the Output panel.

Drag a Loader component to the library, then add the following code to the first frame of the timeline.

```
/**
 * Requires:
 *   - Loader component in Library.
 */

System.security.allowDomain("http://www.flash-mx.com");

//Create loader instance.
this.createClassObject(mx.controls.Loader, "my_ldr", 10);

//Create listener object.
var loaderListener:Object = new Object();
loaderListener.complete = function(evt_obj:Object){
```

```
        trace("loading complete");
    }

    //Add listener.
    my_ldr.addEventListener("complete", loaderListener);
    my_ldr.load("http://www.flash-mx.com/images/image2.jpg");
```

Loader.content

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.content

Description

Property (read-only); a reference to a movie clip instance that contains the contents of the loaded file. The value is `undefined` until the load begins. Set properties for the content within an event handler function for the `Loader.complete` event.

Example

The Loader component has a “complete” event so you can make sure the content is completely loaded before trying to access properties of the loader’s content.

The following example uses the `Loader.content` property within an event handler function for the complete event. Drag a Loader component from the Components panel to the current document’s library, so the component appears in the library. Then add the following ActionScript to the first frame of the main timeline:

```
this.createClassObject(mx.controls.Loader, "my_ldr", 10);
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";
//Assign a variable to the content.
var content_mc:MovieClip = my_ldr.content;

var loadtest:Object = new Object();
loadtest.complete = function(){
//Set properties for the content.
    content_mc._alpha = 50;
    content_mc._rotation= 45;
    trace(content_mc._width);
}
my_ldr.addEventListener("complete", loadtest);
```

Loader.contentPath

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.contentPath

Description

Property; a string that indicates an absolute or relative URL of the file to load into the loader. A relative path must be relative to the SWF file that loads the content. The URL must be in the same subdomain as the loading SWF file.

If you are using Flash Player or test mode in Flash, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive information.

Example

The following example tells the loader instance to display the contents of the logo.swf file:

```
flower.contentPath = "http://www.flash-mx.com/images/image1.jpg"
```

The following example unloads content from the Loader when the button instance my_btn is clicked:

```
flower.contentPath = "http://www.flash-mx.com/images/image1.jpg"
function clicked(){
    flower.contentPath = "";
}
my_btn.addEventListener("click", clicked);
```

Loader.load()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.load([*path*])

Parameters

path An optional parameter that specifies the value for the `contentPath` property before the load begins. If a value is not specified, the current value of `contentPath` is used as is.

Returns

Nothing.

Description

Method; tells the loader to begin loading its content.

Example

The following example creates a `Loader` instance, `my_ldr`, and a `Button` instance and sets the loader `autoLoad` property to `false` so that loading does not begin until a call to the `load()` method is made. Next the example sets `contentPath` to the web location of an image and creates a listener for a `click` event on the button. When the user clicks the button, the event handler calls `my_ldr.load()` to load the image. The event handler also disables the button.

Drag a `Loader` component and a `Button` component from the Component panel to the Library, then add the following code to the first frame of the timeline.

```
/**
 * Requires:
 * - Loader component in Library.
 * - Button component in Library.
 */

System.security.allowDomain("http://www.flash-mx.com");

//Create loader instance.
this.createClassObject(mx.controls.Loader, "my_ldr", 10);
this.createClassObject(mx.controls.Button, "load_button", 20, {label:"Load
    image"});

my_ldr.move(0, 30);

my_ldr.autoLoad = false;
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";

var loadListener:Object = new Object();
loadListener.click = function (evt_obj:Object) {
    my_ldr.load();
    load_button.enabled = false;
}
load_button.addEventListener("click", loadListener);
```

Loader.percentLoaded

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.percentLoaded

Description

Property (read-only); a number indicating what percent of the content has loaded. Typically, this property is used to present the progress to the user in an easily readable form. Use the following code to round the figure to the nearest integer:

```
Math.round(bytesLoaded/bytesTotal*100)
```

Example

The following example creates a Loader instance and then creates a listener object with a progress handler that traces the percent loaded and sends it to the Output panel:

```
import mx.controls.Loader;
this.createClassObject(Loader, "my_ldr", 999);
var loadListener:Object = new Object();
loadListener.progress = function(eventObj) {
    // eventObj.target is the component that generated the progress event,
    // that is, the loader.
    trace("The image is "+my_ldr.percentLoaded+"% loaded.");
    // Track loading progress.
};
my_ldr.addEventListener("progress", loadListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

Loader.progress

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
    // ...
};
loaderInstance.addEventListener("progress", listenerObject);
```

Usage 2:

```
on (progress) {
    // ...
}
```

Description

Event; broadcast to all registered listeners while content is loading. This event occurs when the load is triggered by the `autoload` parameter or by a call to `Loader.load()`. The `progress` event is not always broadcast, and the `complete` event may be broadcast without any `progress` events being dispatched. This can happen if the loaded content is a local file.

The first usage example uses a dispatcher/listener event model. A component instance (`loaderInstance`) dispatches an event (in this case, `progress`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `Loader` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `Loader` instance `myLoaderComponent`, sends “`_level0.myLoaderComponent`” to the Output panel:

```
on (progress) {
    trace(this);
}
```

Example

The following code creates a `Loader` instance and then creates a listener object with an event handler for the `progress` event that sends a message to the Output panel telling what percent of the content has loaded:

```
//Create loader instance.
this.createClassObject(mx.controls.Loader, "my_ldr", 10);

//Create listener object.
var loaderListener:Object = new Object();
loaderListener.progress = function(evt_obj:Object){
    // evt_obj.target is the component that generated the progress event,
    // that is, the loader.
    trace("image is " + my_ldr.percentLoaded + "% loaded.");
}

//Add Listener.
my_ldr.addEventListener("progress", loaderListener);

//Assign content path of loader.
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";
```

Loader.scaleContent

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.scaleContent

Description

Property; indicates whether the content scales to fit the loader (`true`), or the loader scales to fit the content (`false`). The default value is `true`.

Example

The following code tells the loader to resize itself to match the size of its content:

```
my_ldr.scaleContent = false;
```

Media components (Flash Professional only)

The streaming media components make it easy to incorporate streaming media into Macromedia Flash presentations. These components let you present your media in a variety of ways.

You can use the following three media components:

- The `MediaDisplay` component lets media stream into your Flash content without a supporting user interface. You can use this component with video and audio data. When you use this component by itself, the user has no control over the media.
- The `MediaController` component provides standard user interface controls (play, pause, and so on) for media playback. Media is never loaded into or played by this component; it is used only for controlling playback in a `MediaPlayback` or `MediaDisplay` instance. The `MediaController` component features a “drawer,” which displays the contents of the playback controls when the mouse is positioned over the component.
- The `MediaPlayback` component is a combination of the `MediaDisplay` and `MediaController` components; it provides methods to stream your media content.

Bear in mind these points about media components:

- The media components require Flash Player 6 or later. In Flash Player 6, media components support FLV files only through Flash Communication Server, not through HTTP.
- The media components do not support scan forward and scan backward functionality. However, you can effect this functionality by moving the playback slider.
- Only component size and controller policy are reflected in the live preview.
- The media components do not support accessibility.

Interacting with media components (Flash Professional only)

The streaming `MediaPlayer` and `MediaController` components respond to mouse and keyboard activity; the `MediaDisplay` component does not. The following table summarizes the actions for the `MediaPlayer` and `MediaController` components upon receiving focus:

Target	Navigation	Description
Playback controls of a given controller	Mouse rollover	The button is highlighted.
Playback controls of a given controller	Single click of left mouse button	Users can click the playback controls to manipulate the playback of audio and video media. The Pause/Play and Go to Beginning/Go to End buttons behave as standard buttons. When the mouse button is pressed, the onscreen button highlights in its pressed state, and when the mouse button is released, the onscreen button reverts to its unselected appearance. The Go to End button is disabled when FLV media files are playing.
Slider controls of a given controller	Move slider back and forth	The playbar indicates the user's position within the media; the playback slider moves horizontally (by default) to indicate the playback from beginning (left) to end (right). The slider moves from bottom to top when the controls are oriented vertically. As the slider moves from left to right, it highlights the previous display space to indicate that this content has been played back or selected. Display space ahead of the slider remains unhighlighted until the slider passes. Users can drag the slider to affect the media's playback position. If media is playing, automatic playback begins from the point at which the mouse is released. If the media is paused, the user can move and release the slider and the media remains paused. There is also a volume slider, which moves from left (muted) to right (maximum volume) in both the horizontal and vertical layouts.

Target	Navigation	Description
Playback controller navigation	Tab, Shift+Tab	Moves the focus from button to button within the controller component, where the focused element becomes highlighted. This navigation works with the Pause/Play, Go to Beginning, Go to End, Volume Mute, and Volume Max controls. The focus moves from left to right and top to bottom as users tab through the elements. Shift+Tab moves focus from right to left and bottom to top. Upon receiving focus through the Tab key, the control immediately passes focus to the Play/Pause button. When focus is on the Volume Max button, and then Tab is pressed, the focus moves to the next control in the tab index on the Stage.
A given control button	Spacebar or Enter/Return	Selects the element in focus. On press, the button appears in its pressed state. On release, the button reverts back to its focused, mouse-over state.

Understanding media components (Flash Professional only)

This section provides an overview of how the media components work. Most of the properties listed in this section can be set with the Component inspector. (See [“Using the Component inspector with media components” on page 840.](#))

Apart from the layout properties discussed later in this section, the following properties can be set for the `MediaDisplay` and `MediaPlayerback` components:

- The media type, which can be set to MP3 or FLV (see `Media.mediaType` and `Media.setMedia()`).
- The relative or absolute content path, which holds the media file to be streamed (see `Media.contentPath`).
- Cue point objects, along with their name, time, and player properties (see `Media.addCuePoint()` and `Media.cuePoints`). The name of the cue point is arbitrary; use a name that will have meaning when using listener and trace events. A cue point broadcasts a `cuePoint` event when the value of its time property is equal to that of the playhead location of the `MediaPlayerback` or `MediaDisplay` component with which it is associated. The player property is a reference to the `MediaPlayerback` instance with which it is associated. You can remove cue points by using `Media.removeCuePoint()` and `Media.removeAllCuePoints()`.

The streaming media components broadcast several related events. The following broadcast events can be used to set other items in motion:

- A `change` event is broadcast continuously by the `MediaDisplay` and `MediaPlayback` components while media is playing. (See [Media.change](#).)
- A `progress` event is continuously broadcast by the `MediaDisplay` and `MediaPlayback` components while media is loading. (See [Media.progress](#).)
- A `click` event is broadcast by the `MediaController` and `MediaPlayback` components whenever the user clicks the Pause/Play button. In this case, the `detail` property of the event object provides information on which button was clicked. (See [Media.click](#).)
- A `volume` event is broadcast by the `MediaController` and `MediaPlayback` components when the user adjusts the volume controls. (See [Media.volume](#).)
- A `playheadChange` event is broadcast by the `MediaController` and `MediaPlayback` components when the user moves the playback slider or when the Go to Beginning or Go to End buttons are clicked. (See [Media.playheadChange](#).)

The `MediaDisplay` component works with the `MediaController` component. Combined, the components behave in a manner similar to the `MediaPlayback` component, but they give you more flexibility in the look and feel of your media presentation.

Understanding the `MediaDisplay` component

When you place a `MediaDisplay` component on the Stage, it has no user interface. It is simply a container to hold and play media. The following properties affect the appearance of video media playing in a `MediaDisplay` component:

- [Media.aspectRatio](#)
- [Media.autoSize](#)
- Height (in the Property inspector)
- Width (in the Property inspector)

NOTE

The user does not see anything unless some media is playing.

The [Media.aspectRatio](#) property takes precedence over the other properties. When [Media.aspectRatio](#) is set to `true` (the default), the component always readjusts the size of the playing media to maintain the media's aspect ratio.

For FLV files, when `Media.autoSize` is set to `true`, the media is displayed at its preferred size, regardless of the size of the component. This means that if the size of the `MediaDisplay` instance is different from the size of the media, the media either spills out of the instance boundaries or does not fill the instance size. When `Media.autoSize` is set to `false`, Flash uses the instance size as much as possible, while honoring the aspect ratio. If both `Media.autoSize` and `Media.aspectRatio` are set to `false`, the exact size of the component is used.

NOTE

Since there is no image to show with MP3 files, setting `Media.autoSize` would have no effect. For MP3 files, the minimum usable size is 60 pixels high by 256 pixels wide in the default orientation.

The `MediaDisplay` component also supports the `Media.volume` property. This property takes on integer values from 0 (mute) to 100 (maximum volume). The default setting is 75.

Understanding the `MediaController` component

The interface for the `MediaController` component depends on its `Media.controllerPolicy` and `Media.backgroundColor` properties. The `Media.controllerPolicy` property determines if the media control set is always visible, collapsed, or only visible when the mouse hovers over the control portion of the component. When collapsed, the controller draws a modified progress bar, which is a combination of the loadbar and the playbar. It shows the progress of the bytes being loaded at the bottom of the bar, and the progress of the playhead just above it. When expanded, the controller draws an enhanced version of the playbar/loadbar, which contains the following items:

- Text labels on the left that indicate the playback state (streaming or paused), and on the right that indicate playhead location, in seconds
- A playhead location indicator
- A slider, which users can drag to navigate through the media

The `MediaController` component also provides the following items:

- A Play/Pause button
- Go to Beginning and Go to End buttons, which navigate to the beginning and end of the media, respectively
- A volume control that consists of a slider, a mute button, and a maximum volume button

Both the collapsed and expanded states of the `MediaController` component use the `Media.backgroundColor` property. This property determines whether the controller draws a chrome background (the default) or allows the media background to display from behind the controls.

The `MediaController` component has an orientation setting, `Media.horizontal`, which you can use to draw the component with a horizontal orientation (the default) or a vertical one. With a horizontal orientation, the playbar tracks playing media from left to right. With a vertical orientation, the playbar tracks media from bottom to top.

You can associate the `MediaDisplay` and `MediaController` components with each other by using the `Media.associateDisplay()` and `Media.associateController()` methods. These methods allow the `MediaController` instance to update its controls based on events broadcast from the `MediaDisplay` instance, and allow the `MediaDisplay` component to react to user settings in the `MediaController`.

Understanding the `MediaPlayback` component

The `MediaPlayback` contains the `MediaController` and `MediaDisplay` subcomponents. The `MediaController` and `MediaDisplay` portions always scale to fit the size of the overall `MediaPlayback` instance.

The `MediaPlayback` component uses `Media.controlPlacement` to determine the layout of the controls. By setting this property to `top`, `bottom`, `left`, or `right`, you can indicate where the controls are drawn in relation to the display. For example, a value of `right` gives a control a vertical orientation and positions it on the right of the display.

Using media components (Flash Professional only)

With the sharp increase in the use of media to provide information to web users, many developers want their users to be able to stream media and then control it. You might use media components in the following kinds of situations:

- Showing media that introduces a company
- Streaming movies or movie previews
- Streaming songs or song snippets
- Providing learning material in the form of media

Using the `MediaPlayback` component

Suppose you must develop a website that allows users to preview DVDs and CDs that you sell in a rich media environment. The following example shows the steps involved. (It assumes your website is ready for inserting streaming components.)

To create a Flash document that displays a CD or DVD preview:

1. Select File > New; then select Flash Document.
2. Open the Components panel and double-click the MediaPlayer component to place an instance of it on the Stage.
3. Select the MediaPlayer component instance and enter the instance name **myMedia** in the Property inspector.
4. In the Component inspector, set your media type according to the type of media that will be streaming (MP3 or FLV).
5. If you selected FLV, enter the duration of the video in the Video Length text boxes; use the format *HH:MM:SS*.
6. Enter the location of your preview video in the URL text box. For example, you might enter **www.helpexamples.com/flash/video/clouds.flv**.
7. Set the desired options for the Automatically Play, Use Preferred Media Size, and Respect Aspect Ratio check boxes.
8. Set the control placement to the desired side of the MediaPlayer component.
9. Add a cue point toward the end of the media by clicking the Add (+) button; this cue point is used with a listener to open a pop-up window that announces that the movie is on sale. Give the cue point the name **cuePointName** and a position near the end of your media duration.
10. Drag a Window component from the Components panel to the current document's library.

This places a symbol called Window in your library, and makes the Window component available to your SWF file at runtime.
11. Create a text box and write some text informing the user that the movie is on sale.
12. Select Modify > Convert to Symbol to convert the text box to a movie clip, and name it **mySale_mc**.
13. Right-click (Windows) or Control-click (Macintosh) the **mySale_mc** movie clip in the library, select Linkage, and select Export for ActionScript.

This places the movie clip in your runtime library.

14. Add the following ActionScript to Frame 1. This code creates a listener to open a pop-up window informing the user that the movie is on sale.

```
// Import the classes necessary to create the pop-up window dynamically.

import mx.containers.Window;
import mx.managers.PopUpManager;

// Create a listener object to open sale pop-up.
var saleListener:Object = new Object();

saleListener.cuePoint = function(eventObj:Object) {

var saleWin:MovieClip = PopUpManager.createPopUp(_root, Window, false,
    {closeButton:true, title:"Movie Sale", contentPath:"mySale_mc"});

// Enlarge the window so that the content fits.

saleWin.setSize(80, 80);
var delSaleWin:Object = new Object();
delSaleWin.click = function(eventObj:Object) {
    saleWin.deletePopUp();
}
saleWin.addEventListener("click", delSaleWin);

}

myMedia.addEventListener("cuePoint", saleListener);
```

15. Select Control > Test Movie to test the SWF file.

When the application reaches the playback time of the cuePointName cue point, a window pops up to show your message.

Using the MediaDisplay and MediaController components

If you want a lot of control over the look and feel of your media display, you may want to use the MediaDisplay and MediaController components together. The following example creates a Flash application that displays your CD and DVD preview media.

To create a Flash document that displays a CD or DVD preview:

1. In Flash, select File > New; then select Flash Document.
2. From the Components panel, drag a MediaController *and* a MediaDisplay component to the Stage.
3. Select the MediaDisplay instance and enter the instance name **myDisplay** in the Property inspector.
4. Select the MediaController instance and enter the instance name **myController** in the Property inspector.
5. Select the MediaDisplay instance, and open the Component inspector, Parameters tab. Set your media type according to the type of media that will be streaming (MP3 or FLV).
6. If you selected FLV, enter the duration of the video in the Video Length text boxes using the format *HH:MM:SS*.
7. Enter the location of your preview video in the URL text box. For example, you might enter www.helpexamples.com/flash/video/clouds.flv.
8. Set the desired options for the Automatically Play, Use Preferred Media Size, and Respect Aspect Ratio check boxes.
9. Select the MediaController instance and, in the Component inspector, Parameters tab, set your orientation to vertical by setting the `horizontal` property to `false`.
10. In the Component inspector, Parameters tab, set `backgroundStyle` to `none`.
This specifies that the MediaController instance should not draw a background but should instead fill the media between the controls.
Next, you'll use a behavior to associate the MediaController and MediaDisplay instances so that the MediaController instance accurately reflects the playhead movement and other settings in the MediaDisplay instance, and so that the MediaDisplay instance responds to user clicks.
11. With the MediaController instance still selected, open the Behaviors panel (Window > Behaviors).
12. In the Behaviors panel, click the Add (+) button, and select Media > Associate Display.
13. In the Associate Display window, select `myDisplay` under `_root`, and click OK.

For more information on using behaviors with media components, see [“Controlling media components by using behaviors” on page 841](#).

Using the Component inspector with media components

The Component inspector makes it easy to set media component parameters, properties, and so on. To use this panel, click the desired component on the Stage and, with the Property inspector open, click Launch Component Inspector. The Component inspector can be used for the following purposes:

- To automatically play the media (see [Media.activePlayControl](#) and [Media.autoPlay](#))
- To keep or ignore the media's aspect ratio (see [Media.aspectRatio](#))
- To determine if the media will be automatically sized to fit the component instance (see [Media.autoSize](#))
- To enable or disable the chrome background (see [Media.backgroundStyle](#))
- To specify the path to your media in the form of a URL (see [Media.contentPath](#))
- To specify the visibility of the playback controls (see [Media.controllerPolicy](#))
- To add cue point objects (see [Media.addCuePoint\(\)](#))
- To delete cue point objects (see [Media.removeCuePoint\(\)](#))
- To set the orientation of MediaController instances (see [Media.horizontal](#))
- To set the type of media being played (see [Media.setMedia\(\)](#))
- To set the play time of the FLV media (see [Media.totalTime](#))
- To set the last few digits of the time display to indicate milliseconds or frames per second (fps)

It is important to understand a few concepts when working with the Component inspector:

- The video time control is not available when you select an MP3 video type, because this information is automatically read in when MP3 files are used. For FLV files created with Flash Video Exporter 1.0, you must enter the total time of the media ([Media.totalTime](#)) in order for the playbar of the MediaPlayer component (or any listening MediaController component) to accurately reflect play progress. FLV files created with Flash Video Exporter 1.1 or later set the duration automatically.

- With the file type set to FLV, you'll notice a Milliseconds option and (if Milliseconds is unselected) a Frames Per Second (FPS) pop-up menu. When Milliseconds is selected, the FPS control is not visible. In this mode, the time displayed in the playbar at runtime is formatted as *HH:MM:SS.mmm* (*H* = hours, *M* = minutes, *S* = seconds, *m* = milliseconds), and cue points are set in seconds. When Milliseconds is unselected, the FPS control is enabled and the playbar time is formatted as *HH:MM:SS.FF* (*F* = frames per second), while cue points are set in frames.

NOTE

You can set the FPS value only by using the Component inspector. Setting an fps value by using ActionScript has no effect and is ignored.

Controlling media components by using behaviors

Behaviors are prewritten ActionScript scripts that you add to an instance, such as a `MediaDisplay` component, to control that object. Behaviors let you add the power, control, and flexibility of ActionScript coding to your document without having to create the ActionScript code yourself.

To control a media component with a behavior, you use the Behaviors panel to apply the behavior to a given media component instance. You specify the event that triggers the behavior (such as reaching a specified cue point), select a target object (the media components that are affected by the behavior), and, if necessary, select settings for the behavior (such as the movie clip within the media to navigate to).

The following behaviors are packaged with Flash Professional 8 and are used to control embedded media components.

Behavior	Purpose	Parameters
Associate Controller	Associates a <code>MediaController</code> component with a <code>MediaDisplay</code> component	Instance name of target <code>MediaController</code> components
Associate Display	Associates a <code>MediaDisplay</code> component with a <code>MediaController</code> component	Instance name of target <code>MediaController</code> components
Labeled Frame CuePoint Navigation	Places an action on a <code>MediaDisplay</code> or <code>MediaPlayback</code> instance that tells an indicated movie clip to navigate to a frame with the same name as a given cue point	Name of frame and name of cue point (the names should be equal)
Slide CuePoint Navigation	Makes a slide-based Flash document navigate to a slide with the same name as a given cue point	Name of slide and name of cue point (the names should be equal)

To associate a **MediaDisplay** component with a **MediaController** component:

1. Place a **MediaDisplay** instance and a **MediaController** instance on the Stage.
2. Select the **MediaDisplay** instance and, using the Property inspector, enter the instance name **myMediaDisplay**.
3. Select the **MediaController** instance that will trigger the behavior.
4. In the Behaviors panel, click the Add (+) button and select **Media > Associate Display**.
5. In the Associate Display window, select **myMediaDisplay** under `_root` and click OK.

NOTE

If you have associated the **MediaDisplay** component with the **MediaController** component, you do not need to associate the **MediaController** component with the **MediaDisplay** component.

To associate a **MediaController** component with a **MediaDisplay** component:

1. Place a **MediaDisplay** instance and a **MediaController** instance on the Stage.
2. Select the **MediaController** instance and, using the Property inspector, enter the instance name **myMediaController**.
3. Select the **MediaDisplay** instance that will trigger the behavior.
4. In the Behaviors panel, click the Add (+) button and select **Media > Associate Controller**.
5. In the Associate Controller window, select **myMediaController** under `_root` and click OK.

To use a **Labeled Frame CuePoint Navigation** behavior:

1. Place a **MediaDisplay** or **MediaPlayback** component instance on the Stage.
2. Select the desired frame that you want the media to navigate to and, using the Property inspector, enter the frame name **myLabeledFrame**.
3. Select your **MediaDisplay** or **MediaPlayback** instance.
4. In the Component inspector, click the Add (+) button and enter the cue point time in the format *HH:MM:SS:mmm* or *HH:MM:SS:FF*, and give the cue point the name **myLabeledFrame**.

The cue point indicates the amount of time that should elapse before you navigate to the selected frame. For example, if you want to jump to `myLabeledFrame` 5 seconds into the media, enter **5** in the **SS** text box and enter **myLabeledFrame** in the **Name** text box.

5. In the Behaviors panel, click the Add (+) button and select **Media > Labeled Frame CuePoint Navigation**.
6. In the Labeled Frame CuePoint Navigation window, select the `_root` clip and click OK.

To use a Slide CuePoint Navigation behavior:

1. Open your new document as a Flash slide presentation.
2. Place a `MediaDisplay` or `MediaPlayback` component instance on the Stage.
3. In the Screen Outline pane to the left of the Stage, click the Insert Screen (+) button to add a second slide; then select the second slide and rename it **mySlide**.
4. Select your `MediaDisplay` or `MediaController` instance.
5. In the Component inspector, click the Add (+) button and enter the cue point time in the format *HH:MM:SS:mmm* or *HH:MM:SS:FF*, and give the cue point the name **MySlide**.
The cue point indicates the amount of time that should elapse before you navigate to the selected slide. For example, if you want to jump to `mySlide` 5 seconds into the media, enter 5 in the SS text box and enter **mySlide** in the Name text box.
6. In the Behaviors panel, click the Add (+) button and select Media > Slide CuePoint Navigation.
7. In the Slide CuePoint Navigation window, select `Presentation` under the `_root` clip and click OK.

Media component parameters (Flash Professional only)

The following tables list `MediaDisplay`, `MediaController`, and `MediaPlayback` authoring parameters that you can set for a given media component instance in the Property inspector.

MediaDisplay parameters

Name	Type	Default value	Description
Automatically Play (Media.autoPlay)	Boolean	Selected	Determines if the media plays as soon as it has loaded.
Use Preferred Media Size (Media.autoSize)	Boolean	Selected	Determines whether the media associated with the <code>MediaDisplay</code> instance conforms to the component size or simply uses its default size.
FPS	Integer	30	Indicates the number of frames per second. When the Milliseconds option is selected, this control is disabled.

Name	Type	Default value	Description
Cue Points (Media.cuePoints)	Array	Undefined	An array of cue point objects, each with a name and position in time in a valid <i>HH:MM:SS:FF</i> (Milliseconds option selected) or <i>HH:MM:SS:mmm</i> format.
FLV or MP3 (Media.mediaType)	FLV or MP3	FLV	Designates the type of media to be played.
Milliseconds	Boolean	Unselected	Determines whether the playbar uses frames or milliseconds, and whether the cue points use seconds or frames. When this option is selected, the FPS control is not visible.
URL (Media.contentPath)	String	Undefined	A string that holds the path and filename of the media to be played.
Video Length (Media.totalTime)	Integer	Undefined	The total time needed to play the FLV media. This setting is required in order for the playbar to work correctly. This control is only visible when the media type is set to FLV.

MediaController parameters

Name	Type	Default value	Description
activePlayControl (Media.activePlayControl)	String: pause or play	pause	Determines whether the playbar is in play or pause mode upon instantiation. This mode determines the image displayed on the Play/Pause button, which is the opposite of the playing/paused state that the controller is actually in.
backgroundStyle (Media.backgroundStyle)	String: default or none	default	Determines whether the chrome background is drawn for the MediaController instance.
controllerPolicy (Media.controllerPolicy)	String: auto, on, or off	auto	Determines whether the controller opens or closes according to mouse position, or is locked in the open or closed state.

Name	Type	Default value	Description
horizontal (Media.horizontal)	Boolean	true	Determines whether the controller portion of the instance is vertically or horizontally oriented. A <code>true</code> value indicates that the component has a horizontal orientation.
enabled	Boolean	true	Determines whether this control can be modified by the user. A <code>true</code> value indicates that the control can be modified.
visible	Boolean	true	Determines whether this control is viewable by the user. A <code>true</code> value indicates that the control is viewable.

MediaPlayer parameters

Name	Type	Default value	Description
Control Placement (Media.controlPlacement)	String:	bottom	Position of the controller. The value is related to orientation.
		top, bottom, left, right	
Control Visibility (Media.controllerPolicy)	Boolean	true	Determines whether the controller opens or closes according to mouse position.
Automatically Play (Media.autoPlay)	Boolean	true	Determines whether the media plays as soon as it loads.
Use Preferred Media Size (Media.autoSize)	Boolean	true	Determines whether the <code>MediaController</code> instance sizes to fit the media or uses other settings.
FPS	Integer	30	Number of frames per second. When the <code>Milliseconds</code> option is selected, this control is disabled.
Cue Points (Media.cuePoints)	Array	undefined	An array of cue point objects, each with a name and position in time in a valid <code>HH:MM:SS:mmm</code> (<code>Milliseconds</code> option selected) or <code>HH:MM:SS:FF</code> format.

Name	Type	Default value	Description
FLV or MP3 (Media.mediaType)	String: FLV or MP3	FLV	Designates the type of media to be played.
Milliseconds	Boolean	false	Determines whether the playbar uses frames or milliseconds, and whether the cue points use seconds or frames. When this option is selected, the FPS control is disabled.
URL (Media.contentPath)	String	undefined	A string that holds the path and filename of the media to be played.
Video Length (Media.totalTime)	Integer	undefined	The total time needed to play the FLV media. This setting is required for the playbar to work correctly.

Creating applications with media components (Flash Professional only)

Creating Flash content by using media components is quite simple and often requires only a few steps. This example shows how to create an application to play a small, publicly available media file.

To add a media component to an application:

1. In Flash, select File > New; then select Flash Document.
2. In the Components panel, double-click the MediaPlayer component to add it to the Stage.
3. In the Property inspector, do the following:
 - Enter the instance name **myMedia**.
 - Click Launch Component Inspector.
4. In the Component inspector, enter <http://www.helpexamples.com/flash/video/water.flv> in the URL text box.
5. Select Control > Test Movie to see the media play.

Customizing media components (Flash Professional only)

If you want to change the appearance of your media components, you can use skinning. For a complete guide to component customization, see Chapter 5, “Customizing Components” in *Using Components*.

Using styles with media components

The media components do not use styles.

Using skins with media components

The media components do not support dynamic skinning, although you can open the media component source document and change their assets to achieve the desired look. It is best to make a copy of this file and work from the copy, so that you always have the installed source to go back to. You can find the media component source document at the following locations:

- Windows: C:\Program Files\Macromedia\Flash 8\<language>\Configuration\ComponentFLA\MediaComponents fla
- Macintosh: HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/MediaComponents fla

Media class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > Media

ActionScript Class Names mx.controls.MediaController, mx.controls.MediaDisplay, mx.controls.MediaPlayback

Each component class has a `version` property, which is a class property. Class properties are available only for the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.MediaPlayback.version);
```

NOTE

The code `trace(myMediaInstance.version);` returns `undefined`.

Method summary for the Media class

The following table lists methods of the Media class.

Method	Components	Description
<code>Media.addCuePoint()</code>	MediaDisplay, MediaPlayback	Adds a cue point object to the component instance.
<code>Media.associateController()</code>	MediaDisplay	Associates a MediaDisplay instance with a MediaController instance.
<code>Media.associateDisplay()</code>	MediaController	Associates a MediaController instance with a MediaDisplay instance.
<code>Media.displayFull()</code>	MediaPlayback	Converts the component instance to full-screen playback mode.
<code>Media.displayNormal()</code>	MediaPlayback	Converts the component instance back to its original screen size.
<code>Media.getCuePoint()</code>	MediaDisplay, MediaPlayback	Returns a cue point object.
<code>Media.pause()</code>	MediaDisplay, MediaPlayback	Pauses the playhead at its current location in the media timeline.
<code>Media.play()</code>	MediaDisplay, MediaPlayback	Plays the media associated with the component instance at a given starting point.
<code>Media.removeAllCuePoints()</code>	MediaDisplay, MediaPlayback	Deletes all cue point objects associated with a given component instance.
<code>Media.removeCuePoint()</code>	MediaDisplay, MediaPlayback	Deletes a specified cue point associated with a given component instance.
<code>Media.setMedia()</code>	MediaDisplay, MediaPlayback	Sets the media type and path to the specified media type.
<code>Media.stop()</code>	MediaDisplay, MediaPlayback	Stops the playhead and moves it to position 0, which is the beginning of the media.

Property summary for the Media class

The following table lists properties of the Media class.

Property	Components	Description
Media.activePlayControl	MediaController	Determines the component state when loaded at runtime.
Media.aspectRatio	MediaDisplay, MediaPlayback	Determines if the component instance maintains its video aspect ratio.
Media.autoPlay	MediaDisplay, MediaPlayback	Determines if the component instance immediately starts to buffer and play.
Media.autoSize	MediaDisplay, MediaPlayback	Determines the size of the media-viewing portion of the MediaDisplay or MediaPlayback component.
Media.backgroundColor	MediaController	Determines if the component instance draws its chrome background.
Media.bytesLoaded	MediaDisplay, MediaPlayback	Read-only; the number of bytes loaded that are available for playing.
Media.bytesTotal	MediaDisplay, MediaPlayback	The number of bytes to be loaded into the component instance.
Media.contentPath	MediaDisplay, MediaPlayback	A string that holds the relative path and filename of the media to be streamed and played.
Media.controllerPolicy	MediaController , MediaPlayback	Determines whether the controller is hidden when instantiated and only appears when the user moves the mouse over the controller's collapsed state.
Media.controlPlacement	MediaPlayback	Determines where the component's controls are positioned.
Media.cuePoints	MediaDisplay, MediaPlayback	An array of cue point objects that have been assigned to a given component instance.
Media.horizontal	MediaController	Determines the orientation of the component instance.
Media.mediaType	MediaDisplay, MediaPlayback	Determines the type of media to be played.
Media.playheadTime	MediaDisplay, MediaPlayback	Holds the current position of the playhead (in seconds) for the media timeline that is playing.

Property	Components	Description
<code>Media.playing</code>	MediaDisplay, MediaPlayback, MediaController	For MediaDisplay and MediaPlayback, this property is read-only and returns a Boolean value to indicate whether a given component instance is playing media. For MediaController, this property is read/write and controls the image (playing or paused) displayed on the Play/Pause button of the controller.
<code>Media.preferredHeight</code>	MediaDisplay, MediaPlayback	The default value of the height of a FLV file.
<code>Media.preferredWidth</code>	MediaDisplay, MediaPlayback	The default value of the width of a FLV file.
<code>Media.totalTime</code>	MediaDisplay, MediaPlayback	An integer that indicates the total length of the media, in seconds.
<code>Media.volume</code>	MediaDisplay, MediaPlayback	An integer from 0 (minimum) to 100 (maximum) that represents the volume level.

Event summary for the Media class

The following table lists events of the Media class.

Event	Components	Description
<code>Media.change</code>	MediaDisplay, MediaPlayback	Broadcast continuously while media is playing.
<code>Media.click</code>	MediaController, MediaPlayback	Broadcast when the user clicks the Play/Pause button.
<code>Media.complete</code>	MediaDisplay, MediaPlayback	Notification that the playhead has reached the end of the media.
<code>Media.cuePoint</code>	MediaDisplay, MediaPlayback	Notification that the playhead has reached a given cue point.
<code>Media.playheadChange</code>	MediaController, MediaPlayback	Broadcast by the component instance when a user moves the playback slider or clicks the Go to Beginning or Go to End button.
<code>Media.progress</code>	MediaDisplay, MediaPlayback	Generated continuously until the media has downloaded completely.

Event	Components	Description
<code>Media.scrubbing</code>	MediaController, MediaPlayer	Generated when the playhead is dragged.
<code>Media.volume</code>	MediaController, MediaPlayer	Broadcast when the user adjusts the volume.

Media.activePlayControl

Applies to

MediaController.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`myMedia.activePlayControl`

Description

Property; a string value that specifies the state the MediaController component should be in when it is loaded at runtime. A value of "play" indicates a play state; a value of "pause" indicates a paused state. Set this property and the `autoPlay` property such that both indicate the same state. The default value is "play".

The button image displayed in the MediaController component is the opposite of the current play/pause state. For example, in the play state, the MediaController displays a pause button, because that is what would result from the user clicking the button and toggling the state.

Because it indicates the state that the controller is in when it is loaded, the `activePlayControl` property must be set before the controller is created, either through the Property inspector or the Component inspector, if the component is on the Stage. If the component is being created by ActionScript code, this property must be set in the `initObj` parameter. Changing the value of this property after the component has been created has no effect. The value can be changed only by the user clicking the Play/Pause button.

Example

The following example indicates that the control is paused when first loaded:

```
myMedia.activePlayControl = "pause";
```

See also

[Media.autoPlay](#)

Media.addCuePoint()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.addCuePoint(cuePointName, cuePointTime)
```

Parameters

cuePointName A string that names the cue point.

cuePointTime A number, in seconds, that indicates when a `cuePoint` event is broadcast.

Returns

Nothing.

Description

Method; adds a cue point object to a `MediaPlayer` or `MediaDisplay` instance. When the playhead time equals a cue point time, a `cuePoint` event is broadcast.

Example

The following code adds a cue point called `Homerun` to `myMedia` when the playhead time equals 16 seconds.

```
myMedia.addCuePoint("Homerun", 16);
```

See also

[Media.cuePoint](#), [Media.cuePoints](#), [Media.getCuePoint\(\)](#),
[Media.removeAllCuePoints\(\)](#), [Media.removeCuePoint\(\)](#)

Media.aspectRatio

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.aspectRatio
```

Description

Property; a Boolean value that determines whether a MediaDisplay or MediaPlayer instance maintains its video aspect ratio during playback. A `true` value indicates that the aspect ratio should be maintained; a `false` value indicates that the aspect ratio can change during playback. The default value is `true`.

Example

The following example indicates that the aspect ratio can change during playback:

```
myMedia.aspectRatio = false;
```

Media.associateController()

Applies to

MediaDisplay.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.associateController(instanceName)
```

Parameters

instanceName A string that specifies the instance name of the MediaController component to associate.

Returns

Nothing.

Description

Method; associates a MediaDisplay instance with a MediaController instance.

If you use `Media.associateDisplay()` to associate a MediaController instance with a MediaDisplay instance, you do not need to use `Media.associateController()`.

Example

The following code associates `myMedia` with `myController`:

```
myMedia.associateController(myController);
```

See also

[Media.associateDisplay\(\)](#)

Media.associateDisplay()

Applies to

MediaController.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.associateDisplay(instanceName)
```

Parameters

instanceName A string that specifies the instance name of the MediaDisplay component to associate.

Returns

Nothing.

Description

Method; associates a `MediaController` instance with a `MediaDisplay` instance.

If you associate a `MediaDisplay` instance with a `MediaController` instance by using `Media.associateController()`, you do not need to use `Media.associateDisplay()`.

Example

The following code associates `myMedia` with `myDisplay`:

```
myMedia.associateDisplay(myDisplay);
```

See also

[Media.associateController\(\)](#)

Media.autoPlay

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.autoPlay
```

Description

Property; a Boolean value that determines whether the `MediaPlayback` or `MediaDisplay` instance should immediately start attempting to buffer and play. A `true` value indicates that the control buffers and plays at runtime; a `false` value indicates the control is stopped at runtime. This property depends on the `contentPath` and `mediaType` properties. If `contentPath` and `mediaType` are not set, no playback occurs at runtime. The default value is `true`.

Example

The following example indicates that the control is not started when first loaded:

```
myMedia.autoPlay = false;
```

See also

[Media.contentPath](#), [Media.mediaType](#)

Media.autoSize

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.autoSize

Description

Property; a Boolean value that determines the size of the media-viewing portion of the MediaDisplay or MediaPlayer component.

For the MediaDisplay component, the property behaves as follows:

- If you set this property to `true`, Flash displays the media at its preferred size, regardless of the size of the component. This implies that, unless the MediaDisplay instance size is the same as the size of the media, the media either spills out of the instance boundaries or does not fill the instance.
- If you set this property to `false`, Flash uses the instance size as much as possible, while honoring the aspect ratio. If both `Media.autoSize` and `Media.aspectRatio` are set to `false`, the exact size of the component is used.

For the MediaPlayer component, the property behaves as follows:

- If you set this property to `true`, Flash displays the media at its preferred size unless the media playback area is smaller than the preferred size. If this is the case, Flash shrinks the media to fit inside the instance and respect the aspect ratio. If the preferred size is smaller than the media area of the instance, part of the media area goes unused.
- If you set this property to `false`, Flash uses the instance size as much as possible, while honoring the aspect ratio. If both `Media.autoSize` and `Media.aspectRatio` are set to `false`, the media area of the component is filled. This area is defined as the area above the controls (in the default layout), minus a surrounding 8-pixel margin that makes up the edges of the component.

The default value is `true`.

Example

The following example indicates that the control is not played back according to its media size:

```
myMedia.autoSize = false;
```

See also

[Media.aspectRatio](#)

Media.backgroundColor

Applies to

MediaController.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.backgroundColor
```

Description

Property; a string value that indicates which background is drawn for the MediaController instance. A value of "default" indicates that the chrome background is drawn, and a value of "none" indicates that no chrome background is drawn. The default value is "default".

This is not a style property and therefore is not affected by style settings.

Example

The following example indicates that the chrome background is not drawn for the control:

```
myMedia.backgroundColor = "none";
```

Media.bytesLoaded

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.bytesLoaded

Description

Read-only property; the number of bytes already loaded into the component that are available for playing. The default value is undefined.

Example

The following code creates a variable called `PlaybackLoad` that is set with the number of bytes loaded. The variable is then used in a `for` loop.

```
// Create variable that holds the number of bytes that are loaded.  
var PlaybackLoad:Number = myMedia.bytesLoaded;  
// Perform some function until playback is ready.  
for (PlaybackLoad < 150) {  
    someFunction();  
}
```

Media.bytesTotal

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.bytesTotal

Description

Read-only property; the number of bytes to be loaded into the `MediaPlayer` or `MediaDisplay` component. The default value is `undefined`.

Example

The following example tells the user the size of the media to be streamed:

```
myTextField.text = myMedia.bytesTotal;
```

Media.change

Applies to

`MediaDisplay`, `MediaPlayer`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    // Insert your code here.  
}  
myMedia.addEventListener("change", listenerObject)
```

Description

Event; broadcast by the `MediaDisplay` and `MediaPlayer` components while the media is playing. The percentage complete can be retrieved from the component instance.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Media.change` event's event object has two additional properties:

`target` A reference to the broadcasting object.

`type` The string "change", which indicates the type of event.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example uses an object listener to determine the playhead position (`Media.playheadTime`), from which the percentage complete can be calculated:

```
var myPlayerListener:Object = new Object();
myPlayerListener.change = function(eventObj:Object) {
    var myPosition:Number = myPlayer.playheadTime;
    var myPercentPosition:Number = (myPosition/myPlayer.totalTime);
};
myPlayer.addEventListener("change", myPlayerListener);
```

See also

[Media.playing](#), [Media.pause\(\)](#)

Media.click

Applies to

`MediaController`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object) {
    // ...
};
myMedia.addEventListener("click", listenerObject);
```

Description

Event; broadcast when the user clicks the Play/Pause button. The detail field can be used to determine which button was clicked. The `Media.click` event object has the following properties:

`detail` The string "pause" or "play".

`target` A reference to the `MediaController` or `MediaPlayback` instance.

`type` The string "click".

Example

For a `MediaController` component instance named `myMedia` (and with a `Window` component in the library), the following example opens a pop-up window when the user clicks the Play/Pause button:

```
var myMediaListener:Object = new Object();
myMediaListener.click = function(eventObj:Object) {
    mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true);
};
myMedia.addEventListener("click", myMediaListener);
```

Media.complete

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObj:Object) {
    // ...
};
myMedia.addEventListener("complete", listenerObject);
```

Description

Event; notification that the playhead has reached the end of the media. The `Media.complete` event object has the following properties:

`target` A reference to the `MediaDisplay` or `MediaPlayback` instance.

`type` The string "complete".

Example

The following example uses an object listener to determine when the media has finished playing:

```
var myListener:Object = new Object();
myListener.complete = function(eventObj:Object) {
    trace("media is Finished");
};
myMedia.addEventListener("complete", myListener);
```

Media.contentPath

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.contentPath

Description

Property; a string that holds the relative path and filename of the media to be streamed and/or played. Setting the `contentPath` property is equivalent to calling the `Media.setMedia()` method without specifying a `mediaType` parameter. When no `mediaType` parameter is set with `Media.setMedia()`, the default type is FLV. The default value of the `contentPath` property is undefined.

Example

The following example displays the name of the media playing in a text box:

```
myTextField.text = myMedia.contentPath;
```

See also

[Media.setMedia\(\)](#)

Media.controllerPolicy

Applies to

MediaController, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.controllerPolicy

Description

Property; determines whether the MediaController component (or the controller subcomponent within the MediaPlayer component) is hidden when instantiated and only appears when the user moves the mouse over the controller's collapsed state.

The possible values for this property are as follows:

- "on" specifies that the controls are always expanded.
- "off" specifies that the controls are always collapsed.
- "auto" (the default) specifies that the control remains in the collapsed state until the user moves the mouse over the hit area. The hit area matches the area in which the collapsed control is drawn. The control remains expanded until the mouse leaves the hit area.

NOTE

The hit area expands and contracts with the controller.

Example

The following example keeps the controller open at all times:

```
myMedia.controllerPolicy = "on";
```

Media.controlPlacement

Applies to

MediaPlayback.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.controlPlacement
```

Description

Property; determines where the controller portion of the MediaPlayback component is positioned in relation to its display. The possible values are "top", "bottom", "left", and "right". The default value is "bottom".

Example

For the following example, the controller portion of the MediaPlayback component is on the right side:

```
myMedia.controlPlacement = "right";
```

Media.cuePoint

Applies to

MediaDisplay, MediaPlayback.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();  
listenerObject.cuePoint = function(eventObj:Object) {  
    // ...  
};  
myMedia.addEventListener("cuePoint", listenerObject);
```

Description

Event; notification that the playhead has reached the cue point. The `Media.cuePoint` event object has the following properties:

`cuePointName` A string that indicates the name of the cue point.

`cuePointTime` A number, expressed in frames or seconds, that indicates when the cue point was reached.

`target` A reference to the `MediaPlayback` object if there is one, or to the `MediaDisplay` object itself.

`type` The string "cuePoint".

Example

The following example uses an object listener to determine when a cue point has been reached:

```
var myCuePointListener:Object = new Object();
myCuePointListener.cuePoint = function(eventObject:Object){
    trace("heard " + eventObject.type + ", " + eventObject.target + ", " +
        eventObject.cuePointName + ", " + eventObject.cuePointTime);
};
myPlayback.addEventListener("cuePoint", myCuePointListener);
```

See also

[Media.addCuePoint\(\)](#), [Media.cuePoints](#), [Media.getCuePoint\(\)](#)

Media.cuePoints

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`myMedia.cuePoints`

or

`myMedia.cuePoints[N]`

Description

Property; an array of cue point objects that have been assigned to a `MediaPlayback` or `MediaDisplay` instance. In the array, each cue point object can have a name, a time in seconds or frames, and a player property (which is the instance name of the component it is associated with). The default value is an empty array (`[]`).

Example

The following example deletes the third cue point if playing an action preview:

```
if (myVariable == actionPreview) {  
    myMedia.removeCuePoint(myMedia.cuePoints[2]);  
}
```

See also

[Media.addCuePoint\(\)](#), [Media.getCuePoint\(\)](#), [Media.removeCuePoint\(\)](#)

Media.displayFull()

Applies to

`MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.displayFull()
```

Returns

Nothing.

Description

Method; sets the `MediaPlayback` instance to full-screen mode. In this mode, the component expands to fill the entire Stage. To return the component to its normal size, use `Media.displayNormal()`.

Example

The following code forces the component to expand to fit the Stage:

```
myMedia.displayFull();
```

See also

[Media.displayNormal\(\)](#)

Media.displayNormal()

Applies to

MediaPlayback.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.displayNormal();
```

Returns

Nothing.

Description

Method; sets the MediaPlayback instance back to its normal size after a `Media.displayFull()` method has been used.

Example

The following code returns a MediaPlayback component to its original size:

```
myMedia.displayNormal();
```

See also

[Media.displayFull\(\)](#)

Media.getCuePoint()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.getCuePoint(cuePointName)
```

Parameters

cuePointName The string that was provided when `Media.addCuePoint()` was used.

Returns

A cue point object.

Description

Method; returns a cue point object based on its cue point name.

Example

The following code retrieves a cue point named `myCuePointName`.

```
myMedia.removeCuePoint(myMedia.getCuePoint("myCuePointName"));
```

See also

[Media.addCuePoint\(\)](#), [Media.cuePoint](#), [Media.cuePoints](#), [Media.removeCuePoint\(\)](#)

Media.horizontal

Applies to

MediaController.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.horizontal
```

Description

Property; determines whether the `MediaController` component displays itself in a vertical or horizontal orientation. A `true` value indicates that the component is displayed in a horizontal orientation; a `false` value indicates a vertical orientation. When set to `false`, the playbar and playback slider move from bottom to top. The default value is `true`.

Example

The following example displays the `MediaController` component in a vertical orientation:

```
myMedia.horizontal = false;
```

Media.mediaType

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.mediaType
```

Description

Property; indicates the type of media (FLV or MP3) to be played. The default value is "FLV". See "Working with Video" in *Using Flash*.

Example

The following example determines the current media type being played:

```
var currentMedia:String = myMedia.mediaType;
```

See also

[Media.setMedia\(\)](#)

Media.pause()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.pause()
```

Returns

Nothing.

Description

Method; pauses the playhead at the current location.

Example

The following code pauses the playback.

```
myMedia.pause();
```

Media.play()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.play(startingPoint)
```

Parameters

startingPoint A non-negative integer that indicates the starting point (in seconds) at which the media should begin playing.

Returns

Nothing.

Description

Method; plays the media associated with the component instance at the given starting point. The default value is the current value of `playheadTime`.

Example

The following code indicates that the media component should start playing at 120 seconds:

```
myMedia.play(120);
```

See also

[Media.pause\(\)](#)

Media.playheadChange

Applies to

MediaController, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.playheadChange = function(eventObject){
    // Insert your code here.
}
myMedia.addEventListener("playheadChange", listenerObject)
```

Description

Event; broadcast by the `MediaController` or `MediaPlayback` component when the user moves the playback slider or clicks the Go to Beginning or Go to End button. The `Media.playheadChange` event object has the following properties:

`detail` A number that indicates the percentage of the media that has played.

`type` The string "playheadChange".

Example

The following example sends the percentage played to the Output panel when the user stops dragging the playhead:

```
var controlListen:Object = new Object();
controlListen.playheadChange = function(eventObj:Object) {
    trace(eventObj.detail);
};
myMedia.addEventListener("playheadChange", controlListen);
```

Media.playheadTime

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`myMedia.playheadTime`

Description

Property; holds the current position of the playhead (in seconds) for the media timeline that is playing. The default value is the location of the playhead.

Example

The following example sets a variable to the location of the playhead, which is indicated in seconds:

```
var myPlayhead:Number = myMedia.playheadTime;
```

Media.playing

Applies to

MediaDisplay, MediaPlayer, MediaController.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.playing

Description

Property; returns a Boolean value that indicates whether the media is playing (*true*) or paused (*false*). This property is read-only for the MediaDisplay and MediaPlayer components, and read/write for the MediaController component.

Example

The following code determines if the media is playing or paused:

```
if(myMedia.playing == true){
    some function;
}
```

See also

[Media.change](#)

Media.preferredHeight

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.preferredHeight

Description

Property; set according to a FLV file's default height value. This property applies only to FLV media, because the height is fixed for MP3 files. This property can be used to set the height and width properties (plus some margin for the component itself). The default value is `undefined` if no FLV media is set.

Example

The following example sizes a `MediaPlayer` instance according to the media it is playing and accounts for the pixel margin needed for the component instance:

```
if (myPlayback.contentPath != undefined) {
    var mediaHeight:Number = myPlayback.preferredHeight;
    var mediaWidth:Number = myPlayback.preferredWidth;
    myPlayback.setSize((mediaWidth + 20), (mediaHeight + 70));
}
```

Media.preferredWidth

Applies to

`MediaDisplay`, `MediaPlayer`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

myMedia.preferredWidth

Description

Property; set according to a FLV file's default width value. The default value is `undefined`.

Example

The following example sets the desired width of the variable `mediaWidth`:

```
var mediaWidth:Number = myMedia.preferredWidth;
```

Media.progress

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
    // ...
};
myMedia.addEventListener("progress", listenerObject);
```

Description

Event: is generated continuously until media has completely downloaded. The `Media.progress` event object has the following properties:

target A reference to the `MediaDisplay` or `MediaPlayer` instance.

type The string "progress".

Example

The following example listens for progress:

```
var myProgressListener:Object = new Object();
myProgressListener.progress = function(eventObj:Object) {
    // Make lightMovieClip blink while progress is occurring.
    var lightVisible:Boolean = lightMovieClip.visible;
    lightMovieClip.visible = !lightVisible;
};
```

The following example listens for progress and calls another function if the progress event continues for more than 3000 milliseconds (3 seconds):

```
// Duration of delay before calling timeout.
var timeout:Number = 3000;

// If timeout has been reached, do this:
function callback(arg) {
    trace(arg);
}

// Listen for progress.
var myListener:Object = new Object();
myListener.progress = function(eventObj:Object) {
    setInterval(callback, timeout, "Experiencing Network Delay");
};
md.addEventListener("progress", myListener);
```

Media.scrubbing

Applies to

MediaController, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
    // ...
};
myMedia.addEventListener("scrubbing", listenerObject);
```

Description

Event; generated when the playhead is dragged.

target A reference to the MediaController or MediaPlayer instance.

type The string "scrubbing".

Example

The following example listens for the user to drag the playhead:

```
my_mp.addEventListener("scrubbing", scrubbingListener);
function scrubbingListener(evt_obj:Object):Void {
    trace(evt_obj.type+" @ "+getTimer()+" ms
        (isScrubbing="+evt_obj.detail+"));
}
```

Media.removeAllCuePoints()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.removeAllCuePoints()
```

Returns

Nothing.

Description

Method; deletes all cue point objects associated with a component instance.

Example

The following code deletes all cue point objects:

```
myMedia.removeAllCuePoints();
```

See also

[Media.addCuePoint\(\)](#), [Media.cuePoints](#), [Media.removeCuePoint\(\)](#)

Media.removeCuePoint()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.removeCuePoint(cuePoint)
```

Parameters

cuePoint A reference to a cue point object that has been assigned previously by means of `Media.addCuePoint()`.

Returns

Nothing.

Description

Method; deletes a cue point associated with a component instance.

Example

The following code deletes a cue point named `myCuePoint`:

```
myMedia.removeCuePoint(getCuePoint("myCuePoint"));
```

See also

[Media.addCuePoint\(\)](#), [Media.cuePoints](#), [Media.removeAllCuePoints\(\)](#)

Media.setMedia()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.setMedia(contentPath [, mediaType])
```

Parameters

contentPath A string that indicates the URL of the media to be played. The default value is undefined.

mediaType A string used to set the media type to either FLV or MP3. This parameter is optional. The default value is FLV.

Returns

Nothing.

Description

Method; sets the media type and path to the specified media type using a URL parameter.

This method provides the recommended way of setting the content path and media type for the MediaPlayer and MediaDisplay components. The `Media.contentPath` property can also be used to set the content path, but does not allow you to set the media type.

If you are working only with FLV files, you do not need to specify a *mediaType* parameter. If you are working exclusively with MP3 files, you must set the *mediaType* parameter to MP3 once. If you are switching back and forth between FLV and MP3 files, you must change the media type each time in your `setMedia()` call. If you attempt to play an MP3 file without explicitly setting the media type to MP3, the file does not play.

Example

The following code provides new media for a component instance to play:

```
myMedia.setMedia("http://www.helpexamples.com/flash/video/clouds.flv",  
    "FLV");
```

Media.stop()

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.stop()
```

Returns

Nothing.

Description

Method; stops the playhead and moves it to position 0, which is the beginning of the media.

Example

The following code stops the playhead and moves it to position 0:

```
myMedia.stop()
```

Media.totalTime

Applies to

MediaDisplay, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.totalTime
```

Description

Property; the total length of the media, in seconds. Since the FLV file format does not provide its play time to a media component until it is completely loaded, you must input `Media.totalTime` manually so that the playbar can accurately reflect the actual play time of the media. The default value for MP3 files is the play time of the media. For FLV files, the default value is `undefined`.

You cannot set this property for MP3 files, because the information is contained in the `Sound` object.

Example

The following example sets the play time (in seconds) for the FLV media:

```
myMedia.totalTime = 151;
```

Media.volume

Applies to

`MediaDisplay`, `MediaPlayback`.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myMedia.volume
```

Description

Property; stores an integer that indicates the volume setting, which can range from 0 to 100. The default value is 75.

Example

The following example sets the maximum volume for media playback:

```
myMedia.volume = 100;
```

See also

[Media.volume](#), [Media.pause\(\)](#)

Media.volume

Applies to

MediaController, MediaPlayer.

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.volume = function(eventObj:Object) {
    // ...
};
myMedia.addEventListener("volume", listenerObject);
```

Description

Event; broadcast when the volume value is adjusted by the user. The `Media.volume` event object has the following properties:

`detail` An integer between 0 and 100 that represents the volume level.

`type` The string "volume".

Example

The following example informs the user that the volume is being adjusted:

```
var myVolListener:Object = new Object();
myVolListener.volume = function(eventObj:Object) {
    mytextfield.text = "Volume adjusted!";
};
myMedia.addEventListener("volume", myVolListener);
```

See also

[Media.volume](#)

Menu component (Flash Professional only)

The Menu component lets a user select an item from a pop-up menu, much like the File or Edit menu of most software applications.

A Menu component usually opens in an application when a user rolls over or clicks a button-like menu activator. You can also script a Menu component to open when a user presses a certain key.

Menu components are always created dynamically at runtime. You drag the component from the Components panel to the library, and then use the following code to create a menu with `ActionScript`:

```
var myMenu = mx.controls.Menu.createMenu(parent, menuDataProvider);
```

Use the following code to open a menu in an application:

```
myMenu.show(x, y);
```

A `menuShow` event is broadcast to all of the Menu instance's listeners immediately before the menu is rendered, so you can update the state of the menu items. Similarly, immediately after a Menu instance is hidden, a `menuHide` event is broadcast.

The items in a menu are described by XML. For more information, see [“Understanding the Menu component: view and data” on page 886](#).

You cannot make the Menu component accessible to screen readers.

Menus are often nested within menu bars. For information about menu bars, see [“MenuBar component \(Flash Professional only\)” on page 945](#).

Interacting with the Menu component (Flash Professional only)

You can use the mouse and keyboard to interact with a Menu component.

After a Menu component is opened, it remains visible until it is closed by a script or until the user clicks the mouse outside the menu or inside an enabled item.

Clicking selects a menu item, except with the following types of menu items:

Disabled items or separators Rollovers and clicks have no effect (the menu remains visible).

anchors for a submenu Rollovers activate the submenu; clicks have no effect; rolling onto any item other than those of the submenu closes the submenu.

When an item is selected, a `Menu.change` event is sent to all of the menu's listeners, the menu is hidden, and the following actions occur, depending on item type:

check The item's `selected` attribute is toggled.

radio The item becomes the current selection of its radio group.

Moving the mouse triggers `Menu.rollOut` and `Menu.rollOver` events.

Pressing the mouse outside the menu closes the menu and triggers a `Menu.menuHide` event.

Releasing the mouse in an enabled item affects item types in the following ways:

check The item's `selected` attribute is toggled.

radio The item's `selected` attribute is set to `true`, and the previously selected item's `selected` attribute in the radio group is set to `false`. The `selection` property of the corresponding radio group object is set to refer to the selected menu item.

undefined and the parent of a hierarchical menu The visibility of the hierarchical menu is toggled.

When a Menu instance has focus either from clicking or tabbing, you can use the following keys to control it:

Key	Description
Down Arrow Up Arrow	Moves the selection down and up the rows of the menu. The selection cycles at the top or bottom row.
Right Arrow	Opens a submenu, or moves selection to the next menu in a menu bar (if a menu bar exists).
Left Arrow	Closes a submenu and returns focus to the parent menu (if a parent menu exists), or moves selection to the previous menu in a menu bar (if the menu bar exists).
Enter	Opens a submenu. If a submenu does not exist, this key has the same effect as clicking and releasing on a row.

NOTE

If a menu is opened, you can press the Tab key to move out of the menu. You must either make a selection or dismiss the menu by pressing Escape.

Using the Menu component (Flash Professional only)

You can use the Menu component to create a menu of selectable choices; this menu is like the File or Edit menu of most software applications. You can also use the Menu component to create context-sensitive menus that appear when a user clicks a hot spot or a presses a modifier key. Use the Menu component with the MenuBar component to create a horizontal menu bar with menus that extend under each menu bar item.

Like standard desktop menus, the Menu component supports menu items whose functions fall into the following general categories:

Command activators These items trigger events; you write code to handle those events.

Submenu anchors These items are anchors that open submenus.

Radio buttons These items operate in groups; you can select only one item at a time.

Check box items These items represent a Boolean (`true` or `false`) value.

Separators These items provide a simple horizontal line that divides the items in a menu into different visual groups.

Understanding the Menu component: view and data

Conceptually, the Menu component consists of a data model and a view that displays the data. The Menu class provides the view and contains the visual configuration methods. The [MenuDataProvider class](#) adds methods to the global XML prototype object (much like the DataProvider API does to the Array object); these methods let you externally construct data providers and add them to multiple menu instances. The data provider broadcasts any changes to all of its client views. (See [“MenuDataProvider class” on page 933.](#))

A Menu instance is a hierarchical collection of XML elements that correspond to individual menu items. The attributes define the behavior and appearance of the corresponding menu item on the screen. The collection is easily translated to and from XML, which is used to describe menus (the `menu` tag) and items (the `menuitem` tag). The built-in ActionScript XML class is the basis for the model underlying the Menu component.

A simple menu with two items can be described in XML with two menu item subelements:

```
<menu>
  <menuitem label="Up" />
  <menuitem label="Down" />
</menu>
```

NOTE

The tag names of the XML nodes (`menu` and `menuitem`) are not important; the attributes and their nesting relationships are used in the menu.

About hierarchical menus

To create hierarchical menus, embed XML elements within a parent XML element, as follows:

```
<menu>
  <menuitem label="MenuItem A" >
    <menuitem label="SubMenuItem 1-A" />
    <menuitem label="SubMenuItem 2-A" />
  </menuitem>
  <menuitem label="MenuItem B" >
    <menuitem label="SubMenuItem 1-B" />
    <menuitem label="SubMenuItem 2-B" />
  </menuitem>
</menu>
```

This converts the parent menu item into a pop-up menu anchor, so it does not generate events when selected.

About menu item XML attributes

The attributes of a menu item XML element determine what is displayed, how the menu item behaves, and how it is exposed to ActionScript. The following table describes the attributes of an XML menu item:

Attribute name	Type	Default	Description
label	String	undefined	The text that is displayed to represent a menu item. This attribute is required for all item types, except <code>separator</code> .
type	separator, check, radio, normal, or undefined	undefined	The type of menu item: <code>separator</code> , <code>check box</code> , <code>radio button</code> , or <code>normal</code> (a command or submenu activator). If this attribute does not exist, the default value is <code>normal</code> .
icon	String	undefined	The linkage identifier of an image asset. This attribute is not required and is not available for the <code>check</code> , <code>radio</code> , or <code>separator</code> type.
instanceName	String	undefined	An identifier that you can use to reference the menu item instance from the root menu instance. For example, a menu item named <i>yellow</i> can be referenced as <code>myMenu.yellow</code> . This attribute is not required.
groupName	String	undefined	An identifier that you can use to associate several radio button items in a radio group, and to expose the state of a radio group from the root menu instance. For example, a radio group named <i>colors</i> can be referenced as <code>myMenu.colors</code> . This attribute is required only for the type <code>radio</code> .
selected	A Boolean value (false or true) or string ("false" or "true")	false	A Boolean or string value indicating whether a <code>check</code> or <code>radio</code> item is on (<code>true</code>) or off (<code>false</code>). This attribute is not required.
enabled	A Boolean value (false or true) or string ("false" or "true")	true	A Boolean or string value indicating whether this menu item can be selected (<code>true</code>) or not (<code>false</code>). This attribute is not required.

About menu item types (Flash Professional only)

There are four kinds of menu items, specified by the `type` attribute:

```
<menu>
  <menuitem label="Normal Item" />
  <menuitem type="separator" />
  <menuitem label="Checkbox Item" type="check" instanceName="check_1"/>
  <menuitem label="RadioButton Item" type="radio"
    groupName="radioGroup_1" />
</menu>
```

Normal menu items

The `Normal Item` menu item doesn't have a `type` attribute, which means that the `type` attribute defaults to `normal`. Normal items can be command activators or submenu activators, depending on whether they have nested subitems.

Separator menu items

A menu item whose `type` attribute is set to `separator` acts as a visual divider in a menu. The following XML creates three menu items, `Top`, `Middle`, and `Bottom`, with separators between them:

```
<menu>
  <menuitem label="Top" />
  <menuitem type="separator" />
  <menuitem label="Middle" />
  <menuitem type="separator" />
  <menuitem label="Bottom" />
</menu>
```

All separator items are disabled. Clicking on or rolling over a separator has no effect.

Check box menu items

A menu item whose `type` attribute is set to `check` acts as check box item in the menu; when the `selected` attribute is set to `true`, a check mark appears beside the menu item's label. When a check box item is selected, its state automatically toggles, and a `change` event is broadcast to all listeners on the root menu. However, although a check box menu item behaves similarly to a `CheckBox` component, a check box menu item appears visually without the box surrounding the check. So an unselected check box menu item looks like a normal menu item until selected.

The following example defines three check box menu items:

```
<menu>
  <menuItem label="Apples" type="check" instanceName="buyApples"
    selected="true" />
  <menuItem label="Oranges" type="check" instanceName="buyOranges"
    selected="false" />
  <menuItem label="Bananas" type="check" instanceName="buyBananas"
    selected="false" />
</menu>
```

You can use the instance names in ActionScript to access the menu items directly from the menu itself, as in the following example:

```
myMenu.setMenuItemSelected(myMenu.buyapples, true);
myMenu.setMenuItemSelected(myMenu.buyoranges, false);
```

NOTE

The `selected` attribute should be modified only with the `setMenuItemSelected()` method. You can directly examine the `selected` attribute, but it returns a string value of `true` or `false`.

Radio button menu items

Menu items whose `type` attribute is set to `radio` can be grouped together so that only one of the items can be selected at a time. Although a radio button menu item behaves similarly to a `RadioButton` component, a radio button menu item appears visually without the border surrounding the button. So an unselected radio button menu item looks like a Normal menu item until selected.

You create a radio group by giving the menu items the same value for their `groupName` attribute, as in the following example:

```
<menu>
  <menuItem label="Center" type="radio" groupName="alignment_group"
    instanceName="center_item"/>
  <menuItem type="separator" />
  <menuItem label="Top" type="radio" groupName="alignment_group" />
  <menuItem label="Bottom" type="radio" groupName="alignment_group" />
  <menuItem label="Right" type="radio" groupName="alignment_group" />
  <menuItem label="Left" type="radio" groupName="alignment_group" />
</menu>
```

When the user selects one of the items, the current selection automatically changes, and a `change` event is broadcast to all listeners on the root menu. The currently selected item in a radio group is available in ActionScript through the `selection` property, as follows:

```
var selectedItem = myMenu.alignment_group.selection;
myMenu.alignment_group = myMenu.center_item;
```

Each `groupName` value must be unique within the scope of the root menu instance.

NOTE

The `selected` attribute should be modified only with the `setMenuItemSelected()` method. You can directly examine the `selected` attribute, but it returns a string value of `true` or `false`.

Exposing menu items to ActionScript

You can assign each menu item a unique identifier in the `instanceName` attribute, which makes the menu item accessible directly from the root menu. For example, the following XML code provides `instanceName` attributes for each menu item:

```
<menu>
  <menuItem label="Item 1" instanceName="item_1" />
  <menuItem label="Item 2" instanceName="item_2" >
    <menuItem label="SubItem A" instanceName="sub_item_A" />
    <menuItem label="SubItem B" instanceName="sub_item_B" />
  </menuItem>
</menu>
```

You can use ActionScript to access the corresponding instances and their attributes directly from the menu component, as follows:

```
var aMenuItem = myMenu.item_1;
myMenu.setMenuItemEnabled(item_2, true);
var aLabel = myMenu.sub_item_A.attributes.label;
```

NOTE

Each `instanceName` attribute must be unique within the scope of the root menu component instance (including all of the submenus of root).

About initialization object properties (Flash Professional only)

The *initObject* (initialization object) parameter is a fundamental concept in creating the layout for the Menu component. This parameter is an object with properties. Each property represents one of the possible the XML attributes of a menu item. (For a description of the properties allowed in the *initObject* parameter, see [“About menu item XML attributes” on page 887.](#))

The *initObject* parameter is used in the following methods:

- `Menu.addItem()`
- `Menu.addItemAt()`
- `MenuDataProvider.addItem()`
- `MenuDataProvider.addItemAt()`

The following example creates an *initObject* parameter with two properties, `label` and `instanceName`:

```
var i = myMenu.addItem({label:"myMenuItem",  
    instanceName:"myFirstItem"});
```

Several of the properties work together to create a particular type of menu item. You assign specific properties to create certain types of menu items (normal, separator, check box, or radio button).

For example, you can initialize a normal menu item with the following *initObject* parameter:

```
myMenu.addItem({label:"myMenuItem", enabled:true, icon:"myIcon",  
    instanceName:"myFirstItem"});
```

You can initialize a separator menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"separator"});
```

You can initialize a check box menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"check", label:"myMenuCheck", enabled:false,  
    selected:true, instanceName:"myFirstCheckItem"});
```

You can initialize a radio button menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"radio", label:"myMenuRadio1", enabled:true,  
    selected:false, groupName:"myRadioGroup",  
    instanceName:"myFirstRadioItem"});
```

You should treat the `instanceName`, `groupName`, and `type` attributes of a menu item as read-only. You should set them only while creating an item (for example, in a call to `addItem()`). Modifying these attributes after creation may produce unpredictable results.

Menu parameters (Flash Professional only)

You can set the following authoring parameter for each Menu component instance in the Property inspector:

rowHeight indicates the height of each row, in pixels. Changing the font size does not change the row height. The default value is 20.

You can write ActionScript to control the Menu component using its properties, methods, and events. For more information, see [“Menu class \(Flash Professional only\)” on page 901](#).

Creating an application with the Menu component (Flash Professional only)

In the following example, a developer is building an application and uses the Menu component to expose some of the commands that users can issue, such as Open, Close, and Save.

To create an application with the Menu component:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the library.
Menus are created dynamically through ActionScript.
3. Drag a Button component from the Components panel to the library.
The button will be used to activate the menu.

4. In the Actions panel, on the first frame, enter the following code to add an event listener to listen for `click` events on the button. The code also listens for a `change` event on the menu and displays the name of the selected menu item in the Output panel:

```
/**
 * Requires:
 * - Menu component in library
 * - Button component in library
 */

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "menu_button", 10, {label:"Launch
  Menu"});

// Create a menu.
var my_menu:Menu = Menu.createMenu();

// Add some menu items.
my_menu.addItem("Open");
my_menu.addItem("Close");
my_menu.addItem("Save");
my_menu.addItem("Revert");

// Add a change-listener to Menu to detect which menu item is selected.
var menuListener:Object = new Object();
menuListener.change = function(evt_obj:Object) {
    var item_obj:Object = evt_obj.menuItem;
    trace("Item selected: "+item_obj.attributes.label);
};
my_menu.addEventListener("change", menuListener);

// Add a button listener that displays the menu when the button is
// clicked.
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    var my_button:Button = evt_obj.target;
    // Display the menu at the bottom of the button.
    my_menu.show(my_button.x, my_button.y + my_button.height);
};
menu_button.addEventListener("click", buttonListener);
```

5. Select Control > Test Movie.

Click the Launch Menu button to display the menu. When you select a menu item, a `trace()` statement reports the selection in the Output panel.

To use XML data from a server to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the library.
Menus are created dynamically through ActionScript.
3. In the Actions panel, add the following code to the first frame to create a menu, and use the `dataProvider` property to load menu items from a web page:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

var my_menu:Menu = Menu.createMenu();

// Import an XML file.
var myDP_xml:XML = new XML();
myDP_xml.ignoreWhite = true;
myDP_xml.onLoad = function(success:Boolean) {
    // When the data arrives, pass it to the menu.
    if (success) {
        my_menu.dataProvider = myDP_xml.firstChild;
    }
};
myDP_xml.load("http://www.flash-mx.com/mm/xml/menu.xml");

// Show and position the menus.
my_menu.show(100, 20);
```

NOTE

The menu items are described by the children of the XML document's first child.

4. Select Control > Test Movie.

The xml menu definition from the web page is provided here for your reference:

```
<?xml version="1.0" ?>
<menu>
<menuitem label="Undo" />
<menuitem type="separator" />
<menuitem label="Cut" />
<menuitem label="Copy" />
<menuitem label="Paste" />
<menuitem label="Clear" />
<menuitem type="separator" />
<menuitem label="Select All" />
</menu>
```

To use a well-formed XML string to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the library.
Menus are created dynamically through ActionScript.
3. In the Actions panel, add the following code to the first frame to create a menu and add some items:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var theMenuElement_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
theMenuElement_obj.addMenuItem({label:"Undo"});
theMenuElement_obj.addMenuItem({type:"separator"});
theMenuElement_obj.addMenuItem({label:"Cut"});
theMenuElement_obj.addMenuItem({label:"Copy"});
theMenuElement_obj.addMenuItem({label:"Paste"});
theMenuElement_obj.addMenuItem({label:"Clear", enabled:"false"});
theMenuElement_obj.addMenuItem({type:"separator"});
theMenuElement_obj.addMenuItem({label:"Select All"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, theMenuElement_obj);

// Show and position the menus.
my_menu.show(100, 20);
```

4. Select Control > Test Movie.

To use the `MenuDataProvider` class to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the library.
Menus are created dynamically through ActionScript.
3. In the Actions panel, add the following code to the first frame to create a menu and add some items:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var xml = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var theMenuElement = xml.addMenuItem("XXXXX");

// Add the menu items.
theMenuElement.addMenuItem({label:"Undo"});
theMenuElement.addMenuItem({type:"separator"});
theMenuElement.addMenuItem({label:"Cut"});
theMenuElement.addMenuItem({label:"Copy"});
theMenuElement.addMenuItem({label:"Paste"});
theMenuElement.addMenuItem({label:"Clear", enabled:"false"});
theMenuElement.addMenuItem({type:"separator"});
theMenuElement.addMenuItem({label:"Select All"});
// Create the Menu object.
var my_menu = mx.controls.Menu.createMenu(_root, theMenuElement);

my_menu.show(100, 20);
```

4. Select Control > Test Movie.

Customizing the Menu component (Flash Professional only)

The menu sizes itself horizontally to fit its widest text. You can also call the `setSize()` method to size the component. Icons should be sized to a maximum of 16 by 16 pixels.

Using styles with the Menu component

You can call the `setStyle()` method to change the style of the menu, its items, and its submenus. The Menu component supports the following styles:

Style	n	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>alternatingRowColors</code>	Both	Specifies colors for rows in an alternating pattern. The value can be an array of two or more colors, for example, <code>0xFF00FF</code> , <code>0xCC6699</code> , and <code>0x996699</code> . Unlike single-value color styles, <code>alternatingRowColors</code> does not accept color names; the values must be numeric color codes. By default, this style is not set, and <code>backgroundColor</code> is used in its place for all rows.
<code>backgroundColor</code>	Both	The background color of the menu. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to "false". The default value is <code>0xDDDDDD</code> (medium gray).
<code>borderStyle</code>	Both	The Menu component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See " RectBorder class " on page 1063. The default border style is "menuBorder".
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).

Style	n	Description
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>defaultIcon</code>	Both	The name of the default icon to display on each row. The default value is <code>undefined</code> , which means no icon is displayed. The icon property is not required, does not work for <code>"check"</code> , <code>"radio"</code> , or <code>"separator"</code> items, and uses the linkage identifier of an image asset as the value parameter. All menu items show the same icon.
<code>popupDuration</code>	Both	The duration of the transition as a menu opens. The value is specified in milliseconds; 0 indicates no transition. The default value is 150.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>0xE3FFD6</code> (bright green) with the Halo theme and <code>0xA9A9A9</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.

Style	n	Description
<code>selectionColor</code>	Both	The background color of a selected row. The default value is a <code>0xCDFFC1</code> (light green) with the Halo theme and <code>0xEFFFFFFF</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal to selected state, in milliseconds. The default value is 200.
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. The default equation uses a sine in/out formula. For more information, see “Customizing component animations” in <i>Using Components</i> .
<code>textRollOverColor</code>	Both	The color of text when the pointer rolls over. The default value is <code>0x2B333C</code> (dark gray). This style is important when you set <code>rollOverColor</code> , because the two settings must complement each other so that text is easily viewable during rollover.
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two must complement each other so that text is easily viewable while selected.
<code>useRollOver</code>	Both	Determines whether rolling over a row activates highlighting. The default value is true.

Setting styles for all Menu components in a document

The Menu class inherits from the ScrollSelectList class. The default class-level style properties are defined on the ScrollSelectList class, which is shared by all List-based components. You can set new default style values on this class directly, and the new settings are reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the Menu components only, you can create a new `CSSStyleDeclaration` and store it in `_global.styles.Menu`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Menu == undefined) {
    _global.styles.Menu = new CSSStyleDeclaration();
}
_global.styles.Menu.setStyle("backgroundColor", 0xFF00AA);
```

When you create a new class-level style declaration, you lose all default values provided by the `ScrollSelectList` declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Menu;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

For more information about class-level styles see “Setting styles for a component class” in *Using Components*.

Using skins with the Menu component

The Menu component uses an instance of `RectBorder` for its border (see “[RectBorder class](#)” on page 1063).

The Menu component has visual assets for the branch, check mark, radio dot, and separator graphics. These assets are not dynamically skinnable, but the assets can be copied from the Flash UI Components 2/Themes/MMDefault/Menu Assets/States folder in both themes, and can be modified as desired. The linkage identifiers cannot be changed, and all Menu instances must use the same symbols.

To create movie clip symbols for Menu assets:

1. Create a new FLA file.
2. Select File > Import > Open External Library and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the Menu Assets folder to the library of your document.
4. Expand the Menu Assets/States folder in the library of your document.

5. Open the symbols that you want to customize for editing.
For example, open the MenuCheckEnabled symbol.
6. Customize the symbol as desired.
For example, change the image to be an X instead of a check mark.
7. Repeat steps 6-7 for all symbols that you want to customize.
8. Click the Back button to return to the main timeline.
9. Drag a Menu component from the Components panel to the current document's library.
This adds the Menu component to the library and makes it available at runtime.
10. Add ActionScript to the main timeline to create a Menu instance at runtime:

```
var myMenu = mx.controls.Menu.createMenu();
myMenu.addItem({label: "One", type: "check", selected: true});
myMenu.addItem({label: "Two", type: "check"});
myMenu.addItem({label: "Three", type: "check"});
myMenu.show(0, 0);
```
11. Select Control > Test Movie.

Menu class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > ScrollView > ScrollSelectList > Menu

ActionScript Class Name mx.controls.Menu

The methods and properties of the Menu class let you create and edit menus at runtime.

Setting a property of the menu class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Menu.version);
```

NOTE

The code `trace(myMenuInstance.version);` returns `undefined`.

Method summary for the Menu class

The following table lists methods of the Menu class.

Method	Description
<code>Menu.addItem()</code>	Adds a menu item to the menu.
<code>Menu.addItemAt()</code>	Adds a menu item to the menu at a specific location.
<code>Menu.createMenu()</code>	Creates an instance of the Menu class. This is a static method.
<code>Menu.getItemAt()</code>	Gets a reference to a menu item at a specified location.
<code>Menu.hide()</code>	Closes a menu.
<code>Menu.indexOf()</code>	Returns the index of a given menu item.
<code>Menu.removeAll()</code>	Removes all items from a menu.
<code>Menu.removeItem()</code>	Removes the specified menu item.
<code>Menu.removeItemAt()</code>	Removes a menu item from a menu at a specified location.
<code>Menu.setMenuItemEnabled()</code>	Indicates whether a menu item is enabled (<code>true</code>) or not (<code>false</code>).
<code>Menu.setMenuItemSelected()</code>	Indicates whether a menu item is selected (<code>true</code>) or not (<code>false</code>).
<code>Menu.show()</code>	Opens a menu at a specific location or at its previous location.

Methods inherited from the UIObject class

The following table lists the methods the Menu class inherits from the UIObject class. When calling these methods from the Menu object, use the form *MenuInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.

Method	Description
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Menu class inherits from the UIComponent class. When calling these methods from the Menu object, use the form

MenuInstance.methodName.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the Menu class

The following table lists the property of the Menu class.

Property	Description
Menu.dataProvider	The data source for a menu.

Properties inherited from the UIObject class

The following table lists the properties the Menu class inherits from the UIObject class.

When accessing these properties from the Menu object, use the form

MenuInstance.propertyName.

Property	Description
UIObject.bottom	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
UIObject.height	The height of the object, in pixels. Read-only.
UIObject.left	The left edge of the object, in pixels. Read-only.
UIObject.right	The position of the right edge of the object, relative to the right edge of its parent. Read-only.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Menu class inherits from the UIComponent class. When accessing these properties from the Menu object, use the form *MenuInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the Menu class

The following table lists events of the Menu class.

Event	Description
<code>Menu.change</code>	Broadcast when a user causes a change in a menu.
<code>Menu.menuHide</code>	Broadcast when a menu closes.
<code>Menu.menuShow</code>	Broadcast when a menu opens.
<code>Menu.rollOut</code>	Broadcast when the pointer rolls off an item.
<code>Menu.rollOver</code>	Broadcast when the pointer rolls over an item.

Events inherited from the UIObject class

The following table lists the events the Menu class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Menu class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Menu.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
menuInstance.addItem(initObject)
```

Usage 2:

```
menuInstance.addItem(childMenuItem)
```

Parameters

initObject An object containing properties that initialize a menu item's attributes. See [“About menu item XML attributes” on page 887](#).

childMenuItem An XML node object.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a menu item at the end of the menu. The menu item is constructed from the values supplied in the *initObject* parameter. Usage 2 adds a menu item that is a prebuilt XML node (in the form of an XML object) at the end of the menu. Adding a preexisting node removes the node from its previous location.

Example

The following example creates two menus, initially adding one menu item to each. The example then adds two more menu items to the first menu, calling `addItem()` to add the first menu item by specifying its attributes. It then adds the second menu item by using the prebuilt menu item node from the second menu.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create the Menu objects.
var first_menu:Menu = Menu.createMenu();
first_menu.addItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addItem({label:"1st Item 2nd Menu"});

// First usage method
first_menu.addItem({label:"Radio Item", instanceName:"radioItem1",
    type:"radio", selected:false, enabled:true, groupName:"myRadioGroup"});

// Second usage method
first_menu.addItem(second_menu.getMenuItemAt(0));

// Show menu.
first_menu.show();
```

Menu.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
menuInstance.addItemAt(index, initObject)
```

Usage 2:

```
menuInstance.addItemAt(index, childMenuItem)
```

Parameters

index An integer indicating the index position (among the child nodes) at which the item is added.

initObject An object containing properties that initialize a menu item's attributes. See [“About menu item XML attributes” on page 887](#).

childMenuItem An XML node object.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a menu item (child node) at the specified location in the menu. The menu item is constructed from the values supplied in the *initObject* parameter. Usage 2 adds a menu item that is a prebuilt XML node (in the form of an XML object) at a specified location in the menu. Adding a preexisting node removes the node from its previous location.

Example

The following example creates two menus, initially adding one menu item to each. The example then adds two more menu items to the first menu, calling `addItemAt()` to add a menu item in the second position by specifying its attributes. It then adds a menu item in the third position by using the prebuilt menu item node from the second menu.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create the Menu objects.
var first_menu:Menu = Menu.createMenu();
first_menu.addItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addItem({label:"1st Item 2nd Menu"});

// First usage method
first_menu.addItemAt(1, {label:"Radio Item", instanceName:"radioItem1",
    type:"radio", selected:false, enabled:true, groupName:"myRadioGroup"});

// Second usage method
first_menu.addItemAt(2, second_menu.getMenuItemAt(0));

// Show menu.
first_menu.show();
```

Menu.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // Insert your code here.
};
menuInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    // Insert your code here.
}
```


Description

Event; broadcast to all registered listeners whenever a user causes a change in the menu.

Version 2 Macromedia Component Architecture components use a dispatcher-listener event model. When a Menu component broadcasts a `change` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.change` event's event object has the following additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is `undefined`.
- `menu` A reference to the `Menu` instance where the target item is located.
- `menuItem` An XML node that is the menu item that was selected.
- `groupName` A string indicating the name of the radio button group to which the item belongs. If the item is not in a radio button group, this value is `undefined`.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a menu, `my_menu`, and defines an event listener for it, `menulistener`, which listens for a `change` event. When a user causes a change event by clicking a menu item, the example displays its label attribute in the Output panel.

You first drag a `Menu` component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("Edit");
```

```
// Add the menu items.
menuDP_obj.addItem({label:"1st Item"});
menuDP_obj.addItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);
my_menu.show();

var menuListener:Object = new Object();
menuListener.change = function(evt_obj:Object) {
    trace("Menu item chosen: " + evt_obj.menuItem.attributes.label);
};
my_menu.addEventListener("change", menuListener);
```

Menu.createMenu()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Menu.createMenu([parent [, mdp]])
```

Parameters

parent A MovieClip instance. The movie clip is the parent component that contains the new Menu instance. This parameter is optional.

mdp The MenuDataProvider instance that describes this Menu instance. This parameter is optional.

Returns

A reference to the new menu instance.

Description

Method (static); instantiates a Menu instance, and optionally attaches it to the specified parent, with the specified MenuDataProvider as the data source for the menu items.

If the *parent* parameter is omitted or null, the Menu is attached to the `_root` timeline.

If the *mdp* parameter is omitted or null, the menu does not have any items; you must call `addItem()` or `setDataProvider()` to populate the menu.

Example

The following example creates a menu with a submenu for the New menu item. It creates the menu by creating an XML object, `my_xml`, and adding menu items to it with calls to `addItem()`. It then creates the menu with a call to `createMenu()`, passing the XML object as the data provider.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

var my_xml:XML = new XML();
var newItem_obj:Object = my_xml.addItem({label:"New"});

// Create other submenu items for main menu.
my_xml.addItem({label:"Open", instanceName:"miOpen"});
my_xml.addItem({label:"Save", instanceName:"miSave"});
my_xml.addItem({type:"separator"});
my_xml.addItem({label:"Quit", instanceName:"miQuit"});

// Create submenu items for "New" submenu.
newItem_obj.addItem({label:"File..."});
newItem_obj.addItem({label:"Project..."});
newItem_obj.addItem({label:"Resource..."});

// Create menu.
var my_menu:Menu = Menu.createMenu(myParent_mc, my_xml);
my_menu.show(100, 20);
```

Menu.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

menuInstance.dataProvider

Description

Property; the data source for items in a Menu component.

`Menu.dataProvider` is an XML node object. Setting this property replaces the existing data source of the menu.

The default value is `undefined`.

NOTE

All XML or XMLNode instances are automatically given the methods and properties of the `MenuDataProvider` class when they are used with the Menu component.

Example

The following example creates a menu (`my_menu`), loads menu items from a web page into an XML object, and then populates the menu with menu items by assigning child nodes of the XML object to the `dataProvider` property of the menu (`my_menu.dataProvider`).

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create the Menu object.
var my_menu:Menu = Menu.createMenu();

var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean){
    if (success) {
        my_menu.dataProvider = my_xml.firstChild;
    }
}
my_xml.load("http://www.flash-mx.com/mm/xml/menu.xml");

// Show and position the menus.
my_menu.show(100, 20);
```

Menu.getItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.getItemAt(index)
```

Parameters

index An integer indicating the index of the node in the menu.

Returns

A reference to the specified node.

Description

Method; returns a reference to the specified child node of the menu.

Example

The following example initially creates two menus with a single menu item for each one. It then adds a second menu item to the first menu by calling the `getItemAt()` method to obtain the menu item from the second menu and add it to the first menu.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create the Menu objects.
var first_menu:Menu = Menu.createMenu();
first_menu.addItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addItem({label:"1st Item 2nd Menu"});

// Add item from second_menu to 2nd position on first menu.
first_menu.addItemAt(1, second_menu.getItemAt(0));

// Show menu.
first_menu.show();
```

Menu.hide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

menuInstance.hide()

Returns

Nothing.

Description

Method; closes a menu.

Example

The following example creates a button and a two-item menu and displays the menu for an interval of 2000 milliseconds. When the interval expires, the `closeMenu()` function calls the `menu.hide()` method to close the menu. Clicking the Reset Menu button triggers the `resetMenu()` listener, which redisplay the menu and resets the interval.

You first drag a Menu component and a Button component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 *   - Button component in library
 */

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10, {label:"Reset Menu",
    _x:100, _y:50});

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
```

```

var menuDP_obj:Object = my_xml.addItem("Edit");

// Add the menu items.
menuDP_obj.addItem({label:"1st Item"});
menuDP_obj.addItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

my_menu.show(100, 100);
// Call closeMenu after 2000 milliseconds.
var interval_id:Number = setInterval(closeMenu, 2000, my_menu);
function closeMenu(the_menu:Menu):Void {
    the_menu.hide();
    clearInterval(interval_id);
}
// Listener for button click; show menu and reset interval.
function resetMenu(evt_obj:Object):Void {
    clearInterval(interval_id);
    my_menu.show(100, 100);
    interval_id = setInterval(closeMenu, 2000, my_menu);
}
my_button.addEventListener("click", resetMenu);

```

See also

[Menu.show\(\)](#)

Menu.indexOf()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

menuInstance.indexOf(*item*)

Parameters

item A reference to an XML node that describes a menu item.

Returns

The index of the specified menu item, or undefined if the item does not belong to this menu.

Description

Method; returns the index of the specified menu item within this menu instance.

Example

The following example creates a menu with two items from an XML data provider and then adds a third item to the menu and saves the reference that is returned by the `addItem()` method. Next, it calls the `indexOf()` method by using the reference to obtain the index of the item and display it in the Output panel.

You first drag a Menu component and a Button component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addItem("Edit");

// Add the menu items.
menuDP_obj.addItem({label:"1st Item"});
menuDP_obj.addItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);
var myItem_obj:Object = my_menu.addItem({label:"That item"});

// Show and position the menus.
my_menu.show(100, 20);

var myIndex_num:Number = my_menu.indexOf(myItem_obj);
trace("Index of 'That Item': " + myIndex_num);
```


Menu.menuHide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.menuHide = function(eventObject:Object) {
    // Insert your code here.
};
menuInstance.addEventListener("menuHide", listenerObject);
```

Usage 2:

```
on (menuHide) {
    // Insert your code here.
}
```

Description

Event; broadcast to all registered listeners whenever a menu closes.

Version 2 components use a dispatcher-listener event model. When a Menu component dispatches a `menuHide` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler and the name of the listener object as parameters.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.menuHide` event's event object has two additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is `undefined`.
- `menu` A reference to the `Menu` instance that is hidden.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a button and a two-item menu. When the user clicks the button, a listener for a button click event displays the menu. When the user clicks a second time, the menu is hidden and a listener for the `menuHide` event, `menuListener`, displays “Menu closed” in the Output panel.

You first drag a Menu component and a Button component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 *   - Button component in library
 */

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10);

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Add a button that displays the menu when the button is clicked.
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    // get reference to the button
    var the_button:Button = evt_obj.target;
    // Display the menu at the bottom of the button.
    my_menu.show(the_button.x, the_button.y + the_button.height);
};
my_button.addEventListener("click", buttonListener);

// Create listener object.
var menuListener:Object = new Object();
```

```
menuListener.menuHide = function(evt_obj:Object) {
    trace("Menu closed.");
};

// Add listener.
my_menu.addEventListener("menuHide", menuListener);
```

See also

[Menu.menuShow](#)

Menu.menuShow

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.menuShow = function(eventObject:Object) {
    // Insert your code here.
};
menuInstance.addEventListener("menuShow", listenerObject);
```

Usage 2:

```
on (menuShow) {
    // Insert your code here.
}
```

Description

Event; broadcast to all registered listeners whenever a menu opens. All parent nodes open menus to show their children.

Version 2 components use a dispatcher-listener event model. When a Menu component dispatches a `menuShow` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler and listener object as parameters.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.menuShow` event's event object has two additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is undefined.
- `menu` A reference to the `Menu` instance that is shown.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a button and a two-item menu. When the user clicks the button, a listener for a button click event displays the menu. A listener for the `menuShow` event, `menuListener`, displays “Menu open” in the Output panel.

You first drag a `Menu` component and a `Button` component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 *   - Button component in library
 */

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10);

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("Edit");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Add a button that displays the menu when the button is clicked.
var buttonListener:Object = new Object();
```

```
buttonListener.click = function(evt_obj:Object) {
    // get reference to the button
    var the_button:Button = evt_obj.target;
    // Display the menu at the bottom of the button.
    my_menu.show(the_button.x, the_button.y + the_button.height);
};
my_button.addEventListener("click", buttonListener);

// Create listener object.
var menuListener:Object = new Object();
menuListener.menuShow = function(evt_obj:Object) {
    trace("Menu open.");
};

// Add listener.
my_menu.addEventListener("menuShow", menuListener);
```

See also

[Menu.menuHide](#)

Menu.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.removeAll()
```

Returns

Nothing.

Description

Method; removes all items and refreshes the menu.

Example

The following example creates a menu with two items and, after an interval of a couple of seconds 2000 milliseconds), removes all nodes from the menu.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);
var interval_id:Number = setInterval(remove, 2000, my_menu);
function remove(the_menu:Menu):Void {
    // Delete all menu items.
    the_menu.removeAll();
    clearInterval(interval_id);
    the_menu.show(100, 20);
}
```

Menu.removeItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.removeMenuItem()
```

Returns

A reference to the returned menu item (XML node). This value is `undefined` if there is no item in that position.

Description

Method; removes the specified menu item and all its children, and refreshes the menu.

Example

The following example creates a menu with three menu items and sets an interval to cause the menu to be displayed for a couple of seconds (2000 milliseconds). When the interval expires, the example calls the `removeItem()` function, which calls the `removeMenuItem()` method to remove the first item in the menu and redisplay it.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"first Item"});
menuDP_obj.addMenuItem({label:"second Item"});
menuDP_obj.addMenuItem({label:"third Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menu.
my_menu.show(100, 20);

// Call closeMenu after 2000 milliseconds.
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
    // Delete the first node item.
    var myItem_obj:Object = my_menu.getMenuItemAt(0);
```

```
myItem_obj.removeMenuItem();
clearInterval(interval_id);
my_menu.show(100, 20);
}
```

Menu.removeItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.removeItemAt(index)
```

Parameters

index The index of the menu item to remove.

Returns

A reference to the returned menu item (XML node). This value is undefined if there is no item in that position.

Description

Method; removes the menu item and all its children at the specified index. If there is no menu item at that index, calling this method has no effect.

Example

The following example creates a menu with two items and, after an interval of a couple of seconds (2000 milliseconds), removes the second item (at index 1).

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
```



```

// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);
var interval_id:Number = setInterval(remove, 2000, my_menu);
function remove(the_menu:Menu):Void {
    // Delete the 2nd node item.
    var item_obj:Object = my_menu.removeItemAt(1);
    trace("Item removed: " + item_obj);
    clearInterval(interval_id);
    the_menu.show(100, 20);
}

```

Menu.rollOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```

var listenerObject:Object = new Object();
listenerObject.rollOut = function(eventObject:Object) {
    // Insert your code here.
};
menuInstance.addEventListener("rollOut", listenerObject);

```

Usage 2:

```

on (rollOut) {
    // Insert your code here.
}

```

Description

Event; broadcast to all registered listeners when the pointer rolls off a menu item.

Version 2 components use a dispatcher-listener event model. When a Menu component broadcasts a `rollOut` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.rollOut` event's event object has one additional property: `menuItem`, which is a reference to the menu item (XML node) that the pointer rolled off.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a menu with two items and a listener for a `rollOut` event. When the `rollOut` event is broadcast, a `trace()` function in the event handler, `menuListener`, displays the name of the menu item for which the event occurred.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
```

```
my_menu.show(100, 20);

// Create listener object.
var menuListener:Object = new Object();
menuListener.rollOut = function(evt_obj:Object) {
    trace("Menu rollOut: " + evt_obj.menuItem.attributes.label);
};

// Add listener.
my_menu.addEventListener("rollOut", menuListener);
```

Menu.rollOver

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.rollOver = function(eventObject:Object) {
    // Insert your code here.
};
menuInstance.addEventListener("rollOver", listenerObject);
```

Usage 2:

```
on (rollOver) {
    // Insert your code here.
}
```

Description

Event; broadcast to all registered listeners when the pointer rolls over a menu item.

Version 2 components use a dispatcher-listener event model. When a Menu component broadcasts a rollOver event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.rollOver` event's event object has one additional property: `menuItem`, which is a reference to the menu item (XML node) that the pointer rolled over.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a menu with two items and a listener for a `rollOver` event. When the `rollOver` event is broadcast, a `trace()` function in the event handler, `menuItemListener`, displays the name of the menu item for which the event occurred.

You first drag a `Menu` component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus
my_menu.show(100, 20);

// Create listener object.
var menuItemListener:Object = new Object();
menuItemListener.rollOver = function(evt_obj:Object) {
    trace("Menu rollOver: "+evt_obj.menuItem.attributes.label);
};

// Add listener.
my_menu.addEventListener("rollOver", menuItemListener);
```

Menu.setMenuItemEnabled()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.setMenuItemEnabled(item, enable)
```

Parameters

item An XML node; the target menu item's node in the data provider.

enable A Boolean value indicating whether the item is enabled (`true`) or not (`false`).

Returns

Nothing.

Description

Method; changes the target item's `enabled` attribute to the state specified in the `enable` parameter. If this call results in a change of state, the item is redrawn with the new state.

Example

The following example creates a menu with two menu items and calls the `setMenuItemEnabled()` method to disable the first one.

First, you drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");
```

```
// Add the menu items.
menuDP_obj.addItem({label:"1st Item"});
menuDP_obj.addItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Select the first menu item and disable it.
var item_obj:Object = my_menu.getItemAt(0);
my_menu.setItemEnabled(item_obj, false);

// Show and position the menu.
my_menu.show(100, 20);
```

See also

[Menu.setSelected\(\)](#)

Menu.setSelected()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuInstance.setSelected(item, select)
```

Parameters

item An XML node. The target menu item's node in the data provider.

select A Boolean value indicating whether the item is selected (`true`) or not (`false`). If the item is a check box, its check mark is visible or not visible. If a selected item is a radio button, it becomes the current selection in the radio group.

Returns

Nothing.

Description

Method; changes the `selected` attribute of the item to the state specified by the `select` parameter. If this call results in a change of state, the item is redrawn with the new state. This is only meaningful for items whose `type` attribute is set to "radio" or "check", because it causes their dot or check to appear or disappear. If you call this method on an item whose `type` is "normal" or "separator", it has no effect.

Example

The following example creates a menu with two menu items, the second of which is a check box menu item. The example calls the `setMenuItemSelected()` method to put the check box menu item in a selected state.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({type:"check", label:"2nd Item"})

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

var myItem = my_menu.getMenuItemAt(1);
my_menu.setMenuItemSelected(myItem, true);

// Show and position the menu.
my_menu.show(100, 20);
```

Menu.show()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

menuInstance.show(x, y)

Parameters

x The *x* coordinate.

y The *y* coordinate.

Returns

Nothing.

Description

Method; opens a menu at a specific location. The menu is automatically resized so that all of its top-level items are visible, and the upper left corner is placed at the specified location in the coordinate system provided by the component's parent.

If the *x* and *y* parameters are omitted, the menu is shown at its previous location.

Example

The following example creates a menu from an XML menu object and calls the `menu.show()` method to display it.

You first drag a Menu component to the library; and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

var my_xml:XML = new XML();

// Create items for the menu.
var newItem_obj:Object = my_xml.addMenuItem({label:"New"});
my_xml.addMenuItem({label:"Open", instanceName:"miOpen"});
my_xml.addMenuItem({label:"Save", instanceName:"miSave"});
my_xml.addMenuItem({type:"separator"});
```



```
my_xml.addItem({label:"Quit", instanceName:"miQuit"});

// Create and show the menu.
var my_menu:Menu = Menu.createMenu(myParent_mc, my_xml);
my_menu.show(100, 20);
```

See also

[Menu.hide\(\)](#)

MenuDataProvider class

ActionScript Class Name mx.controls.menuclasses.MenuDataProvider

The MenuDataProvider class is a decorator (mix-in) class that adds functionality to the XMLNode global class. This functionality lets XML instances assigned to a Menu.dataProvider property use the MenuDataProvider methods and properties to manipulate their own data as well as the associated menu views.

Keep in mind these concepts about the MenuDataProvider class:

- MenuDataProvider is a decorator (mix-in) class. You do not need to instantiate it to use it.
- Menus natively accept XML as a dataProvider property value.
- If a Menu class is instantiated, all XML instances in the SWF file are decorated by the MenuDataProvider class.
- Only MenuDataProvider methods broadcast events to the Menu components. You can still use native XML methods, but they do not broadcast events that refresh the Menu views. To control the data model, use MenuDataProvider methods. For read-only operations like moving through the Menu hierarchy, use XML methods.
- All items in the Menu component are XML objects decorated with the MenuDataProvider class.
- Changes to item attributes are not reflected in the onscreen menu until redrawing occurs.

Method summary for the MenuDataProvider class

The following table lists the methods of the MenuDataProvider class.

Method	Description
<code>MenuDataProvider.addItem()</code>	Adds a child item.
<code>MenuDataProvider.addItemAt()</code>	Adds a child item at a specified location.
<code>MenuDataProvider.getItemAt()</code>	Gets a reference to a menu item at a specified location.
<code>MenuDataProvider.indexOf()</code>	Returns the index of a specified menu item.
<code>MenuDataProvider.removeItem()</code>	Removes a menu item.
<code>MenuDataProvider.removeItemAt()</code>	Removes a menu item at a specified location.

MenuDataProvider.addItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuDataProvider.addItem(initObject)
```

Usage 2:

```
myMenuDataProvider.addItem(childMenuItem)
```

Parameters

initObject An object containing the attributes that initialize a Menu item's attributes. For more information, see [“About menu item XML attributes” on page 887](#).

childMenuItem An XML node.

Returns

A reference to an XMLNode object.

Description

Method; Usage 1 adds a child item to the end of a parent menu item (which could be the menu itself). The menu item is constructed from the values passed in the *initObject* parameter. Usage 2 adds a child item that is defined in the specified XML *childMenuItem* parameter to the end of a parent menu item.

Any node or menu item in a *MenuDataProvider* instance can call the methods of the *MenuDataProvider* class.

Example

The following example creates a menu from an XML data provider. It calls the `addItem()` method to add two items to the main menu and also to add two items to a submenu for the first item of the main menu.

You first drag a *Menu* component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addItem("XXXXX");

// Add the menu items.
menuDP_obj.addItem({label:"Folders"});
menuDP_obj.addItem({label:"Radio Edit", type:"radio"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menu.
my_menu.show(100, 20);

// Retrieve the first menu item and add items into it.
var item_obj:Object = menuDP_obj.getMenuItemAt(0);
item_obj.addItem({label:"First item", instanceName:"firstItem1"});
item_obj.addItem({label:"Second item", instanceName:"secondItem1"});
```

MenuDataProvider.addItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuDataProvider.addItemAt(index, initObject)
```

Usage 2:

```
myMenuDataProvider.addItemAt(index, childMenuItem)
```

Parameters

index An integer.

initObject An object containing the specific attributes that initialize a Menu item's attributes. For more information, see [“About menu item XML attributes” on page 887](#).

childMenuItem An XML node.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a child item at the specified index position in the parent menu item (which could be the menu itself). The menu item is constructed from the values passed in the *initObject* parameter. Usage 2 adds a child item that is defined in the specified XML *childMenuItem* parameter to the specified index of a parent menu item.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

The following example creates a menu with one menu item and then calls the `addItemAt()` method to add a second item.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"Edit"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menu.
my_menu.show(100, 20);

// Add the menu item.
menuDP_obj.addMenuItemAt(1, {label:"Save", instanceName:"saveItem1"});
```

MenuDataProvider.getMenuItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.getMenuItemAt(index)
```

Parameters

index An integer indicating the position of the menu.

Returns

A reference to the specified XML node.

Description

Method; returns a reference to the specified child menu item of the current menu item.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

The following example creates a menu, adds a menu item to it, and then calls the `getMenuItemAt()` method to access its node object for the purpose of adding a submenu item to it. It also calls the `getMenuItemAt()` method to display the label of the submenu item in the Output panel.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
```

```
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
var menuItem_obj:Object = menuDP_obj.getMenuItemAt(0);
menuItem_obj.addMenuItem({label:"Submenu Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);

// Retrieve the submenu item from the 1st menu item.
var myMenuItem_obj:Object = menuDP_obj.firstChild;
trace(myMenuItem_obj.getMenuItemAt(0));
```

MenuDataProvider.indexOf()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myMenuDataProvider.indexOf(*item*)

Parameters

item A reference to the XML node that describes the menu item.

Returns

The index of the specified menu item; returns `undefined` if the item does not belong to this menu.

Description

Method; returns the index of the specified menu item in this parent menu item.

Any node or menu item in a `MenuDataProvider` instance can call the methods of the `MenuDataProvider` class.

Example

The following example adds a menu item to a menu and calls the `indexOf()` method to display the item's index in the Output panel.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);

// Add an item and trace the position of that item.
var myItem_obj:Object = menuDP_obj.addMenuItem({label:"That item"});
var myIndex_num:Number = menuDP_obj.indexOf(myItem_obj);
trace("Position: " + myIndex_num);
```


MenuDataProvider.removeItem()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.removeMenuItem()
```

Returns

A reference to the removed Menu item (XML node); undefined if an error occurs.

Description

Method; removes the target item and any child nodes.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

The following example creates a menu with three menu items and, after an interval of a couple of seconds (2000 milliseconds), calls `removeMenuItem()` to remove the first menu item.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();
d
// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});
menuDP_obj.addMenuItem({label:"3rd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);
// Call removeItem after 2000 milliseconds.
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
    // Remove the item at position 0.
    var myItem_obj:Object = menuDP_obj.getMenuItemAt(0);
    myItem_obj.removeMenuItem();
    clearInterval(interval_id);
}
```

MenuDataProvider.removeItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.removeMenuItemAt(index)
```

Parameters

index The index of the menu item.

Returns

A reference to the removed menu item. This value is `undefined` if there is no item in that position.

Description

Method; removes the child item of the menu item specified by the *index* parameter. If there is no menu item at that index, calling this method has no effect.

Any node or menu item in a `MenuDataProvider` instance can call the methods of the `MenuDataProvider` class.

Example

The following example creates a menu with three menu items and, after an interval of a couple of seconds (2000 milliseconds), calls `removeMenuItemAt()` to remove the first menu item.

You first drag a Menu component to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Menu component in library
 */

import mx.controls.Menu;

// Create an XML object to act as a factory.
var my_xml:XML = new XML();

// The item created next does not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name.
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// Add the menu items.
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});
menuDP_obj.addMenuItem({label:"3rd Item"});

// Create the Menu object.
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// Show and position the menus.
my_menu.show(100, 20);
// Call removeItem after 2000 milliseconds.
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
    // Remove the item at position 0.
    menuDP_obj.removeItemAt(0);
    clearInterval(interval_id);
}
```

MenuBar component (Flash Professional only)

The MenuBar component lets you create a horizontal menu bar with pop-up menus and commands, just like the menu bars that contain File and Edit menus in common software applications. The MenuBar component complements the Menu component by providing a clickable interface to show and hide menus that behave as a group for mouse and keyboard interactivity.

The MenuBar component lets you create an application menu in a few steps. To build a menu bar, you can either assign an XML data provider to the menu bar that describes a series of menus, or use the `MenuBar.addMenu()` method to add menu instances one at a time.

Each menu in the menu bar is composed of two parts: the menu and the button that causes the menu to open (called the menu activator). These clickable menu activators appear in the menu bar as a text label with inset and outset border highlight states that react to interaction from the mouse and keyboard.

When a menu activator is clicked, the corresponding menu opens below it. The menu stays active until the activator is clicked again, or until a menu item is selected or a click occurs outside the menu area.

In addition to creating menu activators that show and hide menus, the MenuBar component creates group behavior among a series of menus. This lets a user scan a large number of command choices by rolling over the series of activators or by using the arrow keys to move through the lists. Mouse and keyboard interactivity work together to let the user jump from menu to menu in the menu bar.

A user cannot scroll through menus on a menu bar. If menus exceed the width of the menu bar, they are masked.

You cannot make the MenuBar component accessible to screen readers.

Menus are often nested within menu bars. For information about menus, see [“Menu component \(Flash Professional only\)” on page 883](#).

Interacting with the MenuBar component (Flash Professional only)

You can use the mouse and keyboard to interact with a MenuBar component.

Rolling over a menu activator displays an outset border highlight around the activator label.

When a MenuBar instance has focus either from clicking or tabbing, you can use the following keys to control it:

Key	Description
Down Arrow	Moves the selection down a menu row.
Up Arrow	Moves the selection up a menu row.
Right Arrow	Moves the selection to the next button.
Left Arrow	Moves the selection to the previous button.
Enter/Escape	Closes an open menu.

NOTE

If a menu is open, you can't press the Tab key to close it. You must either make a selection or close the menu by pressing Escape.

Using the MenuBar component (Flash Professional only)

You can use the MenuBar component to add a set of menus (for example, File, Edit, Special, Window) to the top edge of an application.

MenuBar parameters

You can set the following authoring parameter for each MenuBar component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

Labels An array that adds menu activators with the specified labels to the MenuBar component. The default value is [] (an empty array).

You can set the following additional parameters for each MenuBar component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You cannot access the `Labels` parameter using `ActionScript`. However, you can write `ActionScript` to control additional options for the `MenuBar` component using its properties, methods, and events. For more information, see [“MenuBar class \(Flash Professional only\)” on page 951](#).

Creating an application with the `MenuBar` component

In this example, you drag a `MenuBar` component to the `Stage`, add code to add menu items to it, and attach a listener to the menu to respond to the selection of a menu item.

To use a `MenuBar` component in an application:

1. Select `File > New` and create a new Flash document.
2. Drag the `MenuBar` component from the `Components` panel to the `Stage`.
3. Position the menu at the top of the `Stage` for a standard layout.
4. Select the `MenuBar` instance and, in the `Property` inspector, enter the instance name **`my_mb`**.
5. In the `Actions` panel on `Frame 1`, enter the following code:

```
import mx.controls.Menu;
import mx.controls.MenuBar;

var my_mb:MenuBar;

var my_menu:Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});
my_menu.addItem({label:"Close", instanceName:"closeInstance"});
```

This code adds a `File` menu to the `MenuBar` instance. It then uses a `Menu` method to add three menu items: `New`, `Open`, and `Close`.

6. In the Actions panel on Frame 1, enter the following code:

```
//Create listener object.
var mListener:Object = new Object();
mListener.change = function(evt_obj:Object) {
    var menuItem_obj:Object = evt_obj.menuItem;
    switch (menuItem_obj.attributes.instanceName) {
        case "newInstance":
            trace("New menu item");
            break;
        case "openInstance":
            trace("Open menu item");
            break;
        case "closeInstance":
            trace("Close menu item");
            break;
    }
    trace(menuItem_obj);
};

//Add listener.
my_menu.addEventListener("change", mListener);
```

This code creates a listener object, `mListener`, that catches a menu item selection and displays its name and the value of the menu item object.

NOTE

You must call the `addEventListener()` method to register the listener with the menu instance, not with the menu bar instance.

7. Select Control > Test Movie to test the MenuBar component.

Customizing the MenuBar component (Flash Professional only)

This component sizes itself according to the activator labels that are supplied through the `dataProvider` property or the methods of the MenuBar class. When an activator button is in a menu bar, it remains at a fixed size that is dependent on the font styles and the text length.

Using styles with the MenuBar component

The MenuBar component creates an activator label for each menu in a group. You can use styles to change the look of the activator labels. A MenuBar component supports the following styles:

Style	Theme	Description
themeColor	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
color	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
embedFonts	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".

The MenuBar component also forwards all style settings for Menu style properties to the composed Menu instances. For a list of Menu style properties, see [“Using styles with the Menu component” on page 897](#).

Using skins with the MenuBar component

The MenuBar component uses three skins to represent its background, uses a movie clip symbol for highlighting individual items, and contains a Menu component as the pop-up, which itself is skinnable. The MenuBar skins are described in the following table. For information about skinning the Menu component, see [“Using skins with the Menu component” on page 900](#).

The MenuBar component supports the following skin properties.

Property	Description
menuBarBackLeftName	The up state of the pop-up icon
menuBarBackRightName	The down state of the pop-up icon
menuBarBackMiddleName	The disabled state of the pop-up icon

To create movie clip symbols for MenuBar skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library and then select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the MenuBar Assets folder to the library of your document.
4. Expand the MenuBar Assets/Elements folder in the library of your document.
5. Open the symbols that you want to customize for editing.
For example, open the MenuBarBackLeft symbol.
6. Customize the symbol as desired.
For example, change the outer edge to blank.
7. Repeat steps 5-6 for all symbols that you want to customize.
For example, set the outer edges for the middle and right symbols to black.
8. Click the Back button to return to the main timeline.
9. Drag a MenuBar component to the Stage.
10. Set MenuBar properties so that they display items on the bar.

11. Select Control > Test Movie.

NOTE

The border used to highlight individual items in a MenuBar component is an instance of ActivatorSkin found in the Flash UI Components 2/Themes/MMDefault/Button Assets folder. This symbol can be customized to point to a different class to provide a different border. However, the symbol name cannot be modified, and you cannot use a different symbol for different MenuBar instances in a single document.

MenuBar class (Flash Professional only)

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > MenuBar

ActionScript Class Name mx.controls.MenuBar

The methods and properties of the MenuBar class let you create a horizontal menu bar with pop-up menus and commands. These methods and properties complement those of the Menu class by allowing you to create a clickable interface to show and hide menus that behave as a group for mouse and keyboard interactivity.

Method summary for the MenuBar class

The following table lists methods of the MenuBar class.

Method	Description
MenuBar.addMenu()	Adds a menu to the menu bar.
MenuBar.addMenuAt()	Adds a menu at a specified location to the menu bar.
MenuBar.getMenuAt()	Gets a reference to a menu at a specified location.
MenuBar.getMenuEnabledAt()	Returns a Boolean value indicating whether a menu is enabled (<code>true</code>) or not (<code>false</code>).
MenuBar.removeMenuAt()	Removes a menu at a specified location from a menu bar.
MenuBar.removeAll()	Removes all menu items from the menu bar.
MenuBar.setMenuEnabledAt()	A Boolean value indicating whether a menu is can be chosen (<code>true</code>) or not (<code>false</code>).

Methods inherited from the UIObject class

The following table lists the methods the MenuBar class inherits from the UIObject class. When calling these methods from the MenuBar object, use the form `MenuBar.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the MenuBar class inherits from the UIComponent class. When calling these methods from the MenuBar object, use the form `MenuBar.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the MenuBar class

The following table lists properties of the MenuBar class.

Property	Description
<code>MenuBar.dataProvider</code>	The data model for a menu bar.
<code>MenuBar.labelField</code>	A string that determines which attribute of each XMLNode to use as the label text of the menu.
<code>MenuBar.labelFunction</code>	A function that determines what to display in each menu's label.

Properties inherited from the UIObject class

The following table lists the properties the MenuBar class inherits from the UIObject class.

When calling these properties from the MenuBar object, use the form

`MenuBar.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the MenuBar class inherits from the UIComponent class. When calling these properties from the MenuBar object, use the form `MenuBar.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the MenuBar class

There are no events exclusive to the MenuBar class.

Events inherited from the Menu class

The following table lists the events the MenuBar class inherits from the Menu class. When calling these events from the MenuBar object, use the form `MenuBar.eventName`.

Event	Description
<code>Menu.change</code>	Broadcast when a user causes a change in a menu.
<code>Menu.menuHide</code>	Broadcast when a menu closes.
<code>Menu.menuShow</code>	Broadcast when a menu opens.
<code>Menu.rollOut</code>	Broadcast when the pointer rolls off an item.
<code>Menu.rollOver</code>	Broadcast when the pointer rolls over an item.

Events inherited from the UIObject class

The following table lists the events the MenuBar class inherits from the UIObject class. When calling these events from the MenuBar object, use the form `MenuBar.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.

Event	Description
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the MenuBar class inherits from the UIComponent class. When calling these events from the MenuBar object, use the form `MenuBar.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

MenuBar.addMenu()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
menuBarInstance.addMenu(label)
```

Usage 2:

```
menuBarInstance.addMenu(label, menuDataProvider)
```

Parameters

label A string indicating the label of the new menu.

menuDataProvider An XML or XMLNode instance that describes the menu and its items. If the value is an XML instance, the instance's first child is used.

Returns

A reference to the new Menu object.

Description

Method; Usage 1 adds a single menu and menu activator at the end of the menu bar and uses the specified label. Usage 2 adds a single menu and menu activator that are defined in the specified XML *menuDataProvider* parameter.

Example

Usage 1: The following example adds a File menu and then uses `Menu.addItem()` to add the menu items New and Open.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */
var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});
```

Usage 2: The following example adds a Font menu with the menu items Bold and Italic, which are defined in the XML data provider `myDP_xml`.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.addItem({type:"check", label:"Bold", instanceName:"check1"});
myDP_xml.addItem({type:"check", label:"Italic",
  instanceName:"check2"});

var my_menu:mx.controls.Menu = my_mb.addMenu("Font", myDP_xml);
```


MenuBar.addMenuAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
menuBarInstance.addMenuAt(index, label)
```

Usage 2:

```
menuBarInstance.addMenuAt(index, label, menuDataProvider)
```

Parameters

index An integer indicating the position where the menu should be inserted. The first position is 0. To append to the end of the menu, call `MenuBar.addMenu(label)`.

label A string indicating the label of the new menu.

menuDataProvider An XML or XMLNode instance that describes the menu. If the value is an XML instance, the instance's first child is used.

Returns

A reference to the new Menu object.

Description

Method; Usage 1 adds a single menu and menu activator at the specified index with the specified label. Usage 2 adds a single menu and a labeled menu activator at the specified index. The content for the menu is defined in the *menuDataProvider* parameter.

Example

Usage 1: The following example places a menu in the first position on the MenuBar instance `my_mb`.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenuAt(0, "Flash");
my_menu.addItem({label:"About Macromedia Flash",
  instanceName:"aboutInst"});
my_menu.addItem({label:"Preferences", instanceName:"PrefInst"});
```

Usage 2: The following example adds an Edit menu with the menu items Undo, Redo, Cut, and Copy, which are defined in the XML data provider `myDP_xml`. It adds the menu to the first position of the MenuBar instance `my_mb`.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.addItem({label:"Undo", instanceName:"undoInst"});
myDP_xml.addItem({label:"Redo", instanceName:"redoInst"});
myDP_xml.addItem({type:"separator"});
myDP_xml.addItem({label:"Cut", instanceName:"cutInst"});
myDP_xml.addItem({label:"Copy", instanceName:"copyInst"});

my_mb.addMenuAt(0, "Edit", myDP_xml);
```

MenuBar.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

menuBarInstance.dataProvider

Description

Property; the data model for items in a MenuBar component.

MenuBar.dataProvider is an XML node object. Setting this property replaces the existing data model of the MenuBar component. Whatever child nodes the data provider might have are used as the items for the menu bar itself; any subnodes of these child nodes are used as the items for their respective menus.

The default value is undefined.

NOTE

All XML or XMLNode instances are automatically given the methods and properties of the MenuDataProvider class when they are used with the MenuBar component.

Example

The following example loads an XML menu file from a web page and uses the `onLoad` event handler to assign it to the `dataProvider` property of the MenuBar instance `my_mb`.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.ignoreWhite = true;
myDP_xml.onLoad = function(success:Boolean) {
    if (success) {
        my_mb.dataProvider = myDP_xml.firstChild;
    } else {
        trace("error loading XML file");
    }
};
myDP_xml.load("http://www.flash-mx.com/mm/xml/menubar.xml");
```

MenuBar.getMenuAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance getMenuAt(index)
```

Parameters

index An integer indicating the position of the menu.

Returns

A reference to the menu at the specified index. This value is undefined if there is no menu at that position.

Description

Method; returns a reference to the menu at the specified index. Because `getMenuAt()` returns a reference, it is possible to add items to a menu at the specified index.

Example

The following example creates a File menu and calls `getMenuAt()`, which creates a reference to it. It then uses the reference to add two menu items, New and Open, to the File menu.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

my_mb.addMenu("File");

var my_menu:mx.controls.Menu = my_mb.getMenuAt(0);
my_menu.addItem({label:"New",instanceName:"newInst"});
my_menu.addItem({label:"Open",instanceName:"openInst"});
```

MenuBar.getMenuEnabledAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance.getMenuEnabledAt(index)
```

Parameters

index The index of the menu in the menu bar.

Returns

A Boolean value that indicates whether this menu can be chosen (*true*) or not (*false*).

Description

Method; returns a Boolean value that indicates whether this menu can be chosen (*true*) or not (*false*).

Example

The following example creates a File menu with two menu items and then calls `setMenuEnabledAt()` with a value of `false` to disable it. It also calls `getMenuEnabledAt()` and displays the result to show you how to determine whether a menu is enabled.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});

//Disable "file" menu.
my_mb.setMenuEnabledAt(0, false);

//Check if "file" menu can be selected.
trace("Menu can be selected: " + my_mb.getMenuEnabledAt(0));
```

MenuBar.labelField

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance.labelField
```

Description

Property; a string that specifies which attribute of each XML node to use as the label text of the menu. The value of this property is also passed to any menus that are created from the menu bar. The default value is "label".

After the `dataProvider` property is set, this property is read-only.

Example

The following example specifies that the `name` attribute of each XML node is to provide the label text for menu items.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */
var my_mb:mx.controls.MenuBar;

//Change label text to be read from "name".
my_mb.labelField = "name";

var my_menu:mx.controls.Menu = my_mb.addMenu({name:"File"});
my_menu.addItem({name:"New", instanceName:"newInstance"});
my_menu.addItem({name:"Open", instanceName:"openInstance"});
```

MenuBar.labelFunction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance.labelFunction
```

Description

Property; a function that determines what to display in each menu's label text. The function accepts the XML node associated with an item as a parameter and returns a string to be used as label text. This property is passed to any menus created from the menu bar. The default value is undefined.

After the `dataProvider` property is set, this property is read-only.

Example

The following example uses a label function to build and return a custom label, such as New (Control +N), from the node attributes.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 */

var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", data:"Control+N",
  instanceName:"newInstance"});
my_menu.addItem({label:"Open", data:"Control+O",
  instanceName:"openInstance"});
my_menu.addItem({label:"Close", data:"Control+W",
  instanceName:"closeInstance"});

//Format XML data provided for menu.
my_menu.labelFunction = function(node:XMLNode):String {
  var attr:Object = node.attributes;
  return (attr.label + " (" + attr.data + ")");
};
```

MenuBar.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all menu items on the menu bar.

Example

The following example creates File, Edit, Tools, and Window menus on the menu bar. Then when a button is clicked, the script calls `removeAll()` to remove the menu items.

Drag the MenuBar component onto the Stage, and enter the instance name `myMenuBar` in the Property inspector. Also drag the Button component to the Stage, and enter the instance name `remBtn`. Add the following code to Frame 1 of the timeline:

```
var menu = myMenuBar.addMenu("File");
var menu = myMenuBar.addMenu("Edit");
var menu = myMenuBar.addMenu("Tools");
var menu = myMenuBar.addMenu("Window");
// Add a button that removes the menu items.
var rem_listener = new Object();
rem_listener.click = function() {
    myMenuBar.removeAll();
};
remBtn.addEventListener("click", rem_listener);
```


MenuBar.removeMenuAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuBarInstance.removeMenuAt(index)
```

Parameters

index The index of the menu to be removed from the menu bar.

Returns

A reference to the menu at the specified index in the menu bar. This value is undefined if there is no menu in that position in the menu bar.

Description

Method; removes the menu at the specified index. If there is no menu item at that index, calling this method has no effect. Also, when more than one menu is removed, the index assignments shift accordingly as each menu is removed.

Example

The following example creates a File menu and an Edit menu on the menu bar. It then calls `removeMenuAt()` to remove the menu at position 0, which is the File menu, leaving the Edit menu.

Drag an instance of the MenuBar component onto the Stage, and enter the instance name `my_mb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - MenuBar component on Stage (instance name: my_mb)
 */

import mx.controls.Menu;
import mx.controls.MenuBar;

var my_mb:MenuBar;

var file_menu:Menu = my_mb.addMenu("File");
file_menu.addItem({label:"New", instanceName:"newInstance"});
file_menu.addItem({label:"Open", instanceName:"openInstance"});

var edit_menu:Menu = my_mb.addMenu("Edit");
edit_menu.addItem({label:"Cut", instanceName:"cutInstance"});
edit_menu.addItem({label:"Copy", instanceName:"copyInstance"});
edit_menu.addItem({label:"Paste", instanceName:"pasteInstance"});

//Delete "file" menu.
my_mb.removeMenuAt(0);
```

MenuBar.setEnabledAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuItemInstance.setEnabledAt(index, boolean)
```

Parameters

index The index of the menu item to set in the MenuBar instance.

boolean A Boolean value indicating whether the menu item at the specified index is enabled (`true`) or not (`false`).

Returns

Nothing.

Description

Method; enables the menu at the specified index. If there is no menu at that index, calling this method has no effect.

Example

The following example adds a File menu to the menu bar and calls the `setMenuEnabledAt()` method to enable or disable the menu, depending on whether the `menuEnabled_ch` check box is selected or clear.

Drag an instance of the `MenuBar` component onto the Stage, and enter the instance name **my_mb** in the Property inspector. Drag a `CheckBox` component to the Stage and give it an instance name of **menuEnabled_ch**. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - MenuBar component on Stage (instance name: my_mb)
 *   - CheckBox component on Stage (instance name: menuEnabled_ch)
 */

import mx.controls.CheckBox;
import mx.controls.Menu;
import mx.controls.MenuBar;

var my_mb:MenuBar;
var menuEnabled_ch:CheckBox;

menuEnabled_ch.selected = true;
var my_menu:Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});

var chListener:Object = new Object();
chListener.click = function(evt_obj:Object) {
    // Toggle "file" menu.
    my_mb.setMenuEnabledAt(0, evt_obj.target.selected);
}
menuEnabled_ch.addEventListener("click", chListener);
```


The `NumericStepper` component allows a user to step through an ordered set of numbers. The component consists of a number in a text box displayed beside small up and down arrow buttons. When a user presses the buttons, the number is raised or lowered incrementally according to the unit specified in the `stepSize` parameter, until the user releases the buttons or until the maximum or minimum value is reached. The text in the `NumericStepper` component's text box is also editable.

The `NumericStepper` component handles only numeric data. Also, you must resize the stepper while authoring to display more than two numeric places (for example, the numbers 5246 or 1.34).

A stepper can be enabled or disabled in an application. In the disabled state, a stepper doesn't receive mouse or keyboard input. An enabled stepper receives focus if you click it or tab to it and its internal focus is set to the text box. When a `NumericStepper` instance has focus, you can use the following keys control it:

Key	Description
Down Arrow	Value changes by one unit.
Left Arrow	Moves the insertion point to the left within the text box.
Right Arrow	Moves the insertion point to the right within the text box.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.
Up Arrow	Value changes by one unit.

For more information about controlling focus, see “[FocusManager class](#)” on page 721 or “Creating custom focus navigation” in *Using Components*.

A live preview of each stepper instance reflects the setting of the value parameter in the Property inspector or Component inspector during authoring. However, there is no mouse or keyboard interaction with the stepper’s arrow buttons in the live preview.

When you add the NumericStepper component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.NumericStepperAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the NumericStepper component

You can use the NumericStepper anywhere you want a user to select a numeric value. For example, you could use a NumericStepper component in a form to allow a user to set a credit card expiration date. You could also use a NumericStepper component to allow a user to increase or decrease a font size.

NumericStepper parameters

You can set the following authoring parameters for each NumericStepper instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

maximum sets the maximum value that can be displayed in the stepper. The default value is 10. If you set a `stepSize` so that the minimum value plus the `stepSize` value at some point doesn’t equal the maximum value (minimum + `stepSize` + `stepSize` + `stepSize`, and so on), the maximum value *will* display when the stepper surpasses the maximum.

minimum sets the minimum value that can be displayed in the stepper. The default value is 0.

stepSize sets the unit by which the stepper increases or decreases with each click. The default value is 1.

value sets the value displayed in the text area of the stepper. The default value is 0.

You can set the following additional parameters for each `NumericStepper` component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to control these and additional options for the `NumericStepper` component using its properties, methods, and events. For more information, see [“NumericStepper class” on page 975](#).

Creating an application with the `NumericStepper` component

The following procedure explains how to add a `NumericStepper` component to an application while authoring. The example places a `NumericStepper` component and a `Label` component on the Stage and creates a listener for a change event on the `NumericStepper` instance. When the value in the numeric stepper changes, the example displays the new value in the `Label` instance.

To create an application with the `NumericStepper` component:

1. Drag a `NumericStepper` component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name `my_nstep`.
3. Drag a `Label` component from the Components panel to the Stage.
4. In the Property inspector, enter the instance name `my_label`.

5. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
/**
 * Requires:
 *   - NumericStepper component on Stage (instance name: my_nstep)
 *   - Label component on Stage (instance name: my_label)
 */

var my_nstep:mx.controls.NumericStepper;
var my_label:mx.controls.Label;

my_label.text = "value = " + my_nstep.value;

//Create listener object.
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
    my_label.text = "value = " + evt_obj.target.value;
};

//Add listener.
my_nstep.addEventListener("change", nstepListener);
```

The last line of code adds a change event handler to the `my_nstep` instance. The handler (`nstepListener`) assigns the current value in the numeric stepper to the text property of the Label instance.

Customizing the NumericStepper component

You can transform a `NumericStepper` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the `NumericStepper` class. (See “[NumericStepper class](#)” on page 975.)

Resizing the `NumericStepper` component does not change the size of the down and up arrow buttons. If the stepper is resized to be greater than the default height, the arrow buttons are pinned to the top and bottom of the component. The arrow buttons always appear to the right of the text box.

Using styles with the NumericStepper component

You can set style properties to change the appearance of a NumericStepper instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A NumericStepper component supports the following styles:

Style	Theme	Description
themeColor	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
color	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
embedFonts	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
textAlign	Both	The text alignment: either "left", "right", or "center". The default value is "center".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".
repeatDelay	Both	The number of milliseconds of delay between when a user first presses a button and when the action begins to repeat. The default value is 500 (half a second).

Style	Theme	Description
repeatInterval	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button. The default value is 35.
symbolColor	Sample	The color of the arrows. The default value is 0x2B333C (dark gray).

Using skins with the NumericStepper component

The NumericStepper component uses skins to represent its up and down button states. To skin the NumericStepper component while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/Stepper Assets/States folder in the library. For more information, see “About skinning components” in *Using Components*.

If a stepper is enabled, the down and up buttons display their over states when the pointer moves over them. The buttons display their down state when pressed. The buttons return to their over state when the mouse is released. If the pointer moves off the buttons while the mouse is pressed, the buttons return to their original state.

If a stepper is disabled, it displays its disabled state, regardless of user interaction.

A NumericStepper component supports the following skin properties:

Property	Description
upArrowUp	The up arrow button’s up state. The default value is StepUpArrowUp.
upArrowDown	The up arrow button’s pressed state. The default value is StepUpArrowDown.
upArrowOver	The up arrow button’s over state. The default value is StepUpArrowOver.
upArrowDisabled	The up arrow button’s disabled state. The default value is StepUpArrowDisabled.
downArrowUp	The down arrow button’s up state. The default value is StepDownArrowUp.
downArrowDown	The down arrow button’s down state. The default value is StepDownArrowDown.
downArrowOver	The down arrow button’s over state. The default value is StepDownArrowOver.
downArrowDisabled	The down arrow button’s disabled state. The default value is StepDownArrowDisabled.

To create movie clip symbols for NumericStepper skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the Stepper Assets folder to the library of your document.
4. Expand the Stepper Assets folder in the library of your document.
5. Expand the Stepper Assets/States folder in the library of your document.
6. Open the symbols that you want to customize for editing.
For example, open the StepDownArrowDisabled symbol.
7. Customize the symbol as desired.
For example, change the white inner graphics to a light gray.
8. Repeat steps 6-7 for all symbols that you want to customize.
For example, repeat the same change on the up arrow.
9. Click the Back button to return to the main timeline.
10. Drag a NumericStepper component to the Stage.
This example has customized the disabled skins, so use ActionScript to set the NumericStepper instance to be disabled in order to see the modified skins.
11. Select Control > Test Movie.

NOTE

The Stepper Assets/States folder also contains a StepTrack symbol, which is used as a spacer between the up and down skins if the total height of the NumericStepper instance is greater than the sum of the two arrow heights. This symbol linkage identifier is not available for modification through a skin property, but the library symbol can be modified, provided that the linkage identifier remains unchanged.

NumericStepper class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > NumericStepper

ActionScript Class Name mx.controls.NumericStepper

The properties of the NumericStepper class let you set the following at runtime: the minimum and maximum values displayed in the stepper, the unit by which the stepper increases or decreases in response to a click, and the current value displayed in the stepper.

Setting a property of the NumericStepper class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The NumericStepper component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in *Using Components*.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.NumericStepper.version);
```

NOTE

The code `trace(myNumericStepperInstance.version);` returns `undefined`.

Method summary for the NumericStepper class

There are no methods exclusive to the NumericStepper class.

Methods inherited from the UIObject class

The following table lists the methods the NumericStepper class inherits from the UIObject class. When calling these methods from the NumericStepper object, use the form `NumericStepper.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the NumericStepper class inherits from the UIComponent class. When calling these methods from the NumericStepper object, use the form `NumericStepper.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the NumericStepper class

The following table lists properties of the NumericStepper class.

Property	Description
<code>NumericStepper.maximum</code>	A number indicating the maximum range value.
<code>NumericStepper.minimum</code>	A number indicating the minimum range value.
<code>NumericStepper.nextValue</code>	A number indicating the next sequential value. This property is read-only.
<code>NumericStepper.previousValue</code>	A number indicating the previous sequential value. This property is read-only.
<code>NumericStepper.stepSize</code>	A number indicating the unit of change for each click.
<code>NumericStepper.value</code>	A number indicating the current value of the stepper.

Properties inherited from the UIObject class

The following table lists the properties the NumericStepper class inherits from the UIObject class. When calling these properties from the NumericStepper object, use the form `NumericStepper.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the `NumericStepper` class inherits from the `UIComponent` class. When calling these properties from the `NumericStepper` object, use the form `NumericStepper.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the NumericStepper class

The following table lists the event of the `NumericStepper` class.

Event	Description
<code>NumericStepper.change</code>	Triggered when the value of the stepper changes.

Events inherited from the UIObject class

The following table lists the events the NumericStepper class inherits from the UIObject class. When calling these events from the NumericStepper object, use the form `NumericStepper.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the NumericStepper class inherits from the UIComponent class. When calling these events from the NumericStepper object, use the form `NumericStepper.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

NumericStepper.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    //...
};
numericStepperInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the value of the stepper is changed.

A component instance (*stepperInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a listener for a *change* event on the numeric stepper called *my_nstep*. When you change the value in the numeric stepper, the listener displays the value (*value* property) in the Output panel.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - NumericStepper component on Stage (instance name: my_nstep)
 */

var my_nstep:mx.controls.NumericStepper;

// Create listener object.
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object){
    // evt_obj.target is the component that generated the change event,
    // i.e., the numeric stepper.
    trace("Value changed to " + evt_obj.target.value);
}
// Add listener.
my_nstep.addEventListener("change", nstepListener);
```

NumericStepper.maximum

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.maximum

Description

Property; the maximum range value of the stepper. This property can contain a number of up to three decimal places. The default value is 10.

Example

The following example sets the maximum value of the stepper range to 20.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - NumericStepper component on Stage (instance name: my_nstep)
 */
var my_nstep:mx.controls.NumericStepper;

my_nstep.maximum = 20;
```

See also

[NumericStepper.minimum](#)

NumericStepper.minimum

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.minimum

Description

Property; the minimum range value of the stepper. This property can contain a number of up to three decimal places. The default value is 0.

Example

The following example sets the minimum value and the initial value of the NumericStepper instance to 100 and the maximum value to 120.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - NumericStepper component on Stage (instance name: my_nstep)
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.minimum = 100;
my_nstep.maximum = 120;
my_nstep.value = my_nstep.minimum;
```

See also

[NumericStepper.maximum](#)

NumericStepper.nextValue

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.nextValue

Description

Property (read-only); the next sequential value. This property can contain a number of up to three decimal places.

Example

The following example sets the initial value of the NumericStepper component instance to -6 and sets the `stepSize` property to 3. It then displays the value of the `nextValue` property in the Output panel. You should see the same value when you click the up arrow on the stepper.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - NumericStepper component on Stage (instance name: my_nstep)
 */
var my_nstep:mx.controls.NumericStepper;

my_nstep.stepSize = 3;
my_nstep.minimum = -6;
my_nstep.maximum = 12;
my_nstep.value = my_nstep.minimum;
trace(my_nstep.nextValue); // -3
```

See also

[NumericStepper.previousValue](#)

NumericStepper.previousValue

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.previousValue

Description

Property (read-only); the previous sequential value. This property can contain a number of up to three decimal places.

Example

The following example sets the initial value of the NumericStepper instance to equal the minimum value of 6. It sets the `stepSize` value to 3 and creates a listener object for a change event. When a change event occurs, the example displays the `previousValue` property in the Output panel.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - NumericStepper component on Stage (instance name: my_nstep)
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.minimum = 6;
my_nstep.value = my_nstep.minimum;
my_nstep.maximum = 120;
my_nstep.stepSize = 3;

// Create listener object.
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
    trace("previous value = " + evt_obj.target.previousValue);
}

// Add listener.
my_nstep.addEventListener("change", nstepListener);
```

See also

[NumericStepper.nextValue](#)

NumericStepper.stepSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.stepSize

Description

Property; the unit amount to change from the current value. The default value is 1. This value cannot be 0. This property can contain a number of up to three decimal places.

Example

The following example sets the initial value of the NumericStepper instance to equal the minimum value of 3. It also sets the `stepSize` value to 3 to cause the numeric stepper to increment by 3 when the user clicks the up arrow and decrement by 3 when the user clicks the down arrow.

Drag an instance of the NumericStepper component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - NumericStepper component on Stage (instance name: my_nstep)
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.minimum = 3;
my_nstep.maximum = 120;
my_nstep.value = my_nstep.minimum;
my_nstep.stepSize = 3;
```

NumericStepper.value

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

numericStepperInstance.value

Description

Property; the current value displayed in the text area of the stepper. The value is not assigned if it does not correspond to the stepper's range and step increment as defined in the `stepSize` property. This property can contain a number of up to three decimal places.

Example

The following example sets the current value of the `NumericStepper` instance to 10 and sends the value to the Output panel.

Drag an instance of the `NumericStepper` component onto the Stage, and enter the instance name `my_nstep` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - NumericStepper component on Stage (instance name: my_nstep)
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.value = 10;
my_nstep.maximum = 100;
trace(my_nstep.value); // 10
```

ActionScript Class Name mx.managers.PopUpManager

The PopUpManager class lets you create overlapping windows that can be modal or nonmodal. (A modal window doesn't allow interaction with other windows while it's active.) You use the methods of this class to create and destroy pop-up windows.

Method summary for the PopUpManager class

The following table lists the methods of the PopUpManager class.

Method	Description
PopUpManager.createPopUp()	Creates a pop-up window.
PopUpManager.deletePopUp()	Deletes a pop-up window created by a call to PopUpManager.createPopUp() .

PopUpManager.createPopUp()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
PopUpManager.createPopUp(parent, class, modal [, initobj, outsideEvents])
```

Parameters

parent A reference to a window to pop-up over.

class A reference to the class of object you want to create.

modal A Boolean value indicating whether the window is modal (`true`) or not (`false`).

initobj An object containing initialization properties. This parameter is optional.

outsideEvents A Boolean value indicating whether an event is triggered if the user clicks outside the window (`true`) or not (`false`). This parameter is optional.

Returns

A reference to the object that was created.

If the *class* parameter is `Window` and a window component is in the library, the returned reference is a `Window`.

Description

Method; if modal, a call to `createPopUp()` finds the topmost parent window starting with *parent* and creates an instance of *class*. If nonmodal, a call to `createPopUp()` creates an instance of the class as a child of the parent window.

Example

The following code creates a modal window when the button is clicked:

```
lo = new Object();
lo.click = function(){
    mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true);
}
button.addEventListener("click", lo);
```


PopUpManager.deletePopUp()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004

Usage

```
windowInstance.deletePopUp();
```

Parameters

None.

Returns

Nothing.

Description

Method; deletes a pop-up window and removes the modal state. It is the responsibility of the overlapped window to call `PopUpManager.deletePopUp()` when the window is being destroyed.

Example

The following code creates a modal window named `win` with a close button, and deletes the window when the close button is clicked:

```
import mx.managers.PopUpManager
import mx.containers.Window
win = PopUpManager.createPopUp(_root, Window, true, {closeButton:true});
lo = new Object();
lo.click = function(){
    win.deletePopUp();
}
win.addEventListener("click", lo);
```


The `ProgressBar` component displays the progress of loading content. The `ProgressBar` is useful for displaying the status of loading images and pieces of an application. The loading process can be determinate or indeterminate. A *determinate* progress bar is a linear representation of a task's progress over time and is used when the amount of content to load is known. An *indeterminate* progress bar is used when the amount of content to load is unknown. You can add a label to display the progress of the loading content.

The `ProgressBar` component contains a left cap, a right cap, and a progress track. The caps are simply the ends of the progress bar, where the progress track visually ends. A live preview of each `ProgressBar` instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. The following parameters are reflected in the live preview: conversion, direction, label, labelPlacement, mode, and source.

Using the ProgressBar component

A progress bar lets you display the progress of content as it loads. This is essential feedback for users as they interact with an application.

There are several modes in which to use the `ProgressBar` component; you set the mode with the mode parameter. The most commonly used modes are event mode and polled mode. These modes use the source parameter to specify a loading process that either emits `progress` and `complete` events (event and polled mode), or exposes `getBytesLoaded()` and `getBytesTotal()` methods (polled mode). You can also use the `ProgressBar` component in manual mode by manually setting the `maximum`, `minimum`, and `indeterminate` properties along with calls to the `ProgressBar.setProgress()` method.

ProgressBar parameters

You can set the following authoring parameters for each ProgressBar instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

conversion is a number by which to divide the %1 and %2 values in the label string before they are displayed. The default value is 1.

direction indicates the direction toward which the progress bar fills. This value can be `right` or `left`; the default value is `right`.

label is the text indicating the loading progress. This parameter is a string in the format "%1 out of %2 loaded (%3%%)". In this string, %1 is a placeholder for the current bytes loaded, %2 is a placeholder for the total bytes loaded, and %3 is a placeholder for the percent of content loaded. The characters “%%” are a placeholder for the “%” character. If a value for %2 is unknown, it is replaced by two question marks (??). If a value is undefined, the label doesn't display.

labelPlacement indicates the position of the label in relation to the progress bar. This parameter can be one of the following values: `top`, `bottom`, `left`, `right`, `center`. The default value is `bottom`.

mode is the mode in which the progress bar operates. This value can be one of the following: `event`, `polled`, or `manual`. The default value is `event`.

source is a string to be converted into an object representing the instance name of the source.

And, you can set the following additional parameters for each ProgressBar component instance in the Component inspector (through the Window > Component Inspector menu option):

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write `ActionScript` to control these and additional options for the `ProgressBar` component using its properties, methods, and events. For more information, see [“ProgressBar class” on page 999](#).

Creating an application with the ProgressBar component

The following procedure explains how to add a ProgressBar component to an application while authoring. In this example, the progress bar is used in event mode. In event mode, the loading content must emit `progress` and `complete` events that the progress bar uses to display progress. (These events are emitted by the Loader component. For more information, see [“Loader component” on page 813.](#))

To create an application with the ProgressBar component in event mode:

1. Drag a ProgressBar component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name `my_pb`.
 - Select Event for the mode parameter.
3. Drag a Loader component from the Components panel to the Stage.
4. In the Property inspector, enter the instance name `my_ldr`.
5. Select the progress bar on the Stage and, in the Property inspector, enter `my_ldr` for the source parameter.
6. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code, which loads a JPEG file into the Loader component:

```
/**
 * Requires:
 *   - Loader component on Stage (instance name: my_ldr)
 *   - ProgressBar component on Stage (instance name: my_pb)
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.source = my_ldr;
my_ldr.autoLoad = false;
my_ldr.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// when autoLoad is false loading does not start until load() is invoked
my_ldr.load();
```

In the following example, the progress bar is used in polled mode. In polled mode, the ProgressBar uses the `getBytesLoaded()` and `getBytesTotal()` methods of the source object to display its progress.

To create an application with the **ProgressBar** component in polled mode:

1. Drag a **ProgressBar** component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name **my_pb**.
3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code, which creates a **Sound** object called **my_sound** and calls **loadSound()** to load a sound into the **Sound** object:

```
/**
 * Requires:
 * - ProgressBar component on Stage (instance name: my_pb)
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = "my_sound";

var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
    trace("Sound loaded");
}
my_pb.addEventListener("complete", pbListener);

var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
    true);
```

In the following example, the progress bar is used in manual mode. In manual mode, you must set the **maximum**, **minimum**, and **indeterminate** properties in conjunction with the **setProgress()** method to display progress. You do not set the **source** property in manual mode.

To create an application with the **ProgressBar** component in manual mode:

1. Drag a **ProgressBar** component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name **my_pb**.
 - Select **Manual** for the mode parameter.
3. Select **Frame 1** in the Timeline, open the Actions panel, and enter the following code, which updates the progress bar manually on every file download by using calls to `setProgress()`:

```
for (var i:Number = 1; i <= total; i++){  
    // insert code to load file  
    my_pb.setProgress(i, total);  
}
```

Following are two more examples.

To create an application with the **ProgressBar** component in manual mode (example 2):

1. Drag a **Label** component onto the Stage and give it an instance name **my_label**.
2. Drag a **ProgressBar** component onto the Stage and give it an instance name **my_pb**.
3. Select the **my_pb** **ProgressBar** on the Stage and, in the Property inspector, set the component's mode parameter to "manual".
4. Select **Frame 1** in the Timeline, and add the following **ActionScript** in the Actions panel:

```
var feed_xml:XML = new XML();  
feed_xml.onLoad = function(success:Boolean):Void {  
    clearInterval(timer);  
    my_label.text = "XML Loaded";  
    my_pb.setProgress(feed_xml.getBytesLoaded(),  
        feed_xml.getBytesTotal());  
};  
function updatePB(local_xml:XML):Void {  
    my_pb.setProgress(local_xml.getBytesLoaded(),  
        local_xml.getBytesTotal());  
}  
var timer:Number = setInterval(updatePB, 100, feed_xml);  
feed_xml.load("http://www.helpexamples.com/flash/xml/menu.xml");
```

5. Press **Control+Enter** to test.

To create an application with the **ProgressBar** component in manual mode (example 3):

1. Drag a **ProgressBar** component onto the Stage and give it an instance name `my_pb`.
2. Select the `my_pb` **ProgressBar** on the Stage and, in the Property inspector, set the component's mode parameter to "manual".
3. Select Frame 1 in the Timeline, and add the following **ActionScript** in the Actions panel:

```
var img_mc1:MovieClipLoader = new MovieClipLoader();
var mcListener:Object = new Object();
mcListener.onLoadProgress = function(target_mc:MovieClip,
    numBytesLoaded:Number, numBytesTotal:Number) {
    my_pb.setProgress(numBytesLoaded, numBytesTotal);
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
    //my_pb._visible = false;
};
img_mc1.addListener(mcListener);
this.createEmptyMovieClip("image_mc", 20);
img_mc1.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

NOTE

You can uncomment the line `//my_pb._visible = false;` if you want to hide the component after the content loads.

4. Press **Control+Enter** to test.

Customizing the **ProgressBar** component

You can transform a **ProgressBar** component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the **Free Transform** tool or any of the **Modify > Transform** commands. At runtime, use `UIObject.setSize()`.

The progress bar's left cap, right cap, and track graphic are set at a fixed size. When you resize a progress bar, its middle portion is resized to fit between the two caps. If a progress bar is too small, it may not render correctly.

Using styles with the **ProgressBar** component

You can set style properties to change the appearance of a progress bar instance. If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties. For more information, see "Using styles to customize component color and text" in *Using Components*.

A `ProgressBar` component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>barColor</code>	Sample	The foreground color in denoting the percent complete. The default color is white. To set the bar color on a Halo-themed component, set the <code>themeColor</code> style property.
<code>trackColor</code>	Sample	The background color for the bar. The default value is 0x666666 (dark gray).

Using skins with the ProgressBar component

The ProgressBar component uses skins to represent the progress bar track, the completed bar, and an indeterminate bar. To skin the ProgressBar component while authoring, modify symbols in the Flash UI Components 2/Themes/MMDefault/ProgressBar Elements folder. For more information, see “About skinning components” in *Using Components*.

The track and bar graphics are each made up of three skins corresponding to the left and right caps and the middle. The caps are used “as is,” and the middle is resized horizontally to fit the width of the ProgressBar instance.

The indeterminate bar is used when the ProgressBar instance’s `indeterminate` property is set to `true`. The skin is resized horizontally to fit the width of the progress bar.

A ProgressBar component supports the following skin properties:

Property	Description
<code>progTrackMiddleName</code>	The expandable middle of the track. The default value is <code>ProgTrackMiddle</code> .
<code>progTrackLeftName</code>	The fixed-size left cap. The default value is <code>ProgTrackLeft</code> .
<code>progTrackRightName</code>	The fixed-size right cap. The default value is <code>ProgTrackRight</code> .
<code>progBarMiddleName</code>	The expandable middle bar graphic. The default value is <code>ProgBarMiddle</code> .
<code>progBarLeftName</code>	The fixed-size left bar cap. The default value is <code>ProgBarLeft</code> .
<code>progBarRightName</code>	The fixed-size right bar cap. The default value is <code>ProgBarRight</code> .
<code>progIndBarName</code>	The indeterminate bar graphic. The default value is <code>ProgIndBar</code> .

To create movie clip symbols for ProgressBar skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the ProgressBar Assets folder to the library for your document.
4. Expand the ProgressBar Assets/Elements folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the ProgIndBar symbol.

6. Customize the symbol as desired.
For example, flip the track horizontally.
7. Repeat steps 5-6 for all symbols you want to customize.
8. Click the Back button to return to the main timeline.
9. Drag a `ProgressBar` component to the Stage.
To view the skins modified in this example, use `ActionScript` to set the `indeterminate` property to `true`.
10. Select `Control > Test Movie`.

ProgressBar class

Inheritance `MovieClip` > [UIObject class](#) > `ProgressBar`

ActionScript Class Name `mx.controls.ProgressBar`

Setting a property of the `ProgressBar` class with `ActionScript` overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.ProgressBar.version);
```

NOTE

The code `trace(myProgressBarInstance.version);` returns `undefined`.

Method summary for the ProgressBar class

The following table lists the method of the `ProgressBar` class.

Method	Description
ProgressBar.setProgress()	Sets the state of the progress bar to reflect the amount of progress made when the progress bar is in manual mode

Methods inherited from the UIObject class

The following table lists the methods the ProgressBar class inherits from the UIObject class. When calling these methods from the ProgressBar object, use the form `ProgressBar.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the ProgressBar class

The following table lists properties of the ProgressBar class.

Property	Description
<code>ProgressBar.conversion</code>	A number used to convert the current bytes loaded value and the total bytes loaded values.
<code>ProgressBar.direction</code>	The direction in which the progress bar fills.
<code>ProgressBar.indeterminate</code>	Indicates whether the size of the loading source is unknown.
<code>ProgressBar.label</code>	The text that accompanies the progress bar.
<code>ProgressBar.labelPlacement</code>	The location of the label in relation to the progress bar.
<code>ProgressBar.maximum</code>	The maximum value of the progress bar in manual mode.
<code>ProgressBar.minimum</code>	The minimum value of the progress bar in manual mode.
<code>ProgressBar.mode</code>	The mode in which the progress bar loads content.

Property	Description
<code>ProgressBar.percentComplete</code>	Read-only; a number indicating the percent loaded.
<code>ProgressBar.source</code>	The content to load.
<code>ProgressBar.value</code>	Read-only; indicates the amount of progress that has been made.

Properties inherited from the UIObject class

The following table lists the properties the `ProgressBar` class inherits from the `UIObject` class. When calling these properties from the `ProgressBar` object, use the form `ProgressBar.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the ProgressBar class

The following table lists events of the ProgressBar class.

Event	Description
<code>ProgressBar.complete</code>	Triggered when loading is complete.
<code>ProgressBar.progress</code>	Triggered as content loads in manual or polled mode.

Events inherited from the UIObject class

The following table lists the events the ProgressBar class inherits from the UIObject class.

When calling these events from the ProgressBar object, use the form

`ProgressBar.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

ProgressBar.complete

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object) {
    // ...
};
progressBarInstance.addEventListener("complete", listenerObject);
```

Usage 2:

```
on (complete) {
    // ...
}
```

Event object

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.complete` event: `current` (the loaded value equals total), and `total` (the total value).

Description

Event; broadcast to all registered listeners when the loading progress has completed.

The first usage example uses a dispatcher/listener event model. A component instance (*progressBarInstance*) dispatches an event (in this case, `complete`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `ProgressBar` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `progressBarInstance`, sends “_level0.progressBarInstance” to the Output panel:

```
on (complete) {
    trace(this);
}
```

Example

This example creates a `Loader` component, a `ProgressBar` (`my_pb`) for it, and a listener that makes the progress bar invisible when the `complete` event occurs. The example loads an image into the loader `my_ldr`.

You must first drag a `Loader` component and a `ProgressBar` component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.Loader, "my_ldr", 10, {autoLoad:false});
this.createClassObject(mx.controls.ProgressBar, "my_pb", 20,
    {indeterminate:true, source:my_ldr, mode:"polled"});

// Create Listener Object
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
    my_pb.visible = false;
};
// Add Listener
my_pb.addEventListener("complete", pbListener);

my_ldr.load("http://www.helpexamples.com/flash/images/image2.jpg");
```

See also

[EventDispatcher.addEventListener\(\)](#)

ProgressBar.conversion

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.conversion

Description

Property; a number that sets a conversion value for the incoming values. It divides the current and total values, floors them, and displays the converted value in the `label` property. The default value is 1.

NOTE

The floor is the closest integer value that is less than or equal to the specified value. For example, the number 4.6 becomes 4.

Example

The following code displays the progress of loading a sound object by dividing the number of bytes loaded by a conversion value of 1024 to produce a value in kilobytes.

You must first drag a `ProgressBar` component from the Component's panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 20);

//Set progress bar attributes
my_pb.mode = "polled";
my_pb.source = "my_sound";
my_pb.label = "%1 kb loaded";
my_pb.conversion = 1024;

//Load sound
var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
    true);
```

ProgressBar.direction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.direction

Description

Property; indicates the fill direction for the progress bar. A value of `right` specifies that the bar will fill from left to right. A value of `left` specifies that the bar will fill from right to left. The default value is `right`.

Example

The following code loads a sound object and marks the progress with a progress bar that fills to the left.

You must first drag a `ProgressBar` component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 *   - ProgressBar component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 20);

//Set progress bar attributes
my_pb.mode = "polled";
my_pb.source = "my_sound";
my_pb.direction = "left";

//Load sound
var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
    true);
```

ProgressBar.indeterminate

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.indeterminate

Description

Property; a Boolean value that indicates whether the progress bar has a striped fill and a loading source of unknown size (*true*), or a solid fill and a loading source of a known size (*false*). For example, you might use this property if you are loading a large data set into a SWF file and do not know the size of the data you are loading.

Example

The following code creates an indeterminate progress bar that moves from left to right with a striped fill.

You must first drag a Loader component and a ProgressBar component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

//Create Listener Object
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
trace("Height: " + evt_obj.target.height + ", Width: " +
  evt_obj.target.width);};
//Add Listener
my_pb.addEventListener("complete", pbListener);

//Set progress bar settings
my_pb.mode = "polled";
```

```
my_pb.indeterminate = true;
my_pb.source = my_ldr;

//Set loader settings
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.move(100, 100)
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

ProgressBar.label

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
progressBarInstance.label
```

Description

Property; text that indicates the loading progress. This property is a string in the format "%1 out of %2 loaded (%3%)". In this string, %1 is a placeholder for the current bytes loaded, %2 is a placeholder for the total bytes loaded, and %3 is a placeholder for the percentage of content loaded. (The characters %% allow Flash to display a single % character.) If a value for %2 is unknown, it is replaced by ??. If a value is undefined, the label doesn't display. The default value is "LOADING %3%".

Example

The following example loads an image into a loader and marks the progress with a progress bar whose label specifies the percent of total kilobytes that have been loaded.

You must first drag a Loader component and a ProgressBar component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */
```

```
System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

my_ldr.move(0, 30);

//Set progress bar settings
my_pb.mode = "polled";
my_pb.source = my_ldr;
my_pb.label = "%1 of %2 KB loaded";
my_pb.conversion = 1024; // 1024 bytes in a KB

//Set loader settings
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg")
```

See also

[ProgressBar.labelPlacement](#)

ProgressBar.labelPlacement

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.labelPlacement

Description

Property; sets the placement of the label in relation to the progress bar. The possible values are "left", "right", "top", "bottom", and "center".

Example

The following example loads an image into a loader and marks the progress with a progress bar. It sets the `labelPlacement` property to `top` to place the label above the progress bar.

You must first drag a Loader component and a ProgressBar component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

my_ldr.move(0, 30);

// Set progress bar settings
my_pb.mode = "polled";
my_pb.source = my_ldr;
my_pb.label = "%1 of %2 KB loaded";
my_pb.conversion = 1024; // 1024 bytes in a KB
my_pb.labelPlacement = "top";

// Set loader settings
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

See also

[ProgressBar.label](#)

ProgressBar.maximum

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.maximum

Description

Property; the largest value for the progress bar when the `ProgressBar.mode` property is set to "manual".

Example

The following example increments a `ProgressBar` component manually up to a `maximum` value of 200, at which point it stops. It displays the increment in the Output panel as the value increases.

Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - ProgressBar component on Stage (instance name: my_pb)
 */

var my_pb:mx.controls.ProgressBar;

//Set progress bar mode
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

//minimum numerical value before progress bar increments
my_pb.minimum = 100;

//maximum value of progress bar before it stops
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
    if (increment_num < my_pb.maximum) {
        increment_num++;
        //update progress of number incrementing
        my_pb.setProgress(increment_num, my_pb.maximum);
        trace(increment_num);
    } else {
        delete this.onEnterFrame;
    }
};
```

See also

[ProgressBar.minimum](#), [ProgressBar.mode](#)

ProgressBar.minimum

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.minimum

Description

Property; the smallest value for the progress bar when the [ProgressBar.mode](#) property is set to "manual".

Example

The following example manually increments a [ProgressBar](#) component, starting with a minimum value of 100. It displays the increment in the Output panel as the value increases.

Drag an instance of the [ProgressBar](#) component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 *   - ProgressBar component on Stage (instance name: my_pb)
 */

var my_pb:mx.controls.ProgressBar;

//Set progress bar mode
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

//minimum numerical value before progress bar increments
my_pb.minimum = 100;

//maximum value of progress bar before it stops
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
    if (increment_num < my_pb.maximum) {
        increment_num++;
        //update progress of number incrementing
        my_pb.setProgress(increment_num, my_pb.maximum);
    }
}
```



```
    trace(increment_num);
  } else {
    delete this.onEnterFrame;
  }
};
```

See also

[ProgressBar.maximum](#), [ProgressBar.mode](#)

ProgressBar.mode

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.mode

Description

Property; the mode in which the progress bar loads content. This value can be "event", "polled", or "manual".

Event mode and polled mode are the most common modes. In event mode, the `source` property specifies loading content that emits `progress` and `complete` events; you should use a Loader object in this mode. In polled mode, the `source` property specifies loading content (such as a MovieClip object) that exposes `getBytesLoaded()` and `getBytesTotal()` methods. Any object that exposes these methods can be used as a source in polled mode (including a custom object or the root timeline).

You can also use the ProgressBar component in manual mode by manually setting the `maximum`, `minimum`, and `indeterminate` properties and making calls to the [ProgressBar.setProgress\(\)](#) method.

Example

The following example loads an image into a loader and marks the progress of loading with a progress bar that is set to event mode. When the load is complete, a listener for the `complete` event displays the name of the loader object.

You must first drag a Loader component and a ProgressBar component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

//Create Listener Object
var ldrListener:Object = new Object();
ldrListener.complete = function(evt_obj:Object) {
    trace("Event complete for " + evt_obj.target);
};
//Add Listener
my_ldr.addEventListener("complete", ldrListener);

//Set progress bar settings
my_pb.mode = "event";
my_pb.indeterminate = true;
my_pb.source = my_ldr;

//Set loader settings
my_ldr.move(0,30);
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

ProgressBar.percentComplete

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.percentComplete

Description

Property (read-only); tells what percentage of the content has been loaded. This value is floored. (The floor is the closest integer value that is less than or equal to the specified value. For example, the number 7.8 becomes 7.) The following formula is used to calculate the percentage:

$$100 * (\text{value} - \text{minimum}) / (\text{maximum} - \text{minimum})$$

Example

The following example loads an image into a loader that is associated with a progress bar. A listener for the `progress` event and another for the `complete` event both access the `percentComplete` property to display the percent of loading that has completed.

Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Drag an instance of the `Loader` component onto the Stage, and enter the instance name `my_ldr` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - Loader component instance on Stage (instance name: my_ldr)
 * - Progress component instance on Stage (instance name: my_pb)
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

var pbListener:Object = new Object();
pbListener.progress = function(evt_obj:Object) {
    trace("progress = " + my_pb.percentComplete + "%");
}
pbListener.complete = function(evt_obj:Object) {
    trace("complete = " + my_pb.percentComplete + "%");
}
my_pb.addEventListener("progress", pbListener);
my_pb.addEventListener("complete", pbListener);

// when autoLoad is false loading does not start until load() is invoked
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

ProgressBar.progress

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object) {
    // ...
};
progressBarInstance.addEventListener("progress", listenerObject);
```

Usage 2:

```
on (progress) {
    // ...
}
```

Event object

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.progress` event: `current` (the loaded value equals total), and `total` (the total value).

Description

Event; broadcast to all registered listeners whenever the value of a progress bar changes.

The first usage example uses a dispatcher/listener event model. A component instance (*progressBarInstance*) dispatches an event (in this case, *progress*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `ProgressBar` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `progressBarInstance`, sends “_level0.progressBarInstance” to the Output panel:

```
on (progress) {
    trace(this);
}
```

Example

This example loads an image into a loader with an associated progress bar and creates a listener for the `progress` event. When the progress event occurs, the example displays the value property, which is a value between `ProgressBar.minimum` and `ProgressBar.maximum`.

Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Drag an instance of the `Loader` component onto the Stage, and enter the instance name `my_ldr` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - Loader component instance on Stage (instance name: my_ldr)
 * - Progress component instance on Stage (instance name: my_pb)
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

//Create Listener Object
var pbListener:Object = new Object();
pbListener.progress = function(evt_obj:Object) {
    // evt_obj.target is the component that generated the progress event,
    // i.e., the progress bar.
    trace("Current progress value = " + evt_obj.target.value);
};
//Add Listener
my_pb.addEventListener("progress", pbListener);

// when autoLoad is false loading does not start until load() is invoked
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

See also

[EventDispatcher.addEventListener\(\)](#)

ProgressBar.setProgress()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
progressBarInstance.setProgress(completed, total)
```

Parameters

completed A number indicating the amount of progress that has been made. You can use the [ProgressBar.label](#) and [ProgressBar.conversion](#) properties to display the number in percentage form or any units you choose, depending on the source of the progress bar.

total A number indicating the total progress that must be made to reach 100%.

Returns

A number indicating the amount of progress that has been made.

Description

Method; sets the state of the progress bar to reflect the amount of progress made when the [ProgressBar.mode](#) property is set to "manual". You can call this method to make the bar reflect the state of a process other than loading. For example, you might want to explicitly set the progress bar to zero progress.

The *completed* parameter is assigned to the *value* property and the *total* parameter is assigned to the *maximum* property. The *minimum* property is not altered.

Example

The following example sets the progress bar mode to `manual` and calls `setProgress()` from the `onEnterFrame()` function, which is invoked repeatedly at the frame rate of the SWF file. The example sets the minimum value for the progress bar to 100 and the maximum to 200 and marks the progress in increments of 1.

Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - ProgressBar on Stage (instance name: my_pb)
 */

var my_pb:mx.controls.ProgressBar;

//Set progress bar mode
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

//minimum numerical value before progress bar increments
my_pb.minimum = 100;

//maximum value of progress bar before it stops
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
    if (increment_num < my_pb.maximum) {
        increment_num++;
        //update progress of number incrementing
        my_pb.setProgress(increment_num, my_pb.maximum);
        trace(increment_num);
    } else {
        delete this.onEnterFrame;
    }
};
```

ProgressBar.source

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.source

Description

Property; a reference to the instance to be loaded whose loading process will be displayed. The loading content should emit a `progress` event from which the current and total values are retrieved. This property is used only when `ProgressBar.mode` is set to "event" or "polled". The default value is `undefined`.

The ProgressBar component can be used with content within an application, including `_root`.

Example

The following example loads an image into a loader and marks the progress with a progress bar. The example sets the `source` property to the name of the Loader component (`my_loader`) to associate the content with the progress bar.

You must first drag a Loader component and a ProgressBar component from the Components panel to the current document's library; then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - ProgressBar component in library
 * - Loader component in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

//Create Listener Object
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
    evt_obj.target.visible = false;
};
//Add Listener
my_pb.addEventListener("complete", pbListener);

//Set progress bar settings
my_pb.mode = "polled";
my_pb.indeterminate = true;
my_pb.source = my_ldr;

//Set loader settings
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

See also

[ProgressBar.mode](#)

ProgressBar.value

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

progressBarInstance.value

Description

Property (read-only); indicates the amount of progress that has been made. This property is a number between the value of `ProgressBar.minimum` and `ProgressBar.maximum`. The default value is 0.

Example

The following example loads an image into a Loader component and marks the progress with the a progress bar. When the loading is complete, the example displays the minimum, maximum, and current values for the progress bar.

Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Drag an instance of the `Loader` component onto the Stage, and enter the instance name `my_ldr` in the Property inspector. Add the following code to Frame 1 of the timeline:

```
/**
 * Requires:
 * - Loader component instance on Stage (instance name: my_ldr)
 * - Progress component instance on Stage (instance name: my_pb)
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

//Create Listener Object
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object){
    // event_obj.target is the component that generated the complete event,
    // i.e., the progress bar.
    trace("Minimum value is: " + evt_obj.target.minimum + " bytes");
    trace("Maximum value is: " + evt_obj.target.maximum + " bytes");
    trace("Current ProgressBar value = " + evt_obj.target.value + " bytes");
}
//Add Listener
my_pb.addEventListener("complete", pbListener);

// when autoLoad is false loading does not start until load() is invoked
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

The `RadioButton` component lets you force a user to make a single choice within a set of choices. This component must be used in a group of at least two `RadioButton` instances. Only one member of the group can be selected at any given time. Selecting one radio button in a group deselects the currently selected radio button in the group. You set the `groupName` parameter to indicate which group a radio button belongs to.

A radio button can be enabled or disabled. A disabled radio button doesn't receive mouse or keyboard input. When the user clicks or tabs into a `RadioButton` component group, only the selected radio button receives focus. The user can then use the following keys control it:

Key	Description
Up Arrow/Left Arrow	The selection moves to the previous radio button within the radio button group.
Down Arrow/Right Arrow	The selection moves to the next radio button within the radio button group.
Tab	Moves focus from the radio button group to the next component.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each `RadioButton` instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, the mutual exclusion of selection does not display in the live preview. If you set the `selected` parameter to true for two radio buttons in the same group, they both appear selected even though only the last instance created appears selected at runtime. For more information, see [“RadioButton parameters” on page 1024](#).

When you add the `RadioButton` component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.RadioButtonAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the RadioButton component

A radio button is a fundamental part of any form or web application. You can use radio buttons wherever you want a user to make one choice from a group of options. For example, you would use radio buttons in a form to ask which credit card a customer wants to use.

RadioButton parameters

You can set the following authoring parameters for each RadioButton component instance in the Property inspector or in the Component inspector:

data is the value associated with the radio button. There is no default value.

groupName is the group name of the radio button. The default value is `radioGroup`.

label sets the value of the text on the button. The default value is `Radio Button`.

labelPlacement orients the label text on the button. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`. The default value is `right`. For more information, see [RadioButton.labelPlacement](#).

selected sets the initial value of the radio button to `selected` (`true`) or `unselected` (`false`). A selected radio button displays a dot inside it. Only one radio button in a group can have a selected value of `true`. If more than one radio button in a group is set to `true`, the radio button that is instantiated last is selected. The default value is `false`.

You can write ActionScript to set additional options for RadioButton instances using the methods, properties, and events of the RadioButton class. For more information, see “[RadioButton class](#)” on page 1029.

Creating an application with the RadioButton component

The following procedure explains how to add RadioButton components to an application while authoring. In this example, the radio buttons are used to present the yes-or-no question “Are you a Flashist?”. The data from the radio group is displayed in a TextArea component with the instance name `theVerdict`.

To create an application with the `RadioButton` component:

1. Drag two `RadioButton` components from the Components panel to the Stage.
2. Select one of the radio buttons. In the Component inspector, do the following:
 - Enter **Yes** for the label parameter.
 - Enter **Flashist** for the data parameter.
3. Select the other radio button. In the Component inspector, do the following:
 - Enter **No** for the label parameter.
 - Enter **Anti-Flashist** for the data parameter.
4. Drag a `TextArea` component from the Components panel to the Stage and give it an instance name of `theVerdict`.
5. Select Frame 1 in the main Timeline, open the Actions panel, and enter the following code:

```
flashistListener = new Object();
flashistListener.click = function (evt){
    theVerdict.text = evt.target.selection.data
}
radioGroup.addEventListener("click", flashistListener);
```

The last line of code adds a `click` event handler to the `radioGroup` radio button group. The handler sets the `text` property of `theVerdict` (a `TextArea` instance) to the value of the `data` property of the selected radio button in the `radioGroup` radio button group. For more information, see [RadioButton.click](#).

Customizing the `RadioButton` component

You can transform a `RadioButton` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

The bounding box of a `RadioButton` component is invisible and also designates the hit area for the component. If you increase the size of the component, you also increase the size of the hit area.

If the component's bounding box is too small to fit the component label, the label is clipped to fit.

Using styles with the RadioButton component

You can set style properties to change the appearance of a `RadioButton`. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A `RadioButton` component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".

Style	Theme	Description
<code>symbolBackgroundColor</code>	Sample	The background color of the radio button. The default value is <code>OxFFFFFFFF</code> (white).
<code>symbolBackgroundDisabledColor</code>	Sample	The background color of the radio button when disabled. The default value is <code>OxEFEEEEF</code> (light gray).
<code>symbolBackgroundPressedColor</code>	Sample	The background color of the radio button when pressed. The default value is <code>OxFFFFFFFF</code> (white).
<code>symbolColor</code>	Sample	The color of the dot in the radio button. The default value is <code>Ox000000</code> (black).
<code>symbolDisabledColor</code>	Sample	The color of the dot in the radio button when the component is disabled. The default value is <code>Ox848384</code> (dark gray).

Using skins with the RadioButton component

You can skin the `RadioButton` component while authoring by modifying the component's symbols in the library. The skins for the `RadioButton` component are located in the following folder in the library of `HaloTheme.fla` or `SampleTheme.fla`: `Flash UI Components 2/Themes/MMDefault/RadioButton Assets/States`. For more information, see “About skinning components” in *Using Components*.

If a radio button is enabled and unselected, it displays its rollover state when a user moves the pointer over it. When a user clicks an unselected radio button, the radio button receives input focus and displays its false pressed state. When a user releases the mouse, the radio button displays its true state and the previously selected radio button in the group returns to its false state. If a user moves the pointer off a radio button while pressing the mouse, the radio button's appearance returns to its false state and it retains input focus.

If a radio button or radio button group is disabled, it displays its disabled state, regardless of user interaction.

A `RadioButton` component uses the following skin properties:

Name	Description
<code>falseUpIcon</code>	The unselected state. The default value is <code>RadioFalseUp</code> .
<code>falseDownIcon</code>	The pressed-unselected state. The default value is <code>RadioFalseDown</code> .
<code>falseOverIcon</code>	The over-unselected state. The default value is <code>RadioFalseOver</code> .

Name	Description
falseDisabledIcon	The disabled-unselected state. The default value is RadioFalseDisabled.
trueUpIcon	The selected state. The default value is RadioTrueUp.
trueDisabledIcon	The disabled-selected state. The default value is RadioTrueDisabled.

Each of these skins corresponds to the icon indicating the RadioButton state. The RadioButton does not have a border or background.

To create movie clip symbols for RadioButton skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library and then select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the RadioButton Assets folder to the library of your document.
4. Expand the RadioButton Assets/States folder in the library of your document.
5. Open the symbols that you want to customize for editing.
For example, open the RadioFalseDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white circle to a light gray.
7. Repeat steps 5-6 for all symbols that you want to customize.
For example, repeat the color change for the inner circle of the RadioTrueDisabled symbol.
8. Click the Back button to return to the main timeline.
9. Drag a RadioButton component to the Stage.
For this example, drag two instances to show the two new skin symbols.
10. Set the RadioButton instance properties as desired.
For this example, set one RadioButton to selected, and use ActionScript to set both RadioButton instances to disabled.
11. Select Control > Test Movie.

RadioButton class

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [SimpleButton class](#) > [Button component](#) > [RadioButton](#)

ActionScript Package Name mx.controls.RadioButton

The properties of the RadioButton class allow you at runtime to create a text label and position it in relation to the radio button. You can also assign data values to radio buttons, assign them to groups, and select them based on data value or instance name.

Setting a property of the RadioButton class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The RadioButton component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For information about creating focus navigation, see “Creating custom focus navigation” in *Using Components*.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.RadioButton.version);
```

NOTE

The code `trace(myRadioButtonInstance.version);` returns `undefined`.

Method summary for the RadioButton class

There are no methods exclusive to the RadioButton class.

Methods inherited from the UIObject class

The following table lists the methods the RadioButton class inherits from the UIObject class.

When calling these methods from the RadioButton object, use the form

RadioButtonInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.

Method	Description
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the `RadioButton` class inherits from the `UIComponent` class. When calling these methods from the `RadioButton` object, use the form

RadioButtonInstance.methodName.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the RadioButton class

The following table lists properties of the `RadioButton` class.

Property	Description
<code>RadioButton.data</code>	The value associated with a radio button instance.
<code>RadioButton.groupName</code>	The group name for a radio button group instance or a radio button instance.
<code>RadioButton.label</code>	The text that appears next to a radio button.
<code>RadioButton.labelPlacement</code>	The orientation of the label text in relation to a radio button or a radio button group.
<code>RadioButton.selected</code>	Selects the radio button, and deselects the previously selected radio button. This property can be used with a <code>RadioButton</code> instance or a <code>RadioButtonGroup</code> instance.

Property	Description
<code>RadioButton.selectedData</code>	Selects the radio button with the specified data value in a radio button group.
<code>RadioButton.selection</code>	A reference to the currently selected radio button in a radio button group. This property can be used with a <code>RadioButton</code> instance or a <code>RadioButtonGroup</code> instance.

Properties inherited from the UIObject class

The following table lists the properties the `RadioButton` class inherits from the `UIObject` class. When accessing these properties from the `RadioButton` object, use the form *RadioButtonInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `RadioButton` class inherits from the `UIComponent` class. When accessing these properties from the `RadioButton` object, use the form `RadioButtonInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `SimpleButton` class

The following table lists the properties `RadioButton` class inherits from the `SimpleButton` class. When accessing these properties from the `RadioButton` object, use the form `RadioButtonInstance.propertyName`.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the Button class

The following table lists the properties the RadioButton class inherits from the Button class. When accessing these properties from the RadioButton object, use the form *RadioButtonInstance.propertyName*.

Property	Description
Button.icon	Specifies an icon for a button instance.
Button.label	Specifies the text that appears in a button.
Button.labelPlacement	Specifies the orientation of the label text in relation to an icon.

Event summary for the RadioButton class

The following table lists the event of the RadioButton class.

Event	Description
RadioButton.click	Triggered when the mouse button is pressed over a radio button or radio button group.

Events inherited from the UIObject class

The following table lists the events the RadioButton class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `RadioButton` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the `SimpleButton` class

The following table lists the event the `RadioButton` class inherits from the `SimpleButton` class.

Event	Description
<code>SimpleButton.click</code>	Broadcast when the mouse is clicked (released) over a button or if the button has focus and the Spacebar is pressed.

RadioButton.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object) {
    // ...
};
radioButtonGroup.addEventListener("click", listenerObject);
```

Usage 2:

```
on (click) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (pressed and released) over the radio button or if the radio button is selected by means of the arrow keys. The event is also broadcast if the Spacebar or arrow keys are pressed when a radio button group has focus, but none of the radio buttons in the group are selected.

The first usage example uses a dispatcher/listener event model. A component instance (*radioButtonInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `RadioButton` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the radio button `myRadioButton`, sends “_level0.myRadioButton” to the Output panel:

```
on (click) {
    trace(this);
}
```

Example

The following example creates three radio buttons, positions them on the Stage, and creates a listener for the click event. When a user clicks one of the three radio buttons, the listener displays the instance name of the selected radio button.

You first drag a `RadioButton` component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 *   - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
    groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
    groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
    groupName:"radioGroup"});

// Position RadioButtons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

// Create listener object.
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
    trace("The selected radio instance is " + evt_obj.target.selection);
}
// Add listener.
radioGroup.addEventListener("click", rbListener);
```


RadioButton.data

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.data

Description

Property; specifies the data to associate with a RadioButton instance. Setting this property overrides the data parameter value set during authoring. The `data` property can be of any data type.

Example

The following example assigns the data value `0xFF00FF` and the label `#FF00FF` to the radio button instance `my_rb`. It then creates a listener for a click event and displays the button's data value when a user clicks the button.

You first drag a RadioButton component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 * - RadioButton component in library
 */

this.createClassObject(mx.controls.RadioButton, "my_rb", 10,
    {label:"first", groupName:"radioGroup"});

my_rb.data = 0xFF00FF;
my_rb.label = "#FF00FF";

var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
    trace("The data value for my_rb is " + my_rb.data);
}
// Add listener.
my_rb.addEventListener("click", rbListener);
```

RadioButton.groupName

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.groupName

radioButtonGroup.groupName

Description

Property; sets the group name for a radio button instance or group. You can use this property to get or set a group name for a radio button instance or for a radio button group. Calling this method overrides the groupName parameter value set during authoring. The default value is "radioGroup".

Example

The following example sets the group name for a group of three radio buttons to myrbGroup. It positions the buttons and then creates a listener for a click event on the radio button group. When the user clicks a radio button, the example displays the groupName property for the button that was clicked.

You first drag a RadioButton component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline:

```
/**
 * Requires:
 *   - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
    groupName:"myrbGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
    groupName:"myrbGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
    groupName:"myrbGroup"});

// Position radio buttons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);
```

```
// Create listener object.
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
    trace("The selected radio button group name is " +
        evt_obj.target.groupName);
}
// Add listener.
myrbGroup.addEventListener("click", rbListener);
```

RadioButton.label

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.label

Description

Property; specifies the text label for the radio button. By default, the label appears to the right of the radio button. Calling this method overrides the label parameter specified during authoring. If the label text is too long to fit within the bounding box of the component, the text is clipped. You can call `RadioButton.setSize()` to increase the size of the label area, but text does not wrap to the next line.

To provide a label with text that wraps, you can combine a `RadioButton` with no label and a `TextArea` to act as a single `RadioButton` with wrapping text. The following example creates such a radio button. It assumes that you have a `RadioButton` component and a `TextArea` component in the library and turns off the border for the `TextArea`. The `label` property would be undefined in this case, if you accessed it.

```
this.createClassObject(mx.controls.RadioButton, "sameas_rb", 1,
    {groupName:"myGroup"});
sameas_rb.move(0,30)
this.createClassObject(mx.controls.TextArea, "message_ta", 2);
message_ta.setSize(200, 60);
// Turn off the border for the TextArea.
message_ta.borderStyle = "none";
message_ta.wordWrap = true;
message_ta.text = "Click here if your shipping information is the same as
    your billing information.";
message_ta.move(20, 30);
```

Example

The following example creates a radio button and assigns it a label of “Remove from list.”

You first add a `RadioButton` component from the Components panel to the current document’s library, and then add the following code to Frame 1 of the main timeline.

```
/**
 * Requires:
 *   - RadioButton component in library
 */

this.createClassObject(mx.controls.RadioButton, "my_rb", 10);

// Resize RadioButton component.
my_rb.setSize(200, my_rb.height);
my_rb.label = "Remove from list";
```

RadioButton.labelPlacement

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.labelPlacement

radioButtonGroup.labelPlacement

Description

Property; a string that indicates the position of the label in relation to a radio button. You can set this property for an individual instance or for a radio button group. If you set the property for a group, the label is placed in the appropriate position for each radio button in the group.

The following are the four possible values:

- "right" The radio button is pinned to the upper left corner of the bounding area. The label is placed to the right of the radio button.
- "left" The radio button is pinned to the upper right corner of the bounding area. The label is placed to the left of the radio button.
- "bottom" The label is placed below the radio button. The radio button and label grouping are centered horizontally and vertically. If the bounding box of the radio button isn't large enough, the label is clipped.

- "top" The label is placed above the radio button. The radio button and label grouping are centered horizontally and vertically. If the bounding box of the radio button isn't large enough, the label is clipped.

Example

The following code creates three radio buttons and uses the `labelPlacement` property to place the label for the second button on the left of the button.

You first drag a `RadioButton` component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline.

```
/**
 * Requires:
 *   - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
  groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
  groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
  groupName:"radioGroup"});

// Position radio buttons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

second_rb.labelPlacement = "left";
```

RadioButton.selected

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.selected

radioButtonGroup.selected

Description

Property; a Boolean value that sets the state of the radio button to selected (`true`) and deselects the previously selected radio button, or sets the radio button to deselected (`false`).

Example

The following example creates three radio buttons in a radio group, positions the buttons, and sets the `selected` property to `true` to put it in the selected state.

You first drag a `RadioButton` component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline.

```
/**
 * Requires:
 * - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
    groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
    groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
    groupName:"radioGroup"});

// Position radio buttons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

first_rb.selected = true;
```

RadioButton.selectedData

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`radioButtonGroup.selectedData`

Description

Property; selects the radio button with the specified data value and deselects the previously selected radio button. If the `data` property is not specified for a selected instance, the `label` value of the selected instance is selected and returned. The `selectedData` property can be of any data type.

Example

The following example creates three radio buttons in a radio group, positions the buttons, and selects the button that has a data value of 10, which is the second button.

You first drag a `RadioButton` component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline.

```
/**
 * Requires:
 * - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first", data:5,
    groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
    data:10, groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
    data:15, groupName:"radioGroup"});

// Position radio buttons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

radioGroup.selectedData = 10;
```

RadioButton.selection

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.selection

radioButtonGroup.selection

Description

Property; behaves differently depending on whether you get or set the property. If you get the property, it returns the object reference of the currently selected radio button in a radio button group. If you set the property, it selects the specified radio button (passed as an object reference) in a radio button group and deselects the previously selected radio button.

Example

The following example creates three radio buttons in a radio group, positions the buttons, and creates a listener for a click event on the radio group. When the user clicks a radio button, the listener uses the `selection` property to display the instance name of the button that was clicked.

You first drag a `RadioButton` component from the Components panel to the current document's library, and then add the following code to Frame 1 of the main timeline.

```
/**
 * Requires:
 *   - RadioButton component in library
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
  groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
  groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
  groupName:"radioGroup"});

// Position radio buttons on Stage.
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

// Create listener object.
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
  trace("The selected radio instance is " + radioGroup.selection);
}
// Add listener.
radioGroup.addEventListener("click", rbListener);
```


RadioButtonGroup component

For information about the `RadioButtonGroup` class, see [RadioButton component](#).

RDBMSResolver component (Flash Professional only)

Resolver components are used with the DataSet component (part of the data management functionality in the Flash data architecture) to save changes to an external data source. Resolvers include both the RDBMSResolver component and the XUpdateResolver component. Resolvers take a delta packet (returned by `DataSet.deltaPacket`) and convert it to an update packet in a format appropriate to the type of resolver. The update packet can then be transmitted to the external data source by one of the connector components. Resolver components have no visual appearance at runtime.

The RDBMSResolver component creates an XML update packet that can be easily parsed into SQL statements for updating a relational database. The RDBMSResolver component is connected to the `DeltaPacket` property of a DataSet component, sends its own update packet to a connector, receives server errors back from the connector, and communicates them back to the DataSet component—all using bindable properties.

For more information about the Flash data architecture, see “Data resolution (Flash Professional only)” in *Using Flash*. For more information about relational databases, see “Resolving data to a relational database (Flash Professional only)” in *Using Flash*. For a complete example of an application that updates data using the RDBMSResolver component, see www.macromedia.com/devnet/mx/flash/articles/delta_packet.html.

NOTE

You can use the RDBMSResolver component to send data updates to any object you write that can parse XML and generate SQL statements against a database—for example, an ASP page, a Java servlet, or a ColdFusion component.

Using the RDBMSResolver component (Flash Professional only)

You use the RDBMSResolver component only when your Flash application contains a DataSet component and must send an update back to the data source. This component resolves data that you want to return to a relational database.

RDBMSResolver parameters

You can set the following authoring parameters for each RDBMSResolver instance by using the Parameters tab of the Component inspector:

TableName is a string representing the name (in the XML) of the database table to be updated. This string should match the name of the `RDBMSResolver.fieldInfo` item to be updated. If there are no updates to this field, this parameter should be blank, which is the default value.

UpdateMode is an enumerator that determines the way key fields are identified when the XML update packet is generated. Possible values are as follows:

- `umUsingAll` Uses the old values of all of the modified fields to identify the record to be updated. This is the safest value to use for updating, because it guarantees that another user has not modified the record since you retrieved it. However, this approach is time consuming and generates a larger update packet.
- `umUsingModified` Uses the old values of all of the fields modified to identify the record to be updated. This value guarantees that another user has not modified the same fields in the record since you retrieved it.
- `umUsingKey` The default value. This setting uses the old value of the key fields. This implies an “optimistic concurrency” model, which most database systems today employ, and guarantees that you are modifying the same record that you retrieved from the database. Your changes overwrites any other user’s changes to the same data.

NullValue is a string representing a null field value. You can customize this parameter to prevent it from being confused with an empty string (" ") or another valid value. The default value is `{_NULL_}`.

FieldInfo is a collection representing one or more key fields that uniquely identify the records. If your data source is a database table, the table should have one or more fields that uniquely key the records within it. Additionally, some fields may have been calculated or joined from other tables. Those fields must be identified so that the key fields can be set within the XML update packet, and so that any fields that should not be updated are omitted from the XML update packet.

The `FieldInfo` parameter lets you use properties to designate fields that require special handling. Each item in the collection contains three properties:

- `FieldName` Name of a field. This should match a field name in the `DataSet` component.
- `OwnerName` Optional value used to identify fields not “owned” by the same table defined in the `RDBMSResolver` component’s `TableName` parameter. If this property has the same value as the `TableName` parameter or is blank, usually the field is included in the XML update packet. If it has a different value, this field is excluded from the update packet.
- `IsKey` Boolean property that you should set to `true` so that all key fields for the table are updated.

The following example shows `FieldInfo` items that are created to update fields in a customer table. You must identify the key fields in the customer table. The customer table has a single key field, `id`; therefore, you should create a field item with the following values:

```
FieldName = "id"  
OwnerName = <--! leave this value blank -->  
IsKey = "true"
```

Also, the `custType` field is added by means of a join in the query. Because this field should be excluded from the update, you create a field item with the following values:

```
FieldName = "custType"  
OwnerName = "JoinedField"  
IsKey = "false"
```

When the field items are defined, Flash Player can use them to automatically generate the complete XML, which is used to update a table.

NOTE

The `FieldInfo` parameter makes use of a Flash feature called the Collection Editor. When you select the `FieldInfo` parameter, you can use the Collection Editor dialog box to add new `FieldInfo` items and set their `fieldName`, `ownerName`, and `isKey` properties from one location.

Common workflow for the RDBMSResolver component

The following steps describe the typical workflow for the RDBMSResolver component.

To use an RDBMSResolver component:

1. Add two instances of the `WebServiceConnector` component and one instance of the `DataSet` and `RDBMSResolver` components to your application, and give them instance names.
2. Select the first `WebServiceConnector` component. Then use the `Parameters` tab of the `Component inspector` to enter the `Web Service Definition Language (WSDL)` URL for a web service that exposes data from an external data source.

NOTE

The web service must return an array of records to be bound to the data set.

3. Use the `Bindings` tab of the `Component inspector` to bind the first `WebServiceConnector` component's `results` property to the `DataSet` component's `dataProvider` property.
4. Select the `DataSet` component, and use the `Bindings` tab of the `Component inspector` to bind data elements (`DataSet` fields) to the visual components in your application.
5. Bind the `DataSet`'s `deltaPacket` property to the `RDBMSResolver`'s `deltaPacket` property.

The update instructions are sent from the `DataSet` component to the `RDBMSResolver` component when the `DataSet.applyUpdates()` method is called.

6. Bind the `RDBMSResolver` `updatePacket` property to the second `WebServiceConnector` `params` property to send data back to a method that parses the XML update packet. Set the kind of that `params` property to `auto-trigger` so that the connector sends the update packet as soon as data binding copies it over.
7. Add a trigger to initiate the data binding operation: use the `Trigger Data Source` behavior attached to a button, or add `ActionScript`.

In addition to these steps, you can also use the `RDBMSResolver` component to create bindings to apply the result packet sent back from the server to the data set.

For a step-by-step example that resolves data to a relational database using the `RDBMSResolver` component, see the tutorials on DevNet at http://www.macromedia.com/devnet/mx/flash/data_integration.html.

RDBMSResolver class (Flash Professional only)

Inheritance MovieClip > RDBMSResolver

ActionScript Package Name mx.data.components.RDBMSResolver

The methods, properties, and events of the RDBMSResolver class allow you to connect to a DataSet component and make changes to external data sources.

Method summary for the RDBMSResolver component

The following table lists the method of the RDBMSResolver class.

Method	Description
RDBMSResolver.addFieldInfo()	Adds a new item to the <code>fieldInfo</code> collection, which is used for setting up an RDBMSResolver component dynamically at runtime.

Property summary for the RDBMSResolver component

The following table lists properties of the RDBMSResolver class.

Property	Description
RDBMSResolver.deltaPacket	The <code>deltaPacket</code> property of the DataSet object should be bound to this property so that when <code>DataSet.applyUpdates()</code> is called, the binding copies it across and the resolver creates the update packet.
RDBMSResolver.fieldInfo	A collection of fields with properties that identify DataSet fields that require special handling, either because they are key fields or because they cannot be updated.
RDBMSResolver.nullValue	A string that is placed in the update packet to indicate that the value of a field is <code>null</code> .
RDBMSResolver.tableName	Identifies the database table that is to be updated.
RDBMSResolver.updateMode	Values that determine how key fields are identified when the XML update packet is generated.

Property	Description
<code>RDBMSResolver.updatePacket</code>	The XML packet produced by this resolver that contains the changes from the data set's delta packet.
<code>RDBMSResolver.updateResults</code>	A delta packet that contains the results of an update returned from the server through a connector.

Event summary for the RDBMSResolver component

The following table lists the events of the RDBMSResolver class.

Event	Description
<code>RDBMSResolver.beforeApplyUpdates</code>	Defined in your application; called by the RDBMSResolver component to make custom modifications to the XML of the <code>updatePacket</code> property before it is bound to the connector.
<code>RDBMSResolver.reconcileResults</code>	Defined in your application; called by the RDBMSResolver component to compare two packets after results have been received from the server and applied to the delta packet.
<code>RDBMSResolver.reconcileUpdates</code>	Defined in your application; called by the RDBMSResolver component when results have been received from the server after the updates from a delta packet were applied.

RDBMSResolver.addFieldInfo()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.addFieldInfo("fieldName", "ownerName", "isKey")
```

Parameters

fieldName String; provides the name of the field that this information object describes.

ownerName String; provides the name of the table that owns this field. If this name is the same as the RDBMSResolver instance's `tableName` property, you can leave this parameter blank ("").

isKey Boolean; indicates whether this field is a key field.

Returns

Nothing.

Description

Method; adds a new item to the XML `fieldInfo` collection in the update packet. Use this method if you must set up an RDBMSResolver component dynamically at runtime, rather than using the Component inspector in the authoring environment.

Example

The following example creates an RDBMSResolver component and provides the name of the table, provides the name of the key field, and prevents the `personTypeName` field from being updated:

```
var myResolver:RDBMSResolver = new RDBMSResolver();
myResolver.tableName = "Customers";
// Sets up the id field as a key field
// and the personTypeName field so it won't be updated.
myResolver.addFieldInfo("id", "", true);
myResolver.addFieldInfo("personTypeName", "JoinedField", false);
// Sets up the data bindings
//...
```

RDBMSResolver.beforeApplyUpdates

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.beforeApplyUpdates(eventObject)
```

Parameters

eventObject Resolver event object; describes the customizations to the XML packet before the update is sent through the connector to the database. This event object should contain the following properties:

Property	Description
target	Object; the resolver producing this event.
type	String; the name of the event.
updatePacket	XML object; the XML object about to be applied.

Returns

Nothing.

Description

Property; a property of type `deltaPacket`. It receives a delta packet to be translated into an update packet, and outputs a delta packet from any server results placed in the `updateResults` property. This event handler provides a way for you to make custom modifications to the XML before sending the updated data to a connector.

Messages in the `updateResults` property are treated as errors. This means that a delta with messages is added to the delta packet again so it can be re-sent the next time the delta packet is sent to the server. You must write code to handle deltas that have messages so that the messages are presented to the user and the deltas can be modified before being added to the next delta packet.

Example

The following example adds the user authentication data to the XML packet:

```
on (beforeApplyUpdates) {
    // Add user authentication data.
    var userInfo = new XML("" + getUserId() + "+getPassword() + "");
    updatePacket.firstChild.appendChild(userInfo);
}
```

RDBMSResolver.deltaPacket

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.deltaPacket

Description

Property; a property of type `deltaPacket`. It receives a delta packet to be translated into an update packet, and outputs a delta packet from any server results placed in the `updateResults` property.

Messages in the `updateResults` property are treated as errors. This means that a delta with messages is added to the delta packet again so it can be re-sent the next time the delta packet is sent to the server. You must write code to handle deltas that have messages so that the messages are presented to the user and the deltas can be modified before being added to the next delta packet.

RDBMSResolver.fieldInfo

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.fieldInfo

Description

Property; specifies a collection of an unlimited number of fields with properties that identify DataSet fields that require special handling, either because they are key fields or because they cannot be updated (for information about adding a field, see [RDBMSResolver.addFieldInfo\(\)](#)). Each `fieldInfo` item in the collection contains three properties:

Property	Description
<code>fieldName</code>	Name of the special-case field. This field name should match a field name in the DataSet component.
<code>ownerName</code>	An optional property. If this field is not “owned” by the table defined in the <code>RDBMSResolver.tableName</code> property, <code>ownerName</code> is the name of the owner of this field. If <code>ownerName</code> has the same value as <code>RDBMSResolver.tableName</code> or is blank, usually the field is included in the XML update packet. If <code>ownerName</code> doesn’t have any of these values, this field is excluded from the update packet.
<code>isKey</code>	A Boolean value; if <code>true</code> , all key fields for the table are updated.

RDBMSResolver.nullValue

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`resolveData.nullValue`

Description

Property; a string used to provide a null value for a field’s value. You can customize this property to prevent it from being confused with an empty string ("") or another valid value. The default string is `{_NULL_}`.

RDBMSResolver.reconcileResults

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.reconcileResults(eventObject)
```

Parameters

eventObject Resolver event object; describes the event object used to compare two update packets. This event object should contain the following properties:

Property	Description
target	Object; the resolver broadcasting this event.
type	String; the name of the event.

Returns

Nothing.

Description

Event; broadcast by the RDBMSResolver component to compare two packets after results have been received from the server and applied to the delta packet.

A single `updateResults` packet can contain results of operations that were in the delta packet, as well as information about updates performed by other clients. When a new update packet is received, the operation results and database updates are split into two update packets and placed separately in the `deltaPacket` property. The `reconcileResults` event is broadcast just before the delta packet containing the operation results is sent using data binding.

Example

The following example reconciles two update packets and returns and clears the updates on success:

```
on (reconcileResults) {
    // Examine results.
    if (examine(updateResults)) {
        myDataSet.purgeUpdates();
    } else {
        displayErrors(results);
    }
}
```

RDBMSResolver.reconcileUpdates

Availability

Flash Player 7.

Edition

Flash Professional 8.

Usage

resolveData.reconcileUpdates(eventObject)

Parameters

eventObject Resolver event object; describes the customizations to the XML packet before the update is sent through the connector to the database. This event object should contain the following properties:

Property	Description
target	Object; the resolver broadcasting this event.
type	String; the name of the event.

Returns

None.

Description

Event; broadcast by the RDBMSResolver component when results have been received from the server after the updates from a delta packet were applied. A single `updateResults` packet can contain results of operations that were in the delta packet, as well as information about updates that were performed by other clients. When a new update packet is received, the operation results and database updates are split into two delta packets, which are placed separately in the `deltaPacket` property. The `reconcileUpdates` event is broadcast just before the delta packet containing any database updates is sent using data binding.

Example

The following example reconciles two results and clears the updates on success:

```
on (reconcileUpdates) {  
    // Examine results.  
    if (examine(updateResults)) {  
        myDataSet.purgeUpdates();  
    } else {  
        displayErrors(results);  
    }  
}
```

RDBMSResolver.tableName

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.tableName

Description

Property; a string that represents the table name in the XML for the database table to be updated. This property also determines which fields to send in the update packet. To make this determination, the RDBMSResolver component compares the value of this property with the value provided for the `fieldInfo.ownerName` property. If a field has no entry in the `fieldInfo` collection property, the field is placed in the update packet. If a field has an entry in the `fieldInfo` collection property, and its `ownerName` value is blank or identical to the RDBMSResolver component's `tableName` property, the field is placed in the update packet. If a field has an entry in the `fieldInfo` collection property, and its `ownerName` value is not blank and is different from the RDBMSResolver component's `tableName` property, the field is not placed in the update packet.

RDBMSResolver.updateMode

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

`resolveData.updateMode`

Description

Property; contains several values that determine how key fields are identified when the XML update packet is generated. This property can have the following strings as values:

Value	Description
"umUsingAll"	Uses the old values of all of the modified fields to identify the record to be updated. This is the safest value to use for updating, because it guarantees that another user has not modified any field of the record since you retrieved it. However, this approach is more time consuming and generates a larger update packet.

Value	Description
"umUsingModified"	Uses the old values of all of the fields modified to identify the record to be updated. This value guarantees that another user has not modified the same fields in the record since you retrieved it.
"umUsingKey"	The default value. This setting uses the old value of the key fields. This implies an "optimistic concurrency" model, which most database systems today employ, and guarantees that you are modifying the same record that you retrieved from the database. Your changes overwrite any other user's changes to the same data.

RDBMSResolver.updatePacket

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.updatePacket

Description

Property; property of type XML, containing an XML packet used to bind to a connector property that transmits the translated update packet of changes back to the server so the source of the data can be updated. This is an XML document containing the packet of DataSet changes.

RDBMSResolver.updateResults

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.updateResults

Description

Property; a delta packet that contains the results of an update returned from the server through a connector. Use this property to transmit errors and updated data from the server to a data set—for example, when the server assigns new IDs for an auto-assigned field. Bind this property to a connector's `results` property so that it can receive the results of an update and transmit the results back to the data set.

Messages in the `updateResults` property are treated as errors. This means that a delta with messages is added to the delta packet again so it can be re-sent the next time the delta packet is sent to the server. You must write code to handle deltas that have messages so that the messages are presented to the user and the deltas can be modified before being added to the next delta packet.

The `RectBorder` class is used as the border of most components. A separate implementation of this class is provided by each theme, which has its own set of border styles and properties that it supports.

You interact with the `RectBorder` class primarily by setting styles on other components. For example, when you include a `List` component in a document and set the `borderStyle` style property, the `List` component creates a `RectBorder` instance that uses the list's `borderStyle` setting. You can also create a custom `RectBorder` implementation to skin the border of all components that use `RectBorder`.

The `RectBorder` class has four standard display styles: `none`, `inset`, `outset`, and `solid`.

The Halo theme also adds four special display styles, which are used by specific components.

Special style	Component that uses it
<code>default</code>	<code>Window</code>
<code>alert</code>	<code>Alert</code>
<code>dropDown</code>	<code>ComboBox</code> and <code>DateField</code>
<code>menuBorder</code>	<code>Menu</code> and <code>MenuBar</code>

The `RectBorder` behavior and style properties described here are consistent for all components that use the `RectBorder` class.

Using styles with the RectBorder class

You can set style properties to change the appearance of a RectBorder instance. A RectBorder instance uses the following styles:

- borderCapColor
- borderColor
- buttonColor
- highlightColor
- shadowCapColor
- shadowColor
- themeColor

The styles available on a particular RectBorder instance depend on the theme in use and the border style set on the component. For an interactive demonstration that shows the relationship between theme, border style, and available color style properties, see [Using Components Help](#).

The four special Halo styles—default, alert, dropDown, and menuBorder—have some lines whose colors cannot be set through styles. You can modify these colors only by creating a custom theme and modifying the appropriate ActionScript within the custom RectBorder implementation.

To set a border style using `setStyle`:

1. Select **File > New** and create a new Flash document.
2. Drag a **TextArea** component to the Stage and give it the instance name `my_ta`.
3. In the first frame of the main timeline, add the following ActionScript to the Actions panel:

```
my_ta.setStyle("borderStyle", "alert");
```

NOTE

You can set the borderStyle to "alert" because you are using the default theme (Halo). If you are using a different theme, then the four "special" Halo styles, including "alert", may not be available.

4. Select **Control > Test Movie** to test the SWF file.

To set multiple border styles as parameters of the createClassObject method:

1. Select File > New and create a new Flash document.
2. In the first frame of the main timeline, add the following ActionScript to the Actions panel:

```
createClassObject(mx.controls.TextArea, "my_ta", 1, {borderStyle:  
"menuBorder", themeColor: "0x990000"});
```

For more information, see [UIObject.createClassObject\(\)](#). Or, if you want to set multiple styles and apply them to more than one component instance, you can establish a new style declaration containing the style settings, and then attach that style declaration to the component instances (see “Setting custom styles for groups of components” in *Using Components*).

3. Select Control > Test Movie to test the SWF file.

To set a border style using the Sample theme:

1. Select File > New and create a new Flash document.
2. Drag a Button component to the Stage, and give it the instance name **my_btn**.

You can also create the instance by using ActionScript, as follows (be sure to drag a Button component to the library first):

```
createClassObject(mx.controls.Button, "my_btn", 1);
```

3. Select File > Import > Open External Library.
4. Open the SampleTheme.fla file, located in:
 - Windows: \Program Files\Macromedia\Flash 8\language\Configuration\ComponentFLA\
 - Macintosh: HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/
5. In the SampleTheme.fla library, find the Button assets movie clip in Flash UI Components > Themes > MMDefault > Button Assets > Button Skin and drag it to the library of your current document.

6. In the first frame of the main timeline, add the following ActionScript to the Actions panel:

```
my_btn.setStyle("buttonColor", "0xFFFFFFFF");  
my_btn.setStyle("borderStyle", "solid");  
my_btn.setStyle("borderColor", "none");
```

NOTE

If you plan to set multiple styles and need to improve the performance of the component at runtime, you can set a custom style declaration containing those styles and then attach the custom style declaration to the component instance (see “Setting custom styles for groups of components” in *Using Components*).

Or you can append these settings to `createClassObject`, as follows:

```
createClassObject(mx.controls.Button, "my_btn", 1, {buttonColor:  
    "0xFFFFFFFF", borderStyle: "solid", borderColor: "none"});
```

7. Select Control > Test Movie to test the SWF file.

Notice that even with a "borderColor" of "none", the button has a gray border. In this case, "none" does not mean transparent, it means a neutral gray.

Creating a custom RectBorder implementation

The RectBorder class is used as a border skin in most ActionScript 2.0 components. The default implementations in both the Halo and Sample themes use ActionScript to draw the border. A custom implementation must use ActionScript to register itself as the RectBorder implementation and provide sizing functionality, but can use either ActionScript or graphic elements to represent the visuals.

Each RectBorder implementation must comply with the following requirements:

- It must extend `mx.skins.RectBorder` or one of its subclasses.
- It must provide an `offset` property value or implement the `getBorderMetrics` method to return sizing information.
- It must implement the `drawBorder()` method to draw or size the border.
- It must support all four standard styles, as well as the four special styles.
The implementation can reuse standard borders for special borders, as the Sample theme does.
- It must register itself as the RectBorder implementation.

RectBorder global registration

All components look to a central location for a reference to the RectBorder class in use for the document, `_global.styles.rectBorderClass`. You cannot specify that an individual component should use a different RectBorder implementation. To customize RectBorder for a component, you must rely on the `borderStyle` style property.

Custom RectBorder example

The RectBorder implementations provided by the Halo theme and the Sample theme use the ActionScript drawing API to draw the borders for different styles. The following example demonstrates how to create a custom RectBorder implementation that uses graphic symbols for its display.

To create a custom RectBorder implementation:

1. Create a new folder in the Classes/mx/skins folder corresponding to the custom package name that you will use for the custom border.

For this example, use myTheme.

2. Create a new AS file in the new folder and save it as RectBorder.as.
3. Copy the following ActionScript code to the new AS file:

```
import mx.core.ext.UIObjectExtensions;

class mx.skins.myTheme.RectBorder extends mx.skins.RectBorder
{
    static var symbolName:String = "RectBorder";
    static var symbolOwner:Object = RectBorder;
    var className:String = "RectBorder";

    #include "../..../core/ComponentVersion.as"

    // All of these borders have the same size edges, 1 pixel.
    var offset:Number = 4;

    function init(Void):Void
    {
        super.init();
    }

    function drawBorder(Void):Void
    {
        // The graphics are on the symbol's timeline,
        // so all you need to do here is size the border.
        __width = __width;
        __height = __height;
    }

    // Register the class as the RectBorder for all components to use.
    static function classConstruct():Boolean
    {
        UIObjectExtensions.Extensions();
        _global.styles.rectBorderClass = RectBorder;
        _global.skinRegistry["RectBorder"] = true;
        return true;
    }
    static var classConstructed:Boolean = classConstruct();
    static var UIObjectExtensionsDependency = UIObjectExtensions;
}
```

If you're not using the myTheme package, change the class declaration as needed.

4. Save the AS file.

5. Create a new FLA file.
 6. Use Insert > New Symbol to create a new movie clip symbol.
 7. Set the name to `RectBorder`.
 8. If the advanced fields are not displayed, click Advanced.
 9. Select Export for ActionScript
The identifier is automatically filled in as `RectBorder`.
 10. Set the AS 2.0 class to the full class name of the custom border implementation.
This example uses `mx.skins.myTheme.RectBorder`.
 11. Make sure that Export in First Frame is selected and then click OK.
 12. Open the `RectBorder` symbol for editing.
 13. Draw the graphics for the symbol.
For example, draw a hairline square with no fill. To make the custom border easy to see, set the line color to bright red.
 14. Make sure that the graphics are flush against the upper-left corner with the x and y coordinates set to $(0,0)$.
Your custom `drawBorder` implementation sets the width and height according to the component requirements.
 15. Click Back to return to the main timeline.
 16. Drag several components that use `RectBorder` to the Stage.
For example, drag a `List`, `TextArea`, and `TextInput` component to the Stage.
 17. Select Control > Test Movie.
- This example creates a very simple border implementation with static color and graphics. It doesn't respond to different `borderStyle` settings; it always uses the same graphics regardless of `borderStyle`. For examples of more complete border implementations, review the examples provided for the Halo and Sample themes.

Screen class (Flash Professional only)

The Screen class is the base class for screens that you create in the Screen Outline pane in Flash Professional 8. Screens are high-level containers for creating applications and presentations. For an overview of working with screens, see Chapter 14, “Working with Screens (Flash Professional Only),” in *Using Flash*.

The Screen class has two primary subclasses: Slide and Form.

The Slide class provides the runtime behavior for slide presentations. The Slide class provides built-in navigation and sequencing capabilities, and lets you easily attach transitions between slides using behaviors. Slide objects maintain “state,” and allow the user to advance to the next or previous slide/state: when the next slide is shown, the previous slide is hidden. For more information about using the Slide class to control slide presentations, see “[Slide class \(Flash Professional only\)](#)” on page 1135.

The Form class provides the runtime environment for form applications. Forms can overlay and contain, or be contained by, other components. Unlike slides, forms don’t provide any sequencing or navigation capabilities. For more information, see “[Form class \(Flash Professional only\)](#)” on page 735.

The Screen class provides functionality common to both slides and forms.

Screens know how to manage their children Every screen includes a built-in property that contains a list of that screen’s child screens, known as a collection. This collection is determined by the screen hierarchy in the Screen Outline pane. Screens can have any number of children (or none), which themselves can have children.

Screens can hide and show their children Because a screen is, essentially, a collection of nested movie clips, a screen can control the visibility of its children. For form applications, all of a screen’s children are visible by default at the same time; for slide presentations, individual screens are typically shown one at a time.

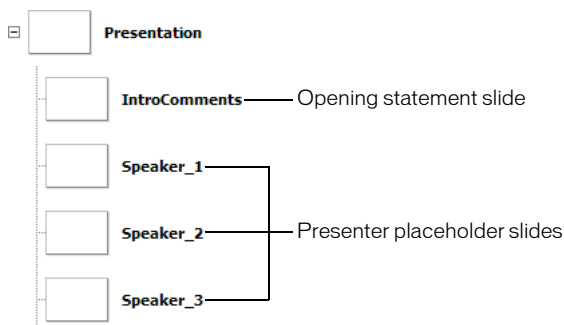
Screens broadcast events You can, for example, trigger a sound to play, or start playing some video, when a particular screen becomes visible.

Loading external content into screens (Flash Professional only)

The Screen class extends the Loader class (see “[Loader component](#)” on page 813), which lets you easily manage and load external SWF and JPEG files. The Loader class contains a `contentPath` property, which specifies the URL of an external SWF or JPEG file, or the linkage identifier of a movie clip in the library.

Using this feature, you can load an external screen tree (or any external SWF file) as a child of any screen node. This provides a useful way to make your screen-based media modular and divide it into separate SWF files.

For example, suppose you have a slide presentation in which three people are each contributing a single section. You could ask each presenter to create a separate slide presentation (SWF file). You would then create a “master slide presentation” that contains three placeholder slides, one for each slide presentation being created by the presenters. For each placeholder slide, you could point its `contentPath` property to one of the SWF files. The master slide presentation could be arranged as shown in the following illustration:



“Master” SWF file slide presentation structure

Suppose presenters provide you with three SWF files, `speaker_1.swf`, `speaker_2.swf`, and `speaker_3.swf`. You could easily assemble the overall presentation by setting the `contentPath` property of each placeholder slide, either using the Property inspector or ActionScript, as shown in the following code:

```
Speaker_1.contentPath = speaker_1.swf;
Speaker_2.contentPath = speaker_2.swf;
Speaker_3.contentPath = speaker_3.swf;
```

By default, when you set a slide's `contentPath` property while authoring in the Property inspector, or using ActionScript (as shown above), the specified SWF file loads as soon as the “master presentation” SWF file has loaded. To reduce initial load time, consider setting the `contentPath` property in an `on(reveal)` handler attached to each slide.

```
// Attached to Speaker_1 slide
on(reveal) {
    this.contentPath="speaker_1.swf";
}
```

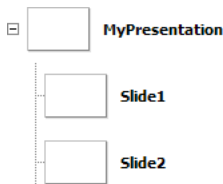
Alternatively, you could set the slide's `autoLoad` property to `false`. Then you could call the `load()` method on the slide when the slide has been revealed. (The `autoLoad` property and the `load()` method are inherited from the Loader class.)

```
// Attached to Speaker_1 slide
on(reveal) {
    this.load();
}
```

Referencing loaded screens with ActionScript

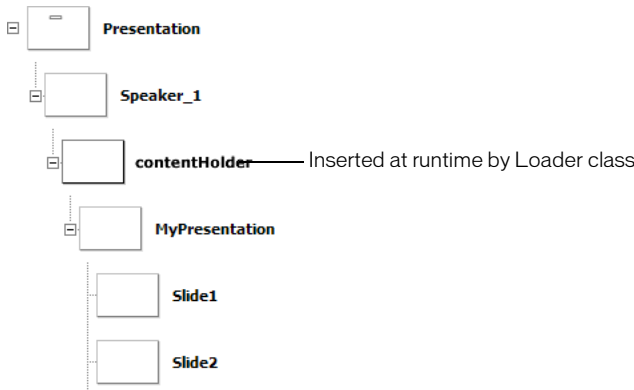
The Loader class creates an internal movie clip named `contentNode` into which it loads the SWF or JPEG file specified by the `contentPath` property. This movie clip, in effect, adds an extra screen node between the “placeholder” slide (that you created in the “master” presentation above) and the first slide in the loaded slide presentation.

For example, suppose the SWF file created for the `Speaker_1` slide placeholder (see above illustration) had the following structure, as shown in the Screen Outline pane:



“Speaker 1” SWF file slide presentation structure

At runtime, when the Speaker 1 SWF file is loaded into the placeholder slide, the overall slide presentation would have the following structure:



Structure of “master” and “speaker” presentation (runtime)

The properties and methods of the Screen, Slide, and Form classes “ignore” this contentHolder node as much as possible. That is, the slide named MyPresentation (along with its subslides) is part of the contiguous slide tree rooted at the Presentation slide, and is not treated as a separate subtree.

Screen class (API) (Flash Professional only)

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > View > [Loader component](#) > Screen

ActionScript Class Name mx.screens.Screen

The methods, properties, and events of the Screen class allow you to create and manipulate screens at runtime.

Method summary for the Screen class

The following table lists the method of the Screen class.

Method	Description
Screen.getChildScreen()	Returns the child screen of this screen at a particular index.

Methods inherited from the UIObject class

The following table lists the methods the Screen class inherits from the UIObject class. When calling these methods from the Screen object, use the form *ScreenInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Screen class inherits from the UIComponent class. When calling these methods from the Screen object, use the form *ScreenInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Methods inherited from the Loader class

The following table lists the method the Screen class inherits from the Loader class. When calling this method from the Screen object, use the form *ScreenInstance.methodName*.

Method	Description
Loader.load()	Loads the content specified by the <code>contentPath</code> property.

Property summary for the Screen class

The following table lists properties of the Screen class.

Property	Description
<code>Screen.currentFocusedScreen</code>	Read-only; returns the screen that contains the global current focus.
<code>Screen.indexInParent</code>	Read-only; returns the screen's index (zero-based) in its parent screen's list of child screens.
<code>Screen.numChildScreens</code>	Read-only; returns the number of child screens contained by the screen.
<code>Screen.parentIsScreen</code>	Read-only; returns a Boolean (<code>true</code> or <code>false</code>) value that indicates whether the screen's parent object is itself a screen.
<code>Screen.parentScreen</code>	Read-only; returns the screen that contains the specified screen.
<code>Screen.rootScreen</code>	Read-only; returns the root screen of the tree or subtree that contains the screen.

Properties inherited from the UIObject class

The following table lists the properties the Screen class inherits from the UIObject class.

When accessing these properties from the Screen object, use the form

ScreenInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.

Property	Description
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Screen class inherits from the UIComponent class. When accessing these properties from the Screen object, use the form `ScreenInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the Loader class

The following table lists the properties the Screen class inherits from the Loader class. When accessing these properties from the Screen object, use the form `ScreenInstance.propertyName`.

Property	Description
<code>Loader.autoLoad</code>	A Boolean value that indicates whether the content loads automatically (<code>true</code>) or you must call <code>Loader.load()</code> (<code>false</code>).
<code>Loader.bytesLoaded</code>	A read-only property that indicates the number of bytes that have been loaded.
<code>Loader.bytesTotal</code>	A read-only property that indicates the total number of bytes in the content.
<code>Loader.content</code>	A reference to the content of the loader. This property is read-only.
<code>Loader.contentPath</code>	A string that indicates the URL of the content to be loaded.

Property	Description
<code>Loader.percentLoaded</code>	A number that indicates the percentage of loaded content. This property is read-only.
<code>Loader.scaleContent</code>	A Boolean value that indicates whether the content scales to fit the loader (<code>true</code>), or the loader scales to fit the content (<code>false</code>).

Event summary for the Screen class

The following table lists events of the Screen class.

Event	Description
<code>Screen.allTransitionsInDone</code>	Broadcast when all “in” transitions applied to a screen have finished.
<code>Screen.allTransitionsOutDone</code>	Broadcast when all “out” transitions applied to a screen have finished.
<code>Screen.mouseDown</code>	Broadcast when the mouse button was pressed over an object (shape or movie clip) directly owned by the screen.
<code>Screen.mouseDownSomewhere</code>	Broadcast when the mouse button was pressed somewhere on the Stage, but not necessarily on an object owned by this screen.
<code>Screen.mouseMove</code>	Broadcast when the mouse is moved while over a screen.
<code>Screen.mouseOut</code>	Broadcast when the mouse is moved from inside the screen to outside it.
<code>Screen.mouseOver</code>	Broadcast when the mouse is moved from outside this screen to inside it.
<code>Screen.mouseUp</code>	Broadcast when the mouse button was released over an object (shape or movie clip) directly owned by the screen.
<code>Screen.mouseUpSomewhere</code>	Broadcast when the mouse button was released somewhere on the Stage, but not necessarily over an object owned by this screen.

Events inherited from the UIObject class

The following table lists the events the Screen class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Screen class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the Loader class

The following table lists the events the Screen class inherits from the Loader class.

Event	Description
<code>Loader.complete</code>	Triggered when the content finished loading.
<code>Loader.progress</code>	Triggered while content is loading.

Screen.allTransitionsInDone

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(allTransitionsInDone) {  
    // Your code here.  
}  
  
listenerObject = new Object();  
listenerObject.allTransitionsInDone = function(eventObject){  
    // Insert your code here.  
}  
  
screenObj.addEventListener("allTransitionsInDone", listenerObject)
```

Description

Event; broadcast when all “in” transitions applied to this screen have finished. The `allTransitionsInDone` event is broadcast by the Transition Manager associated with `screenObj`.

Example

In the following example, a button (`nextSlide_btn`) that's contained by the slide named `mySlide` is made visible when all the “in” transitions applied to `mySlide` have finished.

```
// Attached to mySlide:  
on(allTransitionsInDone) {  
    this.nextSlide_btn._visible = true;  
}
```

See also

[Screen.allTransitionsOutDone](#)

Screen.allTransitionsOutDone

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(allTransitionsOutDone) {  
    // Your code here.  
}  
listenerObject = new Object();  
listenerObject.allTransitionsOutDone = function(eventObject){  
    // Insert your code here.  
}  
screenObj.addEventListener("allTransitionsOutDone", listenerObject)
```

Description

Event; broadcast when all “out” transitions applied to the screen have finished. The `allTransitionsOutDone` event is broadcast by the Transition Manager associated with `screenObj`.

See also

[Screen.currentFocusedScreen](#)

Screen.currentFocusedScreen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myScreen.currentFocusedScreen
```

Description

Static property (read-only); returns a reference to the “leafmost” Screen object that contains the global current focus. *Leafmost* refers to the screen that is furthest away from the root screen in the screen hierarchy. The focus may be on the screen itself, or on a movie clip, text object, or component inside that screen. This property defaults to `null` if there is no current focus.

For example, assume you have a runtime screen hierarchy that looks like this:

```
presentation
  screen1
    subscreen1_1
      mymovieclip
        myUIButton

    screen2
      subscreen1_2
```

If `myUIButton` has focus, the leafmost screen containing the focus is `subscreen1_1`, which is what `currentFocusedScreen` would return. In this case, `presentation`, `screen1`, and `subscreen1_1` all contain the focus but the one that is “closest” (in the screen hierarchy) to the leaves of the tree (that is, farthest away from the root) is `subscreen1_1`.

Example

The following example displays the name of the currently focused screen in the Output panel.

```
var currentFocus:mx.screens.Screen =
    mx.screens.Screen.currentFocusedScreen;
trace("Current screen is: " + currentFocus._name);
```

Screen.getChildScreen()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myScreen.getChildScreen(childIndex)
```

Parameters

childIndex A number that indicates the zero-based index of the child screen to return.

Returns

A Screen object.

Description

Method; returns the child screen of *myScreen* whose index is *childIndex*.

Example

The following example sends the names of all the child screens belonging to the root screen named `Presentation` to the Output panel.

```
for (var i:Number = 0; i < _root.Presentation.numChildScreens; i++) {
    var childScreen:mx.screens.Screen =
        _root.Presentation.getChildScreen(i);
    trace(childScreen._name);
}
```

Screen.indexInParent

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myScreen.indexInParent

Description

Property (read-only); contains the zero-based index of *myScreen* in its parent's list of child screens.

Example

The following example displays the relative position of the screen *myScreen* in its parent screen's list of child screens.

```
var numChildren:Number = myScreen._parent.numChildScreens;
var myIndex:Number = myScreen.indexInParent;
trace("I'm child slide # " + myIndex + " out of " + numChildren + "
    screens.");
```

Screen.mouseDown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseDown) {  
    // Your code here.  
}  
  
listenerObject = new Object();  
listenerObject.mouseDown = function(eventObj){  
    // Insert your code here.  
}  
  
screenObj.addEventListener("mouseDown", listenerObject)
```

Description

Event; broadcast when the mouse button is pressed over an object (for example, a shape or a movie clip) directly owned by the screen.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Example

The following code displays the name of the screen that captured the mouse event in the Output panel.

```
on(mouseDown) {  
    trace("Mouse down event on: " + eventObj.target._name);  
}
```


Screen.mouseDownSomewhere

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseDownSomewhere) {  
    // Your code here.  
}  
  
listenerObject = new Object();  
listenerObject.mouseDownSomewhere = function(eventObject){  
    // Insert your code here.  
}  
  
screenObj.addEventListener("mouseDownSomewhere", listenerObject)
```

Description

Event; broadcast when the mouse button is pressed, but not necessarily over the specified screen.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Screen.mouseMove

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseMove) {  
    // Your code here.  
}  
  
listenerObject = new Object();  
listenerObject.mouseMove = function(eventObject){  
    // Insert your code here.  
}  
  
screenObj.addEventListener("mouseMove", listenerObject)
```

Description

Event; broadcast when the mouse moves while over the screen. This event is sent only when the mouse is over the bounding box of this screen.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

NOTE

This event may affect system performance and should be used judiciously.

Screen.mouseOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseOut) {  
    // Your code here.  
}  
  
listenerObject = new Object();  
listenerObject.mouseOut = function(eventObject){  
    // Insert your code here.  
}  
  
screenObj.addEventListener("mouseOut", listenerObject)
```

Description

Event; broadcast when the mouse moves from inside the screen's bounding box to outside its bounding box.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

NOTE

This event may affect system performance and should be used judiciously.

Screen.mouseOver

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseOver) {  
    // Your code here.  
}  
listenerObject = new Object();  
listenerObject.mouseOver = function(eventObject){  
    // Insert your code here.  
}  
screenObj.addEventListener("mouseOver", listenerObject)
```

Description

Event; broadcast when the mouse moves from outside the screen's bounding box to inside its bounding box.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

NOTE

This event may affect system performance and should be used judiciously.

Screen.mouseUp

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseUp) {  
    // Your code here.  
}  
listenerObject = new Object();  
listenerObject.mouseUp = function(eventObject){  
    // Insert your code here.  
}  
screenObj.addEventListener("mouseUp", listenerObject)
```

Description

Event; broadcast when the mouse is released over the screen.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Screen.mouseUpSomewhere

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(mouseUpSomewhere) {  
    // Your code here.  
}  
listenerObject = new Object();  
listenerObject.mouseUpSomewhere = function(eventObject){  
    // Insert your code here.  
}  
screenObj.addEventListener("mouseUpSomewhere", listenerObject)
```

Description

Event; broadcast when the mouse button is released, but not necessarily over the specified screen.

When the event is triggered, it automatically passes an event object (*eventObj*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

Screen.numChildScreens

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myScreen.numChildScreens

Description

Property (read-only); returns the number of child screens contained by *myScreen*.

Example

The following example displays the names of all the child screens that belong to *myScreen*.

```
var howManyKids:Number = myScreen.numChildScreens;
for(i=0; i<howManyKids; i++) {
    var childScreen = myScreen.getChildScreen(i);
    trace(childScreen._name);
}
```

See also

[Screen.getChildScreen\(\)](#)

Screen.parentIsScreen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

myScreen.parentIsScreen

Description

Property (read-only): returns a Boolean value indicating whether the specified screen's parent object is also a screen (*true*) or not (*false*). If this property is *false*, *myScreen* is at the root of its screen hierarchy.

Example

The following code determines if the parent object of the screen *myScreen* is also a screen. If *myScreen.parentIsScreen* is *true*, a `trace()` statement displays the number of sibling slides of *myScreen* in the Output panel. If the parent screen of *myScreen* is not also a screen, Flash assumes that *myScreen* is the root (master) slide in the presentation and therefore has no sibling slides.

```
if (myScreen.parentIsScreen) {  
    trace("I have "+myScreen._parent.numChildScreens+" sibling screens");  
} else {  
    trace("I am the root screen and have no siblings");  
}
```

Screen.parentScreen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myScreen.parentScreen
```

Description

Property (read-only); returns the screen that contains *myScreen*. Returns `null` if *myScreen* is the root screen.

Example

The following example displays the name of the screen that contains the screen *myScreen*.

```
var myParent:mx.screens.Screen = myScreen.rootScreen;
```

Screen.rootScreen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myScreen.rootScreen
```

Description

Property (read-only); returns the screen at the top of the screen hierarchy that contains *myScreen*.

Example

The following example displays the name of the root screen that contains the screen *myScreen*.

```
var myRoot:mx.screens.Screen = myScreen.rootScreen;
```


The ScrollPane component displays movie clips, JPEG files, and SWF files in a scrollable area. By using a scroll pane, you can limit the amount of screen area occupied by these media types. The scroll pane can display content that is loaded from a local disk or from the Internet. You can set this content while authoring and at runtime by using ActionScript.

Once the scroll pane has focus, if its content has valid tab stops, those markers receive focus. After the last tab stop in the content, focus shifts to the next component. The vertical and horizontal scroll bars in the scroll pane never receive focus.

A ScrollPane instance receives focus if a user clicks it or tabs to it. When a ScrollPane instance has focus, you can use the following keys to control it:

Key	Description
Down Arrow	Content moves up one vertical line scroll.
End	Content moves to the bottom of the scroll pane.
Left Arrow	Content moves right one horizontal line scroll.
Home	Content moves to the top of the scroll pane.
Page Down	Content moves up one vertical page scroll.
Page Up	Content moves down one vertical page scroll.
Right Arrow	Content moves left one horizontal line scroll.
Up Arrow	Content moves down one vertical line scroll.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each ScrollPane instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

Using the ScrollPane component

You can use a scroll pane to display any content that is too large for the area into which it is loaded. For example, if you have a large image and only a small space for it in an application, you could load it into a scroll pane.

You can set up a scroll pane to allow users to drag the content within the pane by setting the `scrollDrag` parameter to `true`; a pointing hand appears on the content. Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released. If the contents of a scroll pane have valid tab stops, you must set `scrollDrag` to `false`; otherwise each mouse interaction with the contents will invoke scroll dragging.

Components such as `Loader`, `ScrollPane`, and `Window` have events to determine when content has finished loading. So, if you want to set properties on the content of a `Loader`, `ScrollPane`, or `Window` component, add the property statement within a “complete” event handler. Consider the following example:

```
loadtest = new Object();
loadtest.complete = function(eventObject){
    content_mc._width= 100;
}
my_scrollpane.addEventListener("complete", loadtest)
```

For more information, see [“ScrollPane.content” on page 1104](#).

ScrollPane parameters

You can set the following authoring parameters for each `ScrollPane` instance in the Property inspector or in the Component inspector (`Window > Component Inspector` menu option):

contentPath indicates the content to load into the scroll pane. This value can be a relative path to a local SWF or JPEG file, or a relative or absolute path to a file on the Internet. It can also be the linkage identifier of a movie clip symbol in the library that is set to Export for ActionScript.

hLineScrollSize indicates the number of units a horizontal scroll bar moves each time an arrow button is clicked. The default value is 5.

hPageScrollSize indicates the number of units a horizontal scroll bar moves each time the track is clicked. The default value is 20.

hScrollPolicy displays the horizontal scroll bars. The value can be `on`, `off`, or `auto`. The default value is `auto`.

scrollDrag is a Boolean value that determines whether scrolling occurs (`true`) or not (`false`) when a user drags on the content within the scroll pane. The default value is `false`.

vLineScrollSize indicates the number of units a vertical scroll bar moves each time a scroll arrow is clicked. The default value is 5.

vPageScrollSize indicates the number of units a vertical scroll bar moves each time the scroll bar track is clicked. The default value is 20.

vScrollPolicy displays the vertical scroll bars. The value can be `on`, `off`, or `auto`. The default value is `auto`.

You can set the following additional parameters for each `ScrollPane` component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write `ActionScript` to control these and additional options for a `ScrollPane` component using its properties, methods, and events. For more information, see [“ScrollPane class” on page 1098](#).

Creating an application with the ScrollPane component

The following procedure explains how to add a `ScrollPane` component to an application while authoring. In this example, the scroll pane loads a picture from a path specified by the `contentPath` property.

To create an application with the ScrollPane component:

1. Drag the `ScrollPane` component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name `my_sp`.

3. Select Frame 1 in the main Timeline, open the Actions panel, and enter the following code:

```
/**
 * Requires:
 * - ScrollPane in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(320, 240);

// Create listener object for scroll vertical position.
var scrollListener:Object = new Object();
scrollListener.scroll = function(evt_obj:Object) {
    trace("hPosition: " + my_sp.hPosition + ", vPosition = " +
        my_sp.vPosition);
};
// Add listener.
my_sp.addEventListener("scroll", scrollListener);

// Create listener object for completed loading.
var completeListener:Object = new Object();
completeListener.complete = function(evt_obj:Object) {
    trace(evt_obj.target.contentPath + " has completed loading.");
};
// Add listener.
my_sp.addEventListener("complete", completeListener);

my_sp.contentPath = "http://www.helpexamples.com/flash/images/
    image1.jpg";
```

The examples creates a scroll pane, sets its size, and loads an image to it using the `contentPath` property. It also creates two listeners. The first one listens for a scroll event and displays the image's position as the user scrolls vertically or horizontally. The second one listens for a complete event and displays a message in the Output panel that says the image has completed loading.

Customizing the ScrollPane component

You can transform a ScrollPane component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the ScrollPane class.

Bear in mind these points about the ScrollPane component:

- The ScrollPane places the registration point of its content in the upper-left corner of the pane.
- When the horizontal scroll bar is turned off, the vertical scroll bar is displayed from top to bottom along the right side of the scroll pane. When the vertical scroll bar is turned off, the horizontal scroll bar is displayed from left to right along the bottom of the scroll pane. You can also turn off both scroll bars.
- If the scroll pane is too small, the content may not display correctly.
- When the scroll pane is resized, the buttons remain the same size. The scroll track and scroll box (thumb) expand or contract, and their hit areas are resized.

Using styles with the ScrollPane component

The ScrollPane supports the following styles:

Style	Theme	Description
themeColor	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
borderStyle	Both	The ScrollPane component uses a RectBorder instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063 . The default border style is "inset".
scrollTrackColor	Sample	The background color for the scroll track. The default value is OxCCCCCC (light gray).
symbolColor	Sample	The color of the arrows on the scrollbar buttons. The default value is Ox000000 (black).
symbolDisabledColor	Sample	The color of disabled arrows on the scrollbar buttons. The default value is Ox848384 (dark gray).

Using skins with the ScrollPane component

The ScrollPane component uses an instance of RectBorder for its border and scroll bars for scroll assets. For more information about skinning these visual elements, see [“RectBorder class” on page 1063](#) and [“Using skins with the UIScrollBar component” on page 1394](#).

ScrollPane class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > ScrollView > ScrollPane

ActionScript Class Name mx.containers.ScrollPane

The properties of the ScrollPane class let you do the following at runtime: set the content, monitor the loading progress, and adjust the scroll amount.

Setting a property of the ScrollPane class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

You can set up a scroll pane so that users can drag the content within the pane. To do this, set the `scrollDrag` property to `true`; a pointing hand appears on the content. Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released. If the contents of a scroll pane have valid tab stops, you must set `scrollDrag` to `false`; otherwise, each mouse interaction with the contents will invoke scroll dragging.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.ScrollPane.version);
```

NOTE

The code `trace(myScrollPaneInstance.version);` returns `undefined`.

Method summary for the ScrollPane class

The following table lists methods of the ScrollPane class.

Method	Description
ScrollPane.getBytesLoaded()	Returns the number of bytes of content loaded.
ScrollPane.getBytesTotal()	Returns the total number of bytes of content to be loaded.
ScrollPane.refreshPane()	Reloads the contents of the scroll pane (but does not redraw the scroll bar).

Methods inherited from the UIObject class

The following table lists the methods the ScrollPane class inherits from the UIObject class. When calling these methods from the ScrollPane object, use the form

ScrollPaneInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the ScrollPane class inherits from the UIComponent class. When calling these methods from the ScrollPane object, use the form

ScrollPaneInstance.methodName.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the ScrollPane class

The following table lists properties of the ScrollPane class.

Method	Description
<code>ScrollPane.content</code>	A reference to the content loaded into the scroll pane (read-only).
<code>ScrollPane.contentPath</code>	A string that indicates an absolute or relative URL of the SWF or JPEG file to load into the scroll pane, or that is the linkage identifier of a movie clip in the current document's library panel.
<code>ScrollPane.hLineScrollSize</code>	The amount of content to scroll horizontally when a scroll arrow is clicked.
<code>ScrollPane.hPageScrollSize</code>	The amount of content to scroll horizontally when the scroll track is clicked.
<code>ScrollPane.hPosition</code>	The horizontal pixel position of the scroll pane's horizontal scroll bar.
<code>ScrollPane.hScrollPolicy</code>	The status of the horizontal scroll bar. It can be always on ("on"), always off ("off"), or on when needed ("auto"). The default value is "auto".
<code>ScrollPane.scrollDrag</code>	Indicates whether scrolling occurs (<code>true</code>) or not (<code>false</code>) when a user drags on content within the scroll pane. The default value is <code>false</code> .
<code>ScrollPane.vLineScrollSize</code>	The amount of content to scroll vertically when a scroll arrow is clicked.
<code>ScrollPane.vPageScrollSize</code>	The amount of content to scroll vertically when the scroll track is clicked.
<code>ScrollPane.vPosition</code>	The pixel position of the scroll pane's vertical scroll bar.
<code>ScrollPane.vScrollPolicy</code>	The status of the vertical scroll bar. It can be always on ("on"), always off ("off"), or on when needed ("auto"). The default value is "auto".

Properties inherited from the UIObject class

The following table lists the properties the ScrollPane class inherits from the UIObject class. When accessing these properties from the ScrollPane object, use the form *ScrollPaneInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the ScrollPane class inherits from the UIComponent class. When accessing these properties from the ScrollPane object, use the form *ScrollPaneInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the ScrollPane class

The following table lists events of the ScrollPane class.

Event	Description
ScrollPane.complete	Broadcast when the scroll pane content is loaded.
ScrollPane.progress	Broadcast while the scroll pane content is loading.
ScrollPane.scroll	Broadcast when the scroll bar is clicked.

Events inherited from the UIObject class

The following table lists the events the ScrollPane class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the ScrollPane class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

ScrollPane.complete

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object) {
    // ...
};
ScrollPaneInstance.addEventListener("complete", listenerObject);
```

Usage 2:

```
on (complete) {
    //...
}
```

Description

Event; broadcast to all registered listeners when the content finishes loading.

The first usage example uses a dispatcher/listener event model. A component instance (*ScrollPaneInstance*) dispatches an event (in this case, *complete*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ScrollPane` instance `myScrollPaneComponent`, sends “_level0.myScrollPaneComponent” to the Output panel:

```
on (complete) {
    trace(this);
}
```

Example

The following example creates a listener object with a `complete` event handler for the `ScrollPane` instance. When the scroll pane's content is loaded, the listener displays a message in the Output panel.

You first drag the `ScrollPane` component from the Components panel to the library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - ScrollPane in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(320, 240);

// Create listener object for completed loading.
var completeListener:Object = new Object();
completeListener.complete = function(evt_obj:Object) {
    trace(evt_obj.target.contentPath + " has completed loading.");
};
// Add listener.
my_sp.addEventListener("complete", completeListener);

my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

ScrollPane.content

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.content

Description

Read-only property; a reference to the content of the scroll pane. The value is undefined until the load begins.

Example

This example sets the `contentPath` property to load a scroll pane with a picture (or technically, a movie clip containing a JPEG image). It also creates a numeric stepper that the user can increment or decrement by 10, up to a value of 100. When the user changes the value in the `NumericStepper`, a listener sets the transparency (`content._alpha`) of the image to the specified percentage. Note that `_alpha` is a `MovieClip` property.

You first drag `ScrollPane` and `NumericStepper` components from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - ScrollPane in library
 *   - NumericStepper in library
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.NumericStepper, "my_nstep", 10,
    {minimum:10, maximum:100, stepSize:10});
my_nstep.value = my_nstep.maximum;

this.createClassObject(mx.containers.ScrollPane, "my_sp", 20);
my_sp.move(0, 30);
my_sp.setSize(180, 160);
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image2.jpg";

var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
    my_sp.content._alpha = my_nstep.value;
}
my_nstep.addEventListener("change", nstepListener);
```

See also

[ScrollPane.contentPath](#)

ScrollPane.contentPath

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.contentPath

Description

Property; a string that indicates an absolute or relative URL of the SWF or JPEG file to load into the scroll pane. A relative path must be relative to the SWF file that loads the content.

If you load content using a relative URL, the loaded content must be relative to the location of the SWF file that contains the scroll pane. For example, an application using a ScrollPane component that resides in the directory `/scrollpane/nav/example.swf` could load contents from the directory `/scrollpane/content/flash/logo.swf` by using the following `contentPath` property: `"/content/flash/logo.swf"`

Example

The following example shows how to set the `contentPath` property to load a ScrollPane from three different sources: 1) an image on the Internet; 2) a movie clip in the library; 3) a SWF file from the current working directory. Use only one source at a time.

You first drag the ScrollPane component from the Components panel to the current document's library. To try option 2, you must create a movie clip in the library and reference its name. To try option 3, create a SWF file in the current working directory and specify its name. Then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane on Stage (instance name: my_sp)
 * - Symbol with Linkage ID of "movieClip_Name" in the library ** optional
 * - logo.swf file in the working directory ** optional
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_sp:mx.containers.ScrollPane;

// method 1: JPEG image
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

```
// method 2: Symbol in library
my_sp.contentPath = "movieClip_Name";

// method 3: SWF file
my_sp.contentPath = "logo.swf";
```

See also

[ScrollPane.content](#)

ScrollPane.getBytesLoaded()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.getBytesLoaded()

Parameters

None.

Returns

The number of bytes loaded in the scroll pane.

Description

Method; returns the number of bytes loaded in the ScrollPane instance. You can call this method at regular intervals while loading content to check its progress.

Example

This example creates a ScrollPane instance called `my_sp` and defines a listener object called `loadListener` with a `progress` event handler. The event handler calls the `getBytesLoaded()` and `getBytesTotal()` functions to display the progress of the load in the Output panel.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var loadListener:Object = new Object();
loadListener.progress = function(evt_obj:Object) {
    trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
        loaded.");
};
my_sp.addEventListener("progress", loadListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

ScrollPane.getBytesTotal()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.getBytesTotal()
```

Parameters

None.

Returns

A number.

Description

Method; returns the total number of bytes to be loaded into the ScrollPane instance.

Example

This example creates a `ScrollPane` instance called `my_sp` and defines a listener object called `loadListener` with a progress event handler. The event handler calls the `getBytesLoaded()` `getBytesTotal()` functions to display the progress of the load in the Output panel.

You first drag the `ScrollPane` component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var loadListener:Object = new Object();
loadListener.progress = function(evt_obj:Object) {
    trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
        loaded.");
};
my_sp.addEventListener("progress", loadListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

See also

[ScrollPane.getBytesLoaded\(\)](#)

ScrollPane.hLineScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.hLineScrollSize

Description

Property; the number of pixels to move the content when an arrow in the horizontal scroll bar is clicked. The default value is 5.

Example

This example creates a ScrollPane instance called `my_sp`, loads it with an image, and sets the `hLineScrollSize` property to scroll 100 pixels when the user clicks an arrow on the horizontal scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// Scroll 100 pixels when clicking on horizontal bar arrows.
my_sp.hLineScrollSize = 100;
```

ScrollPane.hPageScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollPaneInstance.hPageScrollSize

Description

Property; the number of pixels to move the content when the track in the horizontal scroll bar is clicked. The default value is 20.

Example

This example creates a ScrollPane instance called `my_sp`, loads it with an image, and sets the `hPageScrollSize` property to scroll 100 pixels when the user clicks the track in the horizontal scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's library and add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// Scroll 100 pixels when clicking on horizontal bar.
my_sp.hPageScrollSize = 100;
```

ScrollPane.hPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.hPosition

Description

Property; orients the scroll pane's contents in pixels, and adjusts the horizontal scroll box (thumb) proportionally. The 0 position is at the left end of the scroll track, which causes the left edge of the scroll pane content to be visible in the scroll pane.

Example

This example creates a ScrollPane instance called `my_sp`, loads it with an image, and creates a listener to handle the scroll event and display the horizontal (`hPosition`) and vertical (`vPosition`) scroll positions as the user clicks the scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// Scroll 100 pixels when clicking on horizontal bar.
my_sp.hPageScrollSize = 100;

// Create Listener Object.
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object) {
    trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
        my_sp.vPosition);
}
// Add listener.
my_sp.addEventListener("scroll", spListener);
```

ScrollPane.hScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.hScrollPolicy

Description

Property; determines whether the horizontal scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the image ("auto"). The default value is "auto".

Example

The following example creates an instance of a ScrollPane called `my_sp`, sets `hScrollPolicy` to `off` to prevent a horizontal scroll bar from appearing, and loads it with an image.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);
my_sp.hScrollPolicy = "off";

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

ScrollPane.progress

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object) {
    // ...
};
scrollPaneInstance.addEventListener("progress", listenerObject);
```

Usage 2:

```
on (progress) {
    // ...
}
```

Description

Event; broadcast to all registered listeners while content is loading. The progress event is not always broadcast; the complete event may be broadcast without any progress events being dispatched. This can happen especially if the loaded content is a local file. Your application triggers the progress event when the content starts loading by setting the value of the contentPath property.

The first usage example uses a dispatcher/listener event model. A component instance (*ScrollPaneInstance*) dispatches an event (in this case, *progress*) and the event is handled by a function, also called a *handler*, on a listener object (*ListenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*EventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “[EventDispatcher class](#)” on page 499.

The second usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ScrollPane` component instance `mySPComponent`, sends “`_level0.mySPComponent`” to the Output panel:

```
on (progress) {
    trace(this);
}
```

Example

This example creates a `ScrollPane` instance called `my_sp` and defines a listener object called `spListener` with a `progress` event handler. The event handler calls the `getBytesLoaded()` and `getBytesTotal()` functions to display the progress of the load in the Output panel.

You first drag the `ScrollPane` component from the Components panel to the current document’s library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var spListener:Object = new Object();
spListener.progress = function(evt_obj:Object):Void {
    trace("Loading " + my_sp.contentPath);
    trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
    loaded");
};
my_sp.addEventListener("progress", spListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

ScrollPane.refreshPane()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
ScrollPaneInstance.refreshPane()
```

Parameters

None.

Returns

Nothing.

Description

Method; refreshes the scroll pane after content is loaded. This method reloads the content, but does not reset the scroll bar. You could use this method if, for example, you've loaded a form into a scroll pane and an input property (for example, a text field) has been changed by ActionScript. In this case, you would call `refreshPane()` to reload the same form with the new values for the input properties.

Example

This example creates a Refresh button and a ScrollPane instance called `my_sp`. It loads the ScrollPane with an image and creates a listener for a click event on the button. When a click event occurs, the example calls the `refreshPane()` function, which reloads the content of the scroll pane.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.controls.Button, "my_button", 10,
    {label:"Refresh"});
this.createClassObject(mx.containers.ScrollPane, "my_sp", 20);
my_sp.move(0, 30);
my_sp.setSize(360, 280);

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    my_sp.refreshPane();
}
my_button.addEventListener("click", buttonListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

ScrollPane.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object):Void {
    // ...
};
scrollPaneInstance.addEventListener("scroll", listenerObject);
```

Usage 2:

```
on (scroll) {
    // ...
}
```


Event object

In addition to the standard event object properties, there are two additional properties defined for the `scroll` event: a `type` property whose value is "scroll", and a `direction` property whose value can be "vertical" or "horizontal".

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.progress` event: `current` (the loaded value equals `total`), and `total` (the total value).

Description

Event; broadcast to all registered listeners when a user clicks the scroll bar buttons, scroll box (thumb), or scroll track. Unlike other events, the `scroll` event is broadcast when a user presses the mouse button on the scroll bar and continues broadcasting until the button is released.

The first usage example uses a dispatcher/listener event model. A component instance (*ScrollPaneInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*ListenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*EventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `my_sp`, sends “_level0.my_sp” to the Output panel:

```
on (scroll) {  
    trace(this);  
}
```

Example

This example creates a `ScrollPane` instance called `my_sp`, loads it with an image, and creates a listener for the `scroll` event. When a scroll event occurs, the example displays the horizontal (`hPosition`) and vertical (`vPosition`) scroll positions in the Output panel.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// Scroll 100 pixels when clicking on horizontal bar.
my_sp.hPageScrollSize = 100;

// Create listener object.
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object):Void {
    trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
        my_sp.vPosition);
};
// Add listener.
my_sp.addEventListener("scroll", spListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

ScrollPane.scrollDrag

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.scrollDrag

Description

Property; a Boolean value that indicates whether scrolling occurs (`true`) or not (`false`) when a user drags within the scroll pane. The default value is `false`.

Example

This example creates a `ScrollPane` instance called `my_sp`, loads it with an image, and sets the `scrollDrag` property to `true`, allowing the user to scroll by dragging the image within the scroll pane.

You first drag the `ScrollPane` component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// Enable scrolling by dragging scroll pane.
my_sp.scrollDrag = true;
```

ScrollPane.vLineScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.vLineScrollSize

Description

Property; the number of pixels to move the content in the display area when the user clicks a scroll arrow in a vertical scroll bar. The default value is 5.

Example

The following example creates a `ScrollPane` instance called `my_sp`, loads it with an image, and sets the `vLineScrollSize` property to scroll 20 pixels when the user clicks an arrow on the vertical scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's panel and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// Scroll 20 pixels when clicking on vertical bar arrows.
my_sp.vLineScrollSize = 20;
```

ScrollPane.vPageScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.vPageScrollSize

Description

Property; the number of pixels to move the content in the display area when the user clicks the track in a vertical scroll bar. The default value is 20.

Example

This example creates a ScrollPane instance called `my_sp`, loads it with an image, and sets the `vPageScrollSize` property to scroll 30 pixels when the user clicks the track in the vertical scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// Scroll 30 pixels when clicking on vertical bar.
my_sp.vPageScrollSize = 30;
```

ScrollPane.vPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.vPosition

Description

Property; orients the scroll pane's contents in pixels, and adjusts the vertical scroll box (thumb) proportionally. The 0 position is at the top end of the scroll track, which causes the top edge of the scroll pane content to be visible in the scroll pane. The default value is 0.

Example

This example creates a ScrollPane instance called `my_sp`, loads it with an image, and creates a listener to handle the scroll event and display the horizontal (`hPosition`) and vertical (`vPosition`) scroll positions as the user clicks the scroll bar.

You first drag the ScrollPane component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 * - ScrollPane component in library
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/imagel.jpg";
// Scroll 100 pixels when clicking on horizontal bar.
my_sp.hPageScrollSize = 100;

// Create listener object.
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object):Void {
    trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
        my_sp.vPosition);
};
// Add listener.
my_sp.addEventListener("scroll", spListener);
```

ScrollPane.vScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.vScrollPolicy

Description

Property; determines whether the vertical scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the image ("auto"). The default value is "auto".

Example

The following example creates an instance of a `ScrollPane` called `my_sp`, sets `vScrollPolicy` to `off` to prevent a vertical scroll bar from appearing, and loads the `ScrollPane` with an image.

You first drag the `ScrollPane` component from the Components panel to the current document's library and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - ScrollPane component in library
 */
import mx.containers.ScrollPane;

this.createClassObject(ScrollPane, "my_sp", 30);
my_sp.setSize(360, 280);
my_sp.vScrollPolicy = "off";

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```


Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > SimpleButton

ActionScript Class Name mx.controls.SimpleButton

The properties of the SimpleButton class let you control the following at runtime:

- Whether a button has the emphasized look of a default push button
- Whether the button acts as a push button or as a toggle switch
- Whether a button is selected

Method summary for the SimpleButton class

There are no methods exclusive to the SimpleButton class.

Methods inherited from the UIObject class

The following table lists the methods the SimpleButton class inherits from the UIObject class.

When calling these methods from the SimpleButton class, use the form

buttonInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.

Method	Description
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the SimpleButton class inherits from the UIComponent class. When calling these methods from the SimpleButton object, use the form *buttonInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the SimpleButton class

The following table lists properties of the SimpleButton class.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the UIObject class

The following table lists the properties the SimpleButton class inherits from the UIObject class. When accessing these properties from the SimpleButton object, use the form *buttonInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the SimpleButton class inherits from the UIComponent class. When accessing these properties from the SimpleButton object, use the form *buttonInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the SimpleButton class

The following table lists the event of the SimpleButton class.

Event	Description
SimpleButton.click	Broadcast when a button is clicked.

Events inherited from the UIObject class

The following table lists the events the SimpleButton class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the SimpleButton class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

SimpleButton.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object){
    // ...
};
buttonInstance.addEventListener("click", listenerObject);
```

Usage 2:

```
on (click) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the button or if the button has focus and the Spacebar is pressed.

The first usage example uses a dispatcher/listener event model. A component instance (*buttonInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event occurs. When the event occurs, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see [EventDispatcher.addEventListener\(\)](#)) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `Button` component instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `Button` component instance `myButtonComponent`, sends “_level0.myButtonComponent” to the Output panel:

```
on (click) {
    trace(this);
}
```

The behavior of `this` is different when used inside an `on()` handler that is attached to a regular Flash button symbol; in this instance, `this` refers to the `MovieClip` that contains the button. For example, the following code, attached to the button symbol instance `myButton`, sends “_level0” to the Output panel:

```
on (release) {
    trace(this);
}
```

NOTE

The built-in ActionScript `Button` object doesn't have a `click` event; the closest event is `release`.

Example

This example, written on a frame of the timeline, sends a message to the Output panel when a button called `buttonInstance` is clicked. The first line specifies that the button act like a toggle switch. The second line creates a listener object called `form`. The third line defines a function for the `click` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function (in this example, `eventObj`) to generate a message. The `target` property of an event object is the component that generated the event (in this example, `buttonInstance`). The `SimpleButton.selected` property is accessed from the event object's `target` property. The last line calls `addEventListener()` from `buttonInstance` and passes it the `click` event and the `form` listener object as parameters.

```
buttonInstance.toggle = true;
var form:Object = new Object();
form.click = function(eventObj:Object) {
    trace("The selected property has changed to " +
        eventObj.target.selected);
};
buttonInstance.addEventListener("click", form);
```

The following code also sends a message to the Output panel when `buttonInstance` is clicked. The `on()` handler must be attached directly to `buttonInstance`.

```
on (click) {  
    trace("button component was clicked");  
}
```

SimpleButton.emphasized

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.emphasized

Description

Property; indicates whether the button is in an emphasized state (`true`) or not (`false`). The emphasized state is equivalent to the appearance of a default push button. In general, use the [FocusManager.defaultPushButton](#) property instead of setting the `emphasized` property directly. The default value is `false`.

If you aren't using `FocusManager.defaultPushButton`, you might just want to set a button to the emphasized state, or use the `emphasized` state to change text from one color to another. The following example sets the `emphasized` property for the button instance `myButton`:

```
_global.styles.foo = new CSSStyleDeclaration();  
_global.styles.foo.color = 0xFF0000;  
SimpleButton.emphasizedStyleDeclaration = "neutralStyle";  
myButton.emphasized = true;
```

See also

[SimpleButton.emphasizedStyleDeclaration](#)

SimpleButton.emphasizedStyleDeclaration

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.emphasizedStyleDeclaration

Description

Property (static); a string indicating the style declaration that formats a button when the `emphasized` property is set to `true`.

The `emphasizedStyleDeclaration` property is a static property of the `SimpleButton` class. Therefore, you must access it directly from `SimpleButton`, rather than from a *buttonInstance*, as in the following:

```
SimpleButton.emphasizedStyleDeclaration = "3dEmphStyle";
```

See also

[SimpleButton.emphasized](#)

SimpleButton.selected

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.selected

Description

Property; a Boolean value that indicates whether the button is selected (`true`) or not (`false`). The default value is `false`.

SimpleButton.toggle

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.toggle

Description

Property; a Boolean value that indicates whether the button acts as a toggle switch (*true*) or not (*false*). The default value is *false*.

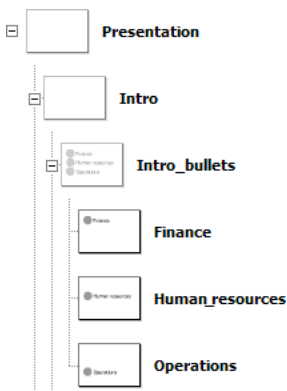
If a button acts as a toggle switch, it stays pressed until you click it again to release it.

Slide class (Flash Professional only)

The Slide class corresponds to a node in a hierarchical slide presentation. In Flash Professional 8, you can create slide presentations using the Screen Outline pane. For an overview of working with screens, see Chapter 14, “Working with Screens (Flash Professional Only),” in *Using Flash*.

The Slide class extends the Screen class (see “[Screen class \(Flash Professional only\)](#)” on page 1071), and provides built-in navigation and sequencing capabilities between slides, as well as the ability to easily attach transitions between slides using behaviors. Slides maintain “state,” so that when a user advances to an adjacent slide, the previous slide is hidden.

Note that you can only navigate to (“stop on”) slides that don’t contain any child slides (“leaf” slides). For example, consider the following illustration, which shows the contents of the Screen Outline pane for a sample slide presentation:



When this presentation starts, it will, by default, “stop” on the slide named Finance, which is the first slide in the presentation that doesn’t contain any child slides.

Also note that child slides “inherit” the visual appearance (graphics and other content) of their parent slides. For example, in the above illustration, in addition to the content on the Finance slide, the user would also see any content on the Intro and Presentation slides.

NOTE

The Slide class inherits from the [Loader class](#), which lets you easily load external SWF or JPEG files into a given slide. This provides a way to make your slide presentations modular and reduce initial download time. For more information, see “[Loading external content into screens \(Flash Professional only\)](#)” on page 1072.

Using the Slide class (Flash Professional only)

You use the methods and properties of the Slide class to control slide presentations you create using the Screen Outline pane for a Flash Slide Presentation, to get information about a slide presentation (for example, to determine the number of child slides contained by parent slide), or to navigate between slides in a slide presentation (for example, to create “Next slide” and “Previous slide” buttons).

You can also use the built-in behaviors that are available in the Behaviors panel to control slide presentations. For more information, see “Adding controls to screens using behaviors (Flash Professional only)” in *Using Flash*.

Slide parameters

You can set the following authoring parameters for each slide in the Property inspector or the Component inspector:

autoKeyNav determines how, or if, the slide responds to the default keyboard navigation. For more information, see [Slide.autoKeyNav](#).

autoload indicates whether the content specified by the `contentPath` parameter should load automatically (`true`), or wait to load until the [Loader.load\(\)](#) method is called (`false`). The default value is `true`.

contentPath specifies the contents of the slide. This can be the linkage identifier of a movie clip or an absolute or relative URL of a SWF or JPEG file to load into the slide. By default, loaded content is clipped to fit the slide.

overlayChildren specifies whether the slide’s child slides remain visible (`true`) or not (`false`) when you navigate from one child slide to the next.

playHidden specifies whether the slide continues to play (`true`) or not (`false`) when hidden.

Using the Slide class to create a slide presentation

You use the methods and properties of the Slide class to control slide presentations you create in the Screen Outline pane for a Flash Slide Presentation in the Flash authoring environment. (The Behaviors panel also contains several behaviors for creating slide navigation.) In this example, you write your own ActionScript to create Next and Previous buttons for a slide presentation.

To create a slide presentation with navigation:

1. In Flash, select File > New.
2. On the General tab, select Flash Slide Presentation.
3. In the Screen Outline pane, click the Insert Screen (+) button twice to create two new slides beneath the Presentation slide.

The Screen Outline pane should look like the following:



4. Select Slide1 in the Screen Outline pane and, using the Text tool, add a text field that reads **This is slide one**.
5. Repeat the previous step for Slide2 and Slide3, creating text fields on each slide that read **This is slide two** and **This is slide three**, respectively.
6. Select the Presentation slide and open the Components panel.
7. Drag a Button component from the Components panel to the bottom of the Stage.
8. In the Property inspector, type **Next Slide** for the Button component's Label property.
9. In the Actions panel, type the following code:

```
on(click) {  
    _parent.currentSlide.gotoNextSlide();  
}
```
10. Test the SWF file (Control > Test Movie) and click the Next Slide button to advance to the next slide.

Slide class (API) (Flash Professional only)

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [View](#) > [Loader component](#) > [Screen class \(Flash Professional only\)](#) > [Slide](#)

ActionScript Class Name `mx.screens.Slide`

The methods, properties, and events of the Slide class allow you to manage and manipulate slides.

Method summary for the Slide class

The following table lists methods of the Slide class:

Method	Description
Slide.getChildSlide()	Returns the specified child slide.
Slide.gotoFirstSlide()	Navigates to the first leaf slide in the slide's hierarchy of subslides.
Slide.gotoLastSlide()	Navigates to the last leaf slide in the slide's hierarchy of subslides.
Slide.gotoNextSlide()	Navigates to the next slide.
Slide.gotoPreviousSlide()	Navigates to the previous slide.
Slide.gotoSlide()	Navigates to an specified slide.

Methods inherited from the UIObject class

The following table lists the methods the Slide class inherits from the UIObject class. When calling these methods from the Slide object, use the form *SlideInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.

Method	Description
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Slide class inherits from the UIComponent class. When calling these methods from the Slide object, use the form *SlideInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the Loader class

The following table lists the method the Slide class inherits from the Loader class. When calling this method from the Slide object, use the form *SlideInstance.methodName*.

Method	Description
<code>Loader.load()</code>	Loads the content specified by the <code>contentPath</code> property.

Methods inherited from the Screen class

The following table lists the method the Slide class inherits from the Screen class. When calling this method from the Slide object, use the form *SlideInstance.methodName*.

Method	Description
<code>Screen.getChildScreen()</code>	Returns the child screen of this screen at a particular index.

Property summary for the Slide class

The following table lists properties of the Slide class:

Property	Description
<code>Slide.autoKeyNav</code>	Determines whether the slide uses default keyboard handling to navigate to the next/previous slide.
<code>Slide.currentChildSlide</code>	Read-only; returns the immediate child of the specified slide that contains the currently active slide.
<code>Slide.currentFocusedSlide</code>	Read-only; returns the “leafmost” slide (the slide farthest from the root of the slide tree) that contains the global current focus.
<code>Slide.currentSlide</code>	Read-only; returns the currently active slide.
<code>Slide.defaultKeyDownHandler</code>	Callback function that overrides the default keyboard navigation (Left and Right Arrow keys).
<code>Slide.firstSlide</code>	Read-only; returns the slide’s first child slide that has no children.
<code>Slide.indexInParentSlide</code>	Read-only; returns the slide’s index (zero-based) in its parent’s list of subslides.
<code>Slide.lastSlide</code>	Read-only; returns the slide’s last child slide that has no children.
<code>Slide.nextSlide</code>	Read-only; returns the slide you would reach if you called <code>mySlide.gotoNextSlide()</code> , but does not actually navigate to that slide.
<code>Slide.numChildSlides</code>	Read-only; returns the number of child slides the slide contains.
<code>Slide.overlayChildren</code>	Determines whether the slide’s child slides are visible when control flows from one child slide to the next.
<code>Slide.parentIsSlide</code>	Read-only; returns a Boolean value indicating whether the parent object of the slide is also a slide (<code>true</code>) or not (<code>false</code>).
<code>Slide.parentSlide</code>	Read-only; slide containing the current slide. May be <code>null</code> for the root slide.
<code>Slide.playHidden</code>	Determines whether the slide continues to play when hidden.
<code>Slide.previousSlide</code>	Read-only; returns the slide you would reach if you called <code>mySlide.gotoPreviousSlide()</code> , but does not actually navigate to that slide.
<code>Slide.rootSlide</code>	Read-only; returns the root of the slide tree that contains the slide.

Properties inherited from the UIObject class

The following table lists the properties the Slide class inherits from the UIObject class. When accessing these properties from the Slide object, use the form

SlideInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Slide class inherits from the UIComponent class.

When accessing these properties from the Slide object, use the form

SlideInstance.propertyName.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the Loader class

The following table lists the properties the Slide class inherits from the Loader class. When accessing these properties from the Slide object, use the form

SlideInstance.propertyName.

Property	Description
<code>Loader.autoLoad</code>	A Boolean value that indicates whether the content loads automatically (<code>true</code>) or you must call <code>Loader.load()</code> (<code>false</code>).
<code>Loader.bytesLoaded</code>	A read-only property that indicates the number of bytes that have been loaded.
<code>Loader.bytesTotal</code>	A read-only property that indicates the total number of bytes in the content.
<code>Loader.content</code>	A reference to the content of the loader. This property is read-only.
<code>Loader.contentPath</code>	A string that indicates the URL of the content to be loaded.
<code>Loader.percentLoaded</code>	A number that indicates the percentage of loaded content. This property is read-only.
<code>Loader.scaleContent</code>	A Boolean value that indicates whether the content scales to fit the loader (<code>true</code>), or the loader scales to fit the content (<code>false</code>).

Properties inherited from the Screen class

The following table lists the properties the Slide class inherits from the Screen class. When accessing these properties from the Slide object, use the form

SlideInstance.propertyName.

Property	Description
<code>Screen.currentFocusedScreen</code>	Read-only; returns the screen that contains the global current focus.
<code>Screen.indexInParent</code>	Read-only; returns the screen's index (zero-based) in its parent screen's list of child screens.
<code>Screen.numChildScreens</code>	Read-only; returns the number of child screens contained by the screen.

Property	Description
<code>Screen.parentIsScreen</code>	Read-only; returns a Boolean (<code>true</code> or <code>false</code>) value that indicates whether the screen's parent object is itself a screen.
<code>Screen.rootScreen</code>	Read-only; returns the root screen of the tree or subtree that contains the screen.

Event summary for the Slide class

The following table lists events of the Slide class.

Event	Description
<code>Slide.hideChild</code>	Broadcast each time a child of a slide changes from visible to invisible.
<code>Slide.revealChild</code>	Broadcast each time a child slide of a slide object changes from invisible to visible.

Events inherited from the UIObject class

The following table lists the events the Slide class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Slide class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the Loader class

The following table lists the events the Slide class inherits from the Loader class.

Event	Description
<code>Loader.complete</code>	Triggered when the content finished loading.
<code>Loader.progress</code>	Triggered while content is loading.

Events inherited from the Screen class

The following table lists the events the Slide class inherits from the Screen class.

Event	Description
<code>Screen.allTransitionsInDone</code>	Broadcast when all “in” transitions applied to a screen have finished.
<code>Screen.allTransitionsOutDone</code>	Broadcast when all “out” transitions applied to a screen have finished.
<code>Screen.mouseDown</code>	Broadcast when the mouse button was pressed over an object (shape or movie clip) directly owned by the screen.
<code>Screen.mouseDownSomewhere</code>	Broadcast when the mouse button was pressed somewhere on the Stage, but not necessarily on an object owned by this screen.
<code>Screen.mouseMove</code>	Broadcast when the mouse is moved while over a screen.
<code>Screen.mouseOut</code>	Broadcast when the mouse is moved from inside the screen to outside it.
<code>Screen.mouseOver</code>	Broadcast when the mouse is moved from outside this screen to inside it.

Event	Description
Screen.mouseUp	Broadcast when the mouse button was released over an object (shape or movie clip) directly owned by the screen.
Screen.mouseUpSomewhere	Broadcast when the mouse button was released somewhere on the Stage, but not necessarily over an object owned by this screen.

Slide.autoKeyNav

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.autoKeyNav

Description

Property; determines whether the slide uses default keyboard handling to navigate to the next/previous slide when *mySlide* has focus. This property accepts the string values "true", "false", and "inherit". You can override this default keyboard handling behavior by using the [Slide.defaultKeyDownHandler](#) property.

When the value of this property is "true", pressing the Right Arrow key (`Key.RIGHT`) or the Spacebar (`Key.SPACE`) when *mySlide* has focus advances to the next slide; pressing the Left Arrow key (`Key.Left`) moves to the previous slide.

When this property is set to "false", no default keyboard handling takes place when *mySlide* has focus.

When this property is set to "inherit", *mySlide* checks the `autoKeyNav` property of its parent slide. If it is also set to "inherit", Flash looks up the slide inheritance chain until it finds a parent slide whose `autoKeyNav` property is set to "true" or "false".

If *mySlide* has no parent slide (that is, if the statement `(mySlide.parentIsSlide == false)` is true), it behaves as if `autoKeyNav` had been set to "true".

Example

This example turns off automatic keyboard navigation for the slide named `loginSlide`.

```
_root.Presentation.loginSlide.autoKeyNav = "false";
```

See also

[Slide.defaultKeydownHandler](#)

Slide.currentChildSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.currentChildSlide
```

Description

Property (read-only); returns the immediate child of *mySlide* that contains the currently active slide; returns `null` if no child slide contained by *mySlide* has the current focus.

Example

Consider the following screen outline:

```
Presentation
  Slide_1
    Bullet1_1
      SubBullet1_1_1
    Bullet1_2
      SubBullet1_2_1
  Slide_2
```

Assuming that `SubBullet1_1_1` is the current slide, then the following statements are all true:

```
Presentation.currentChildSlide == Slide_1;
Slide_1.currentChildSlide == Bullet_1_1;
SubBullet_1_1_1.currentChildSlide == null;
Slide_2.currentChildSlide == null;
```

See also

[Slide.currentSlide](#)

Slide.currentFocusedSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mx.screens.Slide.currentFocusedSlide
```

Description

Property (read-only); returns the “leafmost” slide (the slide farthest from the root of the slide tree) that contains the current global focus. The actual focus may be on the slide itself, or on a movie clip, text object, or component inside that slide; the method returns `null` if there is no current focus.

Example

```
var focusedSlide = mx.screens.Slide.currentFocusedSlide;
```

Slide.currentSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.currentSlide
```

Description

Property (read-only); returns the currently active slide. This is always a “leaf” slide—that is, a slide that contains no child slides.

Example

The following code, attached to a button on the root presentation slide, advances the slide presentation to the next slide each time the button is clicked.

```
// Attached to button instance contained by presentation slide:
on(press) {
    _parent.currentSlide.gotoNextSlide();
}
```

See also

[Slide.gotoNextSlide\(\)](#)

Slide.defaultKeyDownHandler

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.defaultKeyDownHandler = function (eventObj) {
    // Your code here.
}
```

Parameters

eventObj An event object with the following properties:

- *type* A string indicating the type of event. Possible values are "keyUp" and "keyDown".
- *ascii* An integer that represents the ASCII value of the last key pressed; corresponds to the value returned by `Key.getAscii()`.
- *code* An integer that represents the key code of the last key pressed; corresponds to the value returned by `Key.getCode()`.
- *shiftKey* A Boolean value indicating if the Shift key is currently being pressed (`true`) or not (`false`).
- *ctrlKey* A Boolean value indicating if the Control key is currently being pressed (`true`) or not (`false`).

Returns

Nothing.

Description

Callback function; lets you override the default keyboard navigation with a custom keyboard handler that you create. For example, instead of having the Left and Right Arrow keys navigate to the previous and next slides in a presentation, respectively, you could have the Up and Down Arrow keys perform those functions. For a discussion of the default keyboard handling behavior, see [Slide.autoKeyNav](#).

Example

In that example, the default keyboard handling is altered for child slides of the slide to which the `on(load)` handler is attached. This handler uses the Up and Down Arrow keys for navigation instead of the Left and Right Arrow keys.

```
on (load) {
  this.defaultKeyDownHandler = function(eventObj:Object) {
    switch (eventObj.code) {
      case Key.DOWN :
        this.currentSlide.gotoNextSlide();
        break;
      case Key.UP :
        this.currentSlide.gotoPreviousSlide();
        break;
      default :
        break;
    }
  };
}
```

See also

[Slide.autoKeyNav](#)

Slide.firstSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.firstSlide

Description

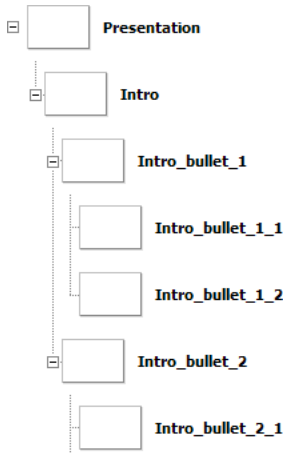
Property (read-only); returns the first child slide of *mySlide* that has no child slides.

Example

In the hierarchy of slides shown below, the following statements are both true:

```
Presentation.Intro.firstSlide == Intro_bullet_1_1;
```

```
Presentation.Intro_bullet_1.firstSlide == Intro_bullet_1_1;
```



Slide.getChildSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.getChildSlide(childIndex)
```

Parameters

childIndex The zero-based index of the child slide to return.

Returns

A slide object.

Description

Method; returns the child slide of *mySlide* whose index is *childIndex*. You can use this method to iterate over a set of child slides whose indices are known.

Example

The following code causes the Output panel to display the names of all the child slides of the root presentation slide.

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
    var childSlide = _root.Presentation.getChildSlide(slideIndex);
    trace(childSlide._name);
}
```

See also

[Slide.numChildSlides](#)

Slide.gotoFirstSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.gotoFirstSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; navigates to the first leaf slide in the tree of child slides beneath *mySlide*. This method is ignored when called from within a slide's `on(hide)` or `on(reveal)` event handler if that event was a result of a slide navigation.

To go to the first slide in a presentation, call `mySlide.rootSlide.gotoFirstSlide()`. (For more information on `rootSlide`, see [Slide.revealChild](#).)

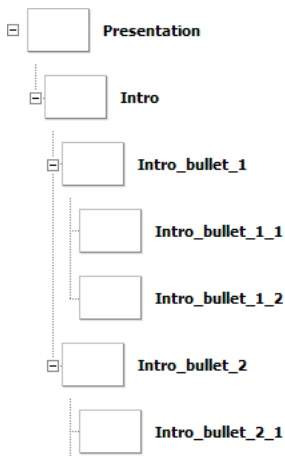
Example

In the slide hierarchy illustrated below, the following method calls would all navigate to the slide named `Intro_bullet_1_1`:

```
Presentation.gotoFirstSlide();  
Presentation.Intro.gotoFirstSlide();  
Presentation.Intro.Intro_bullet_1.gotoFirstSlide();
```

This method call would navigate to the slide named `Intro_bullet_2_1`:

```
Presentation.Intro.Intro_bullet_2.gotoFirstSlide();
```



See also

[Slide.firstSlide](#), [Slide.revealChild](#)

Slide.gotoLastSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.gotoLastSlide()
```

Parameters

None.

Returns

Nothing.

Description

Method; navigates to the last leaf slide in the tree of child slides beneath *mySlide*. This method is ignored when called from within a slide's `on(hide)` or `on(reveal)` event handler if that event was a result of another slide navigation.

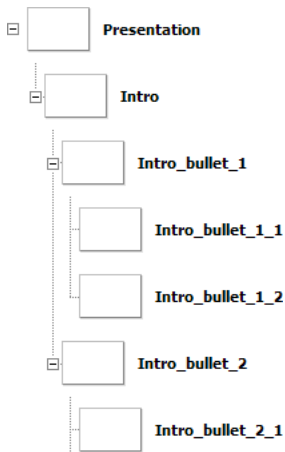
Example

In the slide hierarchy illustrated below, the following method calls would navigate to the slide named `Intro_bullet_1_2`:

```
Presentation.Intro.gotoLastSlide();  
Presentation.Intro.Intro_bullet_1.gotoLastSlide();
```

These method calls would navigate to the slide named `Intro_bullet_2_1`:

```
Presentation.gotoLastSlide();  
Presentation.Intro.gotoLastSlide();
```



See also

[Slide.gotoSlide\(\)](#), [Slide.lastSlide](#)

Slide.gotoNextSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.gotoNextSlide()
```

Parameters

None.

Returns

A Boolean value, or `null`. The method returns `true` if it successfully navigated to the next slide; it returns `false` if the presentation is already at the last slide when the method is invoked (that is, if `currentSlide.nextSlide` is `null`). The method returns `null` if invoked on a slide that doesn't contain the current slide.

Description

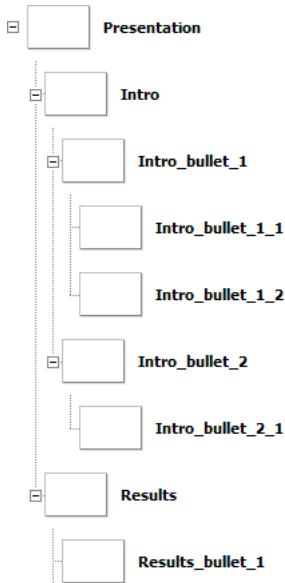
Method; navigates to the next slide in the slide presentation. As control passes from one slide to the next, the outgoing slide is hidden and the incoming slide is revealed. If the outgoing and incoming slides are in different slide subtrees, then all ancestor slides, starting with the outgoing slide and up to the common ancestor of the incoming and outgoing slides, are hidden and receive a `hide` event. Immediately following, all ancestor slides of the incoming slide, up to the common ancestor of the outgoing and incoming slide, are made visible and receive a `reveal` event.

Typically, `gotoNextSlide()` is called on the leaf node that represents the current slide. If called on a nonleaf node, `someNode`, then `someNode.gotoNextSlide()` advances to the first leaf node in the next slide or "section."

This method has no effect when invoked on a slide that does not contain the current slide. This method also has no effect when called from within an `on(hide)` or `on(reveal)` event handler attached to a slide, if that handler was invoked as a result of slide navigation.

Example

Suppose that, in the following slide hierarchy, the slide named `Intro_bullet_1_1` is the current slide being viewed (that is, `_root.Presentation.currentSlide._name == Intro_bullet_1_1`).



In this case, calling `Intro_bullet_1_1.gotoNextSlide()` would navigate to `Intro_bullet_1_2`, which is a sibling slide of `Intro_bullet_1_1`.

However, calling `Intro_bullet_1.gotoNextSlide()` would navigate to `Intro_bullet_2_1`, the first leaf slide contained by `Intro_bullet_2`, which is the next sibling slide of `Intro_bullet_1`. Similarly, calling `Intro.gotoNextSlide()` would navigate to `Results_bullet_1`, the first leaf slide contained by the `Results` slide.

Also, still assuming that the current slide is `Intro_bullet_1_1`, calling `Results.gotoNextSlide()` would have no effect, because `Results` does not contain the current slide (that is, `Results.currentSlide` is `null`).

See also

[Slide.currentSlide](#), [Slide.gotoPreviousSlide\(\)](#), [Slide.nextSlide](#)

Slide.gotoPreviousSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.gotoPreviousSlide()
```

Parameters

None.

Returns

A Boolean value, or `null`. The method returns `true` if it successfully navigated to the previous slide; it returns `false` if the presentation is at the first slide when the method is invoked (that is, if `currentSlide.nextSlide` is `null`). The method returns `null` if invoked on a slide that doesn't contain the current slide.

Description

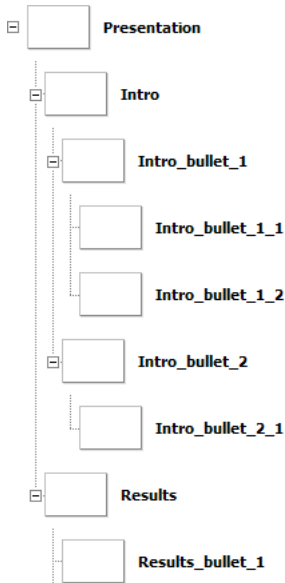
Method; navigates to the previous slide in the slide presentation. As control passes from one slide to the previous slide, the outgoing slide is hidden and the incoming slide is revealed. If the outgoing and incoming slides are in different slide subtrees, then all ancestor slides, starting with the outgoing slide and up to the common ancestor of the incoming and outgoing slides, are hidden and receive a `hide` event. Immediately following, all ancestors slides of the incoming slide, up to the common ancestor of the outgoing and incoming slide, are made visible and receive a `reveal` event.

Typically, `gotoPreviousSlide()` is called on the leaf node that represents the current slide. If called on a nonleaf node, `someNode`, then `someNode.gotoPreviousSlide()` advances to the first leaf node in the previous slide or "section."

This method has no effect when invoked on a slide that does not contain the current slide. This method also has no effect when called from within an `on(hide)` or `on(reveal)` event handler attached to a slide, if that handler was invoked as a result of slide navigation.

Example

Suppose that, in the following slide hierarchy, the slide named `Intro_bullet_1_2` is the current slide being viewed (that is, `_root.Presentation.currentSlide._name == Intro_bullet_1_2`).



In this case, calling `Intro_bullet_1_2.gotoPreviousSlide()` would navigate to `Intro_bullet_1_1`, which is the previous sibling slide of `Intro_bullet_1_2`.

However, calling `Intro_bullet_2.gotoPreviousSlide()` would navigate to `Intro_bullet_1_1`, the first leaf slide contained by `Intro_bullet_1`, which is the previous sibling slide of `Intro_bullet_2`. Similarly, calling `Results.gotoPreviousSlide()` would navigate to `Intro_bullet_1_1`, the first leaf slide contained by the `Intro` slide.

Also, if the current slide is `Intro_bullet_1_1`, then calling `Results.gotoPreviousSlide()` would have no effect, since `Results` does not contain the current slide (that is, `Results.currentSlide` is null).

See also

[Slide.currentSlide](#), [Slide.gotoNextSlide\(\)](#), [Slide.previousSlide](#)

Slide.gotoSlide()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
mySlide.gotoSlide(newSlide)
```

Parameters

newSlide The slide to navigate to.

Returns

A Boolean value indicating if the navigation succeeded (`true`) or not (`false`).

Description

Method; navigates to the slide specified by *newSlide*. For the navigation to succeed, the following must be true:

- The current slide must be a child slide of *mySlide*.
- The slide specified by *newSlide* and the current slide must share a common ancestor slide—that is, the current slide and *newSlide* must reside in the same slide subtree.

If either of these conditions isn't met, the navigation fails and the method returns `false`; otherwise, the method navigates to the specified slide and returns `true`.

For example, consider the following slide hierarchy:

```
Presentation
  Slide1
    Slide1_1
    Slide1_2
  Slide2
    Slide2_1
    Slide2_2
```

If the current slide is `Slide1_2`, the following `gotoSlide()` call fails, because the current slide is not a descendant of `Slide2`:

```
Slide2.gotoSlide(Slide2_1);
```

Also consider the following screen hierarchy, where a form object is the parent screen of two separate slide trees:

```
Form_1
  Slide1
    Slide1_1
    Slide1_2
  Slide2
    Slide2_1
    Slide2_2
```

If the current slide is `Slide1_2`, the following method call also fails, because `Slide1` and `Slide2` are in different slide subtrees:

```
Slide1_2.gotoSlide(Slide2_2);
```

Example

The following code, attached to a `Button` component, uses the `Slide.currentSlide` property and the `gotoSlide()` method to display the next slide in the presentation.

```
on(click) {
  _parent.gotoSlide(_parent.currentSlide.nextSlide);
}
```

This is equivalent to the following code, which uses the `Slide.gotoNextSlide()` method:

```
on(click) {
  _parent.currentSlide.gotoNextSlide();
}
```

See also

[Slide.currentSlide](#), [Slide.gotoNextSlide\(\)](#)

Slide.hideChild

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(hideChild) {
  // Your code here.
}
```

Description

Event; broadcast each time a child of a slide changes from visible to invisible. This event is broadcast only by slides, not forms. The main use of the `hideChild` event is to apply “out” transitions to all the children of a slide.

Example

When attached to the root slide (for example, the presentation slide), this code displays the name of each child slide that belongs to the root slide, as the child slide is hidden.

```
on(hideChild) {  
    var child = eventObj.target._name;  
    trace(child + " has just been hidden");  
}
```

See also

[Slide.revealChild](#)

Slide.indexInParentSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.indexInParent

Description

Property (read-only); returns the zero-based index of *mySlide* in its parent’s list of child slides.

Example

The following code uses the `indexInParentSlide` and `Slide.numChildSlides` properties to display the index of the current slide being viewed and the total number of slides contained by its parent slide. To use this code, attach it to a parent slide that contains one or more child slides.

```
on (revealChild) {  
    trace("Displaying "+(currentSlide.indexInParentSlide+1)+" of  
        "+currentSlide._parent.numChildSlides);  
}
```

Note that because this property is a zero-based index, its value is incremented by 1 (`currentSlide.indexInParent+1`) to display more meaningful values.

See also

[Slide.numChildSlides](#), [Slide.revealChild](#)

Slide.lastSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.lastSlide

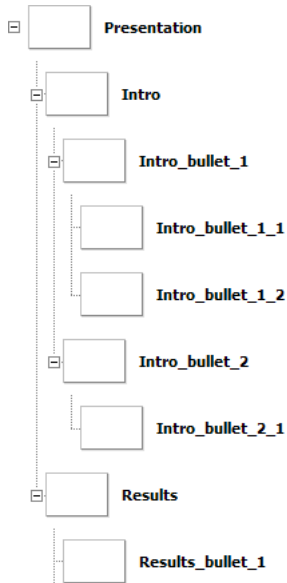
Description

Property (read-only); returns the last child slide of *mySlide* that has no child slides.

Example

The following statements are all true concerning the slide hierarchy shown below:

```
Presentation.lastSlide._name == Results_bullet_1;  
Intro.lastSlide._name == Intro_bullet_1_2;  
Intro_bullet_1.lastSlide._name == Intro_bullet_1_2;  
Results.lastSlide._name == Results_bullet_1;
```



Slide.nextSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.nextSlide

Description

Property (read-only); returns the slide you would reach if you called *mySlide*.gotoNextSlide(), but does not actually navigate to that slide. For example, you can use this property to display the name of the next slide in a presentation and let users select whether they want to navigate to that slide.

Example

In this example, the label of a Button component named `nextButton` displays the name of the next slide in the presentation. If there is no next slide—that is, if `mySlide.nextSlide` is `null`—then the button's label is updated to indicate that the user is at the end of this slide presentation.

```
if (mySlide.nextSlide != null) {
    nextButton.label = "Next slide: " + mySlide.nextSlide._name + " > ";
} else {
    nextButton.label = "End of this slide presentation.";
}
```

See also

[Slide.gotoNextSlide\(\)](#), [Slide.previousSlide](#)

Slide.numChildSlides

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.numChildSlides

Description

Property (read-only); returns the number of child slides that *mySlide* contains. A slide can contain either forms or other slides; if *mySlide* contains both slides and forms, this property only returns the number of slides, and does not count forms.

Example

This example uses `Slide.numChildSlides` and the `Slide.getChildSlide()` method to iterate over all the child slides of the root presentation slide. It then displays their names in the Output panel.

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
    var childSlide = _root.Presentation.getChildSlide(slideIndex);
    trace(childSlide._name);
}
```

See also

[Slide.getChildSlide\(\)](#)

Slide.overlayChildren

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`mySlide.overlayChildren`

Description

Property; determines whether child slides of *mySlide* remain visible when navigating from one child slide to the next. When this property is `true`, the previous slide remains visible when control passes to its next sibling slide; when this property is `false`, the previous slide is invisible when control passes to its next sibling slide.

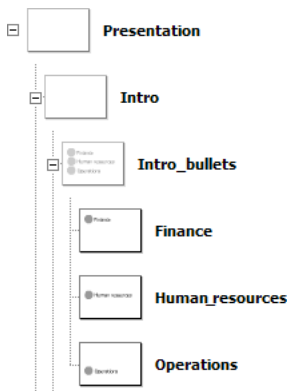
Setting this property to `true` is useful, for example, when a given slide contains several child “bullet point” slides that are revealed separately (using transitions, perhaps), but all need to remain visible as new bullet points appear.

NOTE

This property applies only to the immediate descendants of *mySlide*, not to all (nested) child slides.

Example

The `Intro_bullets` slide in the following illustration contains three child slides (`Finance`, `Human_resources`, and `Operations`) that each display a separate bullet point. By setting `Intro_bullets.overlayChildren` to `true`, each bullet slide remains on the Stage as the other bullet points appear.



Slide.parentIsSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.parentIsSlide

Description

Property (read-only); a Boolean value indicating whether the parent object of *mySlide* is also a slide. If the parent object of *mySlide* is a slide, or belongs to a subclass of Slide, this property returns `true`; otherwise, it returns `false`.

If *mySlide* is the root slide in a presentation, this property returns `false`, because the presentation slide's parent is the main (`_level0`), not a slide. This property also returns `false` if a form is the parent of *mySlide*.

Example

The following code determines whether the parent object of the slide *mySlide* is itself a slide. If *mySlide.parentIsSlide* is `true`, the number of *mySlide*'s sibling slides is displayed in the Output panel. If the parent object is not a slide, Flash assumes that *mySlide* is the root (master) slide in the presentation and therefore has no sibling slides.

```
if (mySlide.parentIsSlide) {  
    trace("I have " + mySlide._parent.numChildSlides+" sibling slides");  
} else {  
    trace("I am the root slide and have no siblings");  
}
```

See also

[Slide.numChildSlides](#)

Slide.parentSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.parentSlide

Description

Property (read-only); a reference to the slide containing the current slide.

Slide.playHidden

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.playHidden

Description

Property; a Boolean value that specifies whether *mySlide* should continue to play when it is hidden. When this property is *true*, *mySlide* continues to play when hidden. When set to *false*, *mySlide* is stopped upon being hidden; upon being revealed, play restarts at Frame 1 of *mySlide*.

Slide.previousSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.previousSlide

Description

Property (read-only); returns the slide you would reach if you called *mySlide*.gotoPreviousSlide(), but does not actually navigate to that slide. For example, you can use this property to display the name of the previous slide in a presentation and let users select whether they want to navigate to that slide.

Example

In this example, the label of a Button component named `previousButton` displays the name of the previous slide in the presentation. If there is no previous slide—that is, if `mySlide.previousSlide` is `null`—the button's label is updated to indicate that the user is at the beginning of this slide presentation.

```
if (mySlide.previousSlide != null) {
    previousButton.label = "Previous slide: " + mySlide.previous._name + "
    > ";
} else {
    previousButton.label = "You're at the beginning of this slide
    presentation.";
}
```

See also

[Slide.gotoPreviousSlide\(\)](#), [Slide.nextSlide](#)

Slide.revealChild

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
on(revealChild) {
    // Your code here.
}
```

Description

Event; broadcast each time a child slide of a slide object changes from invisible to visible. This event is used primarily to attach “in” transitions to all the child slides of a given slide.

Example

When attached to the root slide (for example, the presentation slide), this code displays the name of each child slide as it appears.

```
on(revealChild) {
    var child = eventObj.target._name;
    trace(child + " has just appeared");
}
```

See also

[Slide.hideChild](#)

Slide.rootSlide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

mySlide.rootSlide

Description

Property (read-only); returns the root slide of the slide tree, or slide subtree, that contains *mySlide*.

Example

Suppose you have a movie clip on a slide that, when clicked, goes to the first slide in the presentation. To accomplish this, you would attach the following code to the movie clip:

```
on(press) {  
    _parent.rootSlide.gotoFirstSlide();  
}
```

In this case, `_parent` refers to the slide that contains the movie clip object.

ActionScript Class Name `mx.styles.StyleManager`

The `StyleManager` class keeps track of known inheriting styles and colors. You need to use this class only if you are creating components and want to add a new inheriting style or color.

To determine which styles are inheriting, see the W3C web site at www.w3.org/Style/CSS/.

Method summary for the `StyleManager` class

The following table lists methods of the `StyleManager` class.

Method	Description
<code>StyleManager.registerColorName()</code>	Registers a new color name with the Style Manager.
<code>StyleManager.registerColorStyle()</code>	Adds a new color style to the Style Manager.
<code>StyleManager.registerInheritingStyle()</code>	Registers a new inheriting style with the Style Manager.

StyleManager.registerColorName()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
StyleManager.registerColorName(colorName, value)
```

Parameters

colorName A string indicating the name of the color (for example, "gray", "darkGrey", and so on).

value A hexadecimal number indicating the color (for example, 0x808080, 0x404040, and so on).

Returns

Nothing.

Description

Method; associates a color name with a hexadecimal value and registers it with the Style Manager.

Example

The following example registers "gray" as the color name for the color represented by the hexadecimal value 0x808080:

```
StyleManager.registerColorName("gray", 0x808080);
```


StyleManager.registerColorStyle()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
StyleManager.registerColorStyle(colorStyle)
```

Parameters

colorStyle A string indicating the name of the color (for example, "highlightColor", "shadowColor", "disabledColor", and so on).

Returns

Nothing.

Description

Method; adds a new color style to the Style Manager.

Example

The following example registers "highlightColor" as a color style:

```
StyleManager.registerColorStyle("highlightColor");
```

StyleManager.registerInheritingStyle()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
StyleManager.registerInheritingStyle(propertyName)
```

Parameters

propertyName A string indicating the name of the style property (for example, "newProp1", "newProp2", and so on).

Returns

Nothing.

Description

Method; marks this style property as inheriting. Use this method to register style properties that aren't listed in the CSS specification. Do not use this method to change non-inheriting style properties to inheriting.

When a style's value is not inherited, you can set its style only on an instance, not on a custom or global style sheet. A style that doesn't inherit its value is set on the class style sheet, and therefore, setting it on a custom or global style sheet does not work.

Example

The following example registers `newProp1` as an inheriting style:

```
StyleManager.registerInheritingStyle("newProp1");
```

ActionScript Class Name mx.managers.SystemManager

The SystemManager class works automatically with the FocusManager class to handle which top-level window is activated in an application that contains version 2 components. It also provides a screen property that allows components and movie clips to access Stage coordinates.

Property summary for the SystemManager class

The following table lists the property of the SystemManager class.

Property	Description
SystemManager.screen	Read-only; an object containing the size and position of the Stage.

SystemManager.screen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`SystemManager.screen`

Description

Property; an object with `x`, `y`, `width`, and `height` properties that indicate the size and position of the Stage.

Example

If `Stage.align` is set to something other than "LT", it is difficult to know what coordinates are actually viewable.

Suppose you want to place a watermark movie clip in the lower-right corner of the Stage (similar to the watermarks many television channels use). The following code would work in all Stage alignments for a movie clip instance `watermark`:

```
import mx.managers.SystemManager;

var p1:Number = SystemManager.screen.width + SystemManager.screen.x -
    watermark._width;
var p2:Number = SystemManager.screen.height + SystemManager.screen.y -
    watermark._height;

watermark._x = p1;
watermark._y = p2;
```

The `TextArea` component wraps the native ActionScript `TextField` object. You can use styles to customize the `TextArea` component; when an instance is disabled, its contents display in a color represented by the `disabledColor` style. A `TextArea` component can also be formatted with HTML, or as a password field that disguises the text. See “Applying a style sheet to a `TextArea` component” in *Learning ActionScript 2.0 in Flash*.

A `TextArea` component can be enabled or disabled in an application. In the disabled state, it doesn’t receive mouse or keyboard input. When enabled, it follows the same focus, selection, and navigation rules as an ActionScript `TextField` object. When a `TextArea` instance has focus, you can use the following keys to control it:

Key	Description
Arrow keys	Move the insertion point one line up, down, left, or right.
Page Down	Moves one screen down.
Page Up	Moves one screen up.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see “Creating custom focus navigation” in *Using Components* or “[FocusManager class](#)” on page 721.

A live preview of each `TextArea` instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. If a scroll bar is needed, it appears in the live preview, but it does not function. Text is not selectable in the live preview, and you cannot enter text in the component instance on the Stage.

When you add the `TextArea` component to an application, you can use the Accessibility panel to make it accessible to screen readers.

Using the TextArea component

You can use a `TextArea` component wherever you need a multiline text field. If you need a single-line text field, use the [TextInput component](#). For example, you could use a `TextArea` component as a comment field in a form. You could set up a listener that checks if the field is empty when a user tabs out of the field. That listener could display an error message indicating that a comment must be entered in the field.

TextArea parameters

You can set the following authoring parameters for each `TextArea` component instance in the Property inspector or the Component inspector (Window > Component Inspector menu option):

editable indicates whether the `TextArea` component is editable (`true`) or not (`false`). The default value is `true`.

html indicates whether the text is formatted with HTML (`true`) or not (`false`). If HTML is set to `true`, you can format the text using the `font` tag. The default value is `false`.

text indicates the contents of the `TextArea` component. You cannot enter carriage returns in the Property inspector or the Component inspector. The default value is "" (an empty string).

wordWrap indicates whether the text wraps (`true`) or not (`false`). The default value is `true`.

NOTE

If you create a `TextArea` using the `createClassObject()` method, the default value for `wordWrap` is `false`.

You can set the following additional parameters for each `TextArea` component instance in the Component inspector (Window > Component Inspector):

maxChars is the maximum number of characters that the text area can contain. The default value is `null` (meaning unlimited).

restrict indicates the set of characters that a user can enter in the text area. The default value is `undefined`. See [“TextArea.restrict” on page 1199](#).

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

password is a Boolean value that indicates whether the input is a password or other text that should be hidden from view as it is typed. Flash hides the input characters with asterisks. The default value is `false`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to control these and additional options for the `TextArea` component using its properties, methods, and events. For more information, see “[TextArea class](#)” on page 1182.

Creating an application with the `TextArea` component

The following procedure explains how to add a `TextArea` component to an application while authoring. The example sets up a `focusOut` event handler on the `TextArea` instance that verifies that the user typed something in the text area before giving focus to a different part of the interface.

To create an application with the `TextArea` component:

1. Drag a `TextArea` component from the Components panel to the Stage and give it an instance name of `my_ta`.
2. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

var taListener:Object = new Object();
taListener.focusOut = function(evt_obj:Object) {
    if (my_ta.length < 1) {
        trace("Please enter a comment");
    }
};
my_ta.addEventListener("focusOut", taListener);
```

This code sets up a `focusOut` event handler on the `TextArea` component instance that verifies that the user typed something in the text area.

You can get the value of text that is entered in the `TextArea` instance, as follows:

```
var ta_text:String = my_ta.text;
```

Customizing the TextArea component

You can transform a `TextArea` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use `UIObject.setSize()` or any applicable properties and methods of the [TextArea class](#).

When a `TextArea` component is resized, the border is resized to the new bounding box. The scroll bars are placed on the bottom and right edges if they are required. The text area is then resized within the remaining area; there are no fixed-size elements in a `TextArea` component. If the `TextArea` component is too small to display the text, the text is clipped.

Using styles with the TextArea component

The `TextArea` component has its `backgroundColor` and `borderStyle` style properties defined on a class style declaration. Class styles override global styles; therefore, if you want to set the `backgroundColor` and `borderStyle` style properties, you must create a different custom style declaration on the instance.

If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in *Using Components*.

A `TextArea` component supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>	Both	The background color. The default color is white.
<code>borderStyle</code>	Both	The <code>TextArea</code> component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063 . The default border style is “inset”.
<code>marginLeft</code>	Both	A number indicating the left margin for text. The default value is 0.
<code>marginRight</code>	Both	A number indicating the right margin for text. The default value is 0.
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).

Style	Theme	Description
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , <code>"center"</code> , or <code>"justify"</code> . (The <code>"justify"</code> parameter is supported only in Flash Player 8). The default value is <code>"left"</code> .
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `TextArea` and `TextInput` components use exactly the same styles and are often used in the same manner. Thus, by default they share the same class-level style declaration.

For example, the following code sets a style on the `TextInput` declaration, but it affects both `TextInput` and `TextArea` components.

```
_global.styles.TextInput.setStyle("disabledColor", 0xBBBBFF);
```

To separate the components and provide class-level styles for one and not the other, create a new style declaration.

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextArea = new CSSStyleDeclaration();
_global.styles.TextArea.setStyle("disabledColor", 0xFFBBBB);
```

This example does not check if `_global.styles.TextArea` existed before overwriting it; it assumes you know it exists and want to overwrite it.

You can make the background of `TextArea` components transparent by setting the `backgroundColor` style globally to a value of `undefined`. You then need to set the `backgroundColor` style to a color individually for all `TextArea` components that you do not want to be transparent.

```
// Give all TextArea components transparent backgrounds.
_global.styles.TextArea.backgroundColor = undefined;

//Make this specific component instance have a white background.
myTextArea2.setStyle( "backgroundColor", "white" );
```

The `TextArea` component supports one set of component styles for all text in the field. However, you can also display HTML that is compatible with Flash Player HTML rendering. To display HTML text, set `TextArea.html` to `true`.

If you do set the `TextArea` to display HTML text, the text style is set using the `TextField.StyleSheet` class (see details for this class in the *ActionScript 2.0 Language Reference*). For example:

1. Drag a `TextArea` component to the Stage, and give it the instance name `my_ta`.
2. Enter this code in Actions panel for Frame 1 of the timeline:

```
var my_styles = new TextField.StyleSheet();
my_styles.setStyle("p", {fontFamily:'Arial,Helvetica,sans-serif',
    fontSize:'12px', color:'#CC6699'});
my_ta.styleSheet = my_styles;
my_ta.html = true;
my_ta.text = "<p>This is some text</p>";
```

Using skins with the `TextArea` component

The `TextArea` component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see [“RectBorder class” on page 1063](#) and [“Using skins with the `UIScrollBar` component” on page 1394](#).

TextArea class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `TextArea`

ActionScript Class Name `mx.controls.TextArea`

The properties of the `TextArea` class let you set the text content, formatting, and horizontal and vertical position at runtime. You can also indicate whether the field is editable, and whether it is a “password” field. You can also restrict the characters that a user can enter.

Setting a property of the `TextArea` class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The `TextArea` component overrides the default Flash Player focus rectangle and draws a custom focus rectangle with rounded corners.

The `TextArea` component supports CSS styles and any additional HTML styles supported by Flash Player.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.TextArea.version);
```

NOTE

The code `trace(myTextAreaInstance.version);` returns `undefined`.

Method summary for the `TextArea` class

There are no methods exclusive to the `TextArea` class.

Methods inherited from the `UIObject` class

The following table lists the methods the `TextArea` class inherits from the `UIObject` class.

When calling these methods from the `TextArea` object, use the form

TextAreaInstance.methodName.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.

Method	Description
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the `TextArea` class inherits from the `UIComponent` class. When calling these methods from the `TextArea` object, use the form

TextAreaInstance.methodName.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the TextArea class

The following table lists properties of the `TextArea` class.

Property	Description
TextArea.editable	A Boolean value indicating whether the field is editable (<code>true</code>) or not (<code>false</code>).
TextArea.hPosition	Defines the horizontal position of the text in the field.
TextArea.hScrollPolicy	Indicates whether the horizontal scroll bar is always on ("on"), is never on ("off"), or turns on when needed ("auto").
TextArea.html	A Boolean value that indicates whether the text area contents can be formatted with HTML.
TextArea.length	Read-only; the number of characters in the text area.
TextArea.maxChars	The maximum number of characters that the text area can contain.
TextArea.maxHPosition	Read-only; the maximum value of TextArea.hPosition .
TextArea.maxVPosition	Read-only; the maximum value of TextArea.vPosition .
TextArea.password	A Boolean value indicating whether the field is a password field (<code>true</code>) or not (<code>false</code>).
TextArea.restrict	The set of characters that a user can enter in the text area.
TextArea.styleSheet	Attaches a style sheet to the specified <code>TextArea</code> component.
TextArea.text	The text contents of a <code>TextArea</code> component.

Property	Description
<code>TextArea.vPosition</code>	A number indicating the vertical scrolling position.
<code>TextArea.vScrollPolicy</code>	Indicates whether the vertical scroll bar is always on ("on"), is never on ("off"), or turns on when needed ("auto").
<code>TextArea.wordWrap</code>	A Boolean value indicating whether the text wraps (<code>true</code>) or not (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the `TextArea` class inherits from the `UIObject` class. When accessing these properties from the `TextArea` object, use the form `TextAreaInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	Read-only; a number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the `UIComponent` class

The following table lists the properties the `TextArea` class inherits from the `UIComponent` class. When accessing these properties from the `TextArea` object, use the form

`TextAreaInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `TextArea` class

The following table lists the event of the `TextArea` class.

Event	Description
<code>TextArea.change</code>	Notifies listeners that text has changed.
<code>TextArea.scroll</code>	Notifies listeners that text has scrolled.

Events inherited from the `UIObject` class

The following table lists the events the `TextArea` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the `TextArea` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

TextArea.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // ...
};
textAreaInstance.addEventListener("change", listenerObject);
```

Usage 2:

```
on (change) {
    // ...
}
```

Description

Event; notifies listeners that text has changed. This event is broadcast after the text has changed. This event cannot be used to prevent certain characters from being added to the component's text area; for this purpose, use [TextArea.restrict](#).

The first usage example uses a dispatcher/listener event model. A component instance (*textAreaInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `TextArea` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextArea`, sends “_level0.myTextArea” to the Output panel:

```
on (change) {
    trace(this);
}
```

Example

This example uses the dispatcher/listener event model to track the total of number of times the text area changes in a `TextArea` component named `my_ta`.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

// Create a Number variable to track the number of changes to the TextArea.
var changeCount_num:Number = 0;
```



```
// Define a listener object.
var taListener:Object = new Object();
// Define a function that is executed whenever the listener receives
// notification of a change in the TextArea component.
taListener.change = function(evt_obj:Object) {
    changeCount_num++;
    trace("Text has changed " + changeCount_num + " times now!");
    trace("It now contains: " + evt_obj.target.text);
    trace("");
};
// Register the listener object with the TextArea component instance.
my_ta.addEventListener("change", taListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

TextArea.editable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.editable

Description

Property; a Boolean value that indicates whether the component is editable (*true*) or not (*false*). The default value is *true*.

Example

The following example sets the *editable* property to *false* to prevent the user from editing the text that it loads into the *TextArea* instance called *my_ta*.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.editable = false;

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

TextArea.hPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.hPosition

Description

Property; defines the horizontal position in pixels of the text in the field. The default value is 0.

Example

The following example uses a listener to display the current horizontal position in the Output panel as the user scrolls back and forth through the text that it has loaded into the `TextArea` instance called `my_ta`.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
    trace("hPosition = " + my_ta.hPosition);
}
my_ta.addEventListener("scroll", taListener);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
        my_ta.hPosition = 200;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

TextArea.hScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.hScrollPolicy

Description

Property; determines whether the horizontal scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the field ("auto"). The default value is "auto".

Example

The following example turns off the `hScrollPolicy` property, causing the `TextArea` instance to not have a scroll bar.

You must first add an instance of the `TextArea` component to the `Stage` and name it `my_ta`; then add the following code to `Frame 1`.

```
/**
 * Requires:
 *   - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;
my_ta.hScrollPolicy = "off";

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

TextArea.html

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.html

Description

Property; a Boolean value that indicates whether the text area contents are formatted with HTML (`true`) or not (`false`). If the `html` property is `true`, the text area contents are in HTML. If `html` is `false`, the text area is a non-HTML text area. The default value is `false`.

Example

The following example makes the TextArea called `my_ta` an HTML text area and then formats the text with HTML tags.

You must first add an instance of the TextArea component to the Stage and name it `my_ta`; then add the following code to Frame 1:

```
/**
 * Requires:
 *   - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.html = true;
my_ta.text = "The <b>Royal</b> Nonesuch";
```

TextArea.length

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.length

Description

Property (read-only); indicates the number of characters in a text area. This property returns the same value as the ActionScript `text.length` property, but is faster. A character such as tab ("`\t`") counts as one character. The default value is 0.

Example

The following example accesses the `length` property to display the number of characters that the user types in the TextArea called `my_ta`.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

// Define a listener object.
var taListener:Object = new Object();
taListener.change = function(evt_obj:Object) {
    trace("my_ta.length is now: " + my_ta.length + " characters");
};
my_ta.addEventListener("change", taListener);
```

TextArea.maxChars

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.maxChars

Description

Property; the maximum number of characters that the text area can contain. A script may insert more text than the `maxChars` property allows; the property indicates only how much text a user can enter. If the value of this property is `null`, there is no limit to the amount of text a user can enter. The default value is `null`.

Example

The following example sets the `maxchars` property to limit the number of characters a user can enter to 24.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.maxChars = 24;

// Define a listener object.
var taListener:Object = new Object();
taListener.change = function(evt_obj:Object) {
    trace("my_ta.length is now: " + my_ta.length + " characters");
};
my_ta.addEventListener("change", taListener);
```

TextArea.maxHPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.maxHPosition

Description

Read-only property; the maximum value of `TextArea.hPosition`. The default value is 0.

Example

The following example accesses the `maxHPosition` property to set the initial position in the `TextArea` called `my_ta` to the farthest right position.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
    trace("hPosition = " + my_ta.hPosition);
}
my_ta.addEventListener("scroll", taListener);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
        my_ta.hPosition = my_ta.maxHPosition;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

See also

[TextArea.vPosition](#)

TextArea.maxVPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.maxVPosition

Description

Read-only property; indicates the maximum value of `TextArea.vPosition`. The default value is 0.

Example

The following example accesses the `maxVPosition` property to set the initial vertical position of the `TextArea` called `my_ta` to the bottom. It also traces the current vertical position as the user scrolls up and down.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = true;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
    trace("vPosition = " + my_ta.vPosition);
}
my_ta.addEventListener("scroll", taListener);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
        my_ta.vPosition = my_ta.maxVPosition;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

See also

[TextArea.hPosition](#)

TextArea.password

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.password

Description

Property; a Boolean value indicating whether the text area is a password field (`true`) or not (`false`). If `password` is `true`, the text area is a password text area and hides the input characters with asterisks. If `password` is `false`, the text area is not a password text area. The default value is `false`.

Example

The following example treats the text in the `TextArea` called `my_ta` as a password field if the check box called `my_ch` is checked. Otherwise it treats it as ordinary text.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta` and also add a check box and name it `my_ch`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 * - CheckBox instance on Stage (instance name: my_ch)
 */
var my_ta:mx.controls.TextArea;
var my_ch:mx.controls.CheckBox;

my_ta.wordWrap = false;
my_ta.password = true;
my_ch.selected = my_ta.password;

var chListener:Object = new Object();
chListener.click = function(evt_obj:Object) {
    my_ta.password = my_ch.selected;
}
my_ch.addEventListener("click", chListener);
```

TextArea.restrict

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.restrict

Description

Property; indicates the set of characters that users can enter in the text area. The default value is undefined. If this property is null, users can enter any character. If this property is an empty string, no characters can be entered. If this property is a string of characters, users can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (-).

If the string begins with ^, all characters that follow the ^ are considered unacceptable characters. If the string does not begin with ^, the characters in the string are considered acceptable. The ^ can also be used as a toggle between acceptable and unacceptable characters.

For example, the following code allows A-Z except X and Q:

```
Ta.restrict = "A-Z^XQ";
```

Restricting input to uppercase characters converts alphabetic characters entered in lowercase to uppercase. Likewise, restricting input to lowercase characters converts characters entered in uppercase to lowercase.

The `restrict` property only restricts user interaction; a script may put any text into the text area. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

The following example first sets the `restrict` property to limit the text area to uppercase letters, numbers, and spaces, and then sets it to allow all characters except lowercase letters.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1. Use only one setting for the `restrict` property at a time.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */
var my_ta:mx.controls.TextArea;

my_ta.wordWrap = true;

// Limit control to uppercase letters, numbers, and spaces.
my_ta.restrict = "A-Z 0-9";

// Allow all characters, except lowercase letters
// characters to uppercase
my_ta.restrict = "^a-z";
```

TextArea.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
    // ...
};
textAreaInstance.addEventListener("scroll", listenerObject);
```

Usage 2:

```
on (scroll) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse button is clicked (released) over the scroll bar. The `UIScrollBar.scrollPosition` property and the scroll bar's onscreen image are updated before this event is broadcast.

The first usage example uses a dispatcher/listener event model, in which the script is placed on a frame in the `that` that contains the component instance. A component instance (*textAreaInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event occurs. When the event occurs, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see `EventDispatcher.addEventListener()`) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

In addition to the normal properties of the event object (`type` and `target`), the event object for the `scroll` event includes a third property named `direction`. The `direction` property contains a string describing which way the scroll bar is oriented. The possible values for the `direction` property are `vertical` (the default) and `horizontal`.

For more information about the `type` and `target` event object properties, see “Event objects” on page 499.

The second usage example uses an `on()` handler and must be attached directly to a `TextArea` component instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `TextArea` component instance `myTextAreaComponent`, sends “_level0.myTextAreaComponent” to the Output panel:

```
on (scroll) {  
    trace(this);  
}
```

Example

This example uses the dispatcher/listener event model to track when the user scrolls the `TextArea` using the `TextArea` instance's scroll bars or scroll box.

You first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1:

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

my_ta.setSize(320, 240);
my_ta.move(10, 10);

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String):Void {
    my_ta.text = src;
}
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");

my_ta.addEventListener("scroll", doScroll);
function doScroll(evt_obj:Object):Void {
    trace("target:    " + evt_obj.target);
    trace("type:      " + evt_obj.type);
    trace("direction: " + evt_obj.direction);
    trace("position:  " + evt_obj.position);
    trace("");
}
}
```

See also

[EventDispatcher.addEventListener\(\)](#)

TextArea.styleSheet

Availability

Flash Player 7.

Usage

```
textAreaInstance.styleSheet = TextFieldStyleSheetObject
```

Description

Property; attaches a style sheet to the `TextArea` component specified by *TextAreaInstance*. For information on creating style sheets, see “Formatting text with Cascading Style Sheet styles” in *Learning ActionScript 2.0 in Flash*.

The style sheet associated with a `TextArea` component may be changed at any time. If the style sheet in use is changed, the `TextArea` component is redrawn with the new style sheet. The style sheet may be set to `null` or `undefined` to remove the style sheet. If the style sheet in use is removed, the `TextArea` component is redrawn without a style sheet. The formatting done by a style sheet is not retained if the style sheet is removed.

Example

The following code creates a new `StyleSheet` object named `my_styles` with the new `TextField.StyleSheet` constructor. It then defines styles for `html` and `body` tags. Next, it applies the style by assigning `my_styles` to the `styleSheet` property of the `TextArea` instance `my_ta`.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);

// Create the new StyleSheet object.
var my_styles:TextField.StyleSheet = new TextField.StyleSheet();
my_styles.setStyle("html", {fontFamily:"Arial,Helvetica,sans-serif",
    fontSize:"12px", color:"#0000FF"});
my_styles.setStyle("body", {color:"#00CCFF", textDecoration:"underline"});

// Set the TextAreaInstance.styleSheet property to the newly defined
// styleSheet object named styles.
my_ta.styleSheet = my_styles;
my_ta.html = true;

// Load text to display and define onLoad handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
    } else {
        my_ta.text = "Error loading HTML document.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

TextArea.text

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

TextAreaInstance.text

Description

Property; the text contents of a TextArea component. The default value is "" (an empty string).

Example

The following example places a string in the `text` property of the `my_ta` TextArea instance, and then traces that string to the Output panel.

You must first add an instance of the TextArea component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.text = "The Royal Nonesuch";
trace(my_ta.text); // traces "The Royal Nonesuch"
```


TextArea.vPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.vPosition

Description

Property; defines the vertical scroll position of text in a text area. This property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text areas. You can get and set this property. The default value is 0.

Example

The following example loads text into the TextArea called `my_ta` and sets the `vPosition` property to display the text at the bottom.

You must first add an instance of the TextArea component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
        my_ta.vPosition = my_ta.maxVPosition;
    } else {
        trace("Error loading text.");
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt")
```

TextArea.vScrollPolicy

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.vScrollPolicy

Description

Property; determines whether the vertical scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the field ("auto"). The default value is "auto".

Example

The following example turns off the vertical scroll bar for the TextArea called my_ta so that a scroll bar is not available to scroll the text that the example loads.

You must first add an instance of the TextArea component to the Stage and name it my_ta; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextArea instance on Stage (instance name: my_ta)
 */
var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = true;
my_ta.vScrollPolicy = "off";

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

TextArea.wordWrap

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.wordWrap

Description

Property; a Boolean value that indicates whether the text wraps (*true*) or not (*false*). The default value is *true*.

NOTE

If you create a `TextArea` instance using the `createClassObject()` method, the default for `wordWrap` is `false`.

Example

The following example sets the `wordwrap` property to `false` for the `TextArea` called `my_ta`, causing it to have a horizontal scroll bar to access the text beyond the side boundaries.

You must first add an instance of the `TextArea` component to the Stage and name it `my_ta`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextArea instance on Stage (instance name: my_ta)
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;
//my_ta.vScrollPolicy = "off";

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_ta.text = src;
    } else {
        my_ta.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```


The `TextInput` component is a single-line text component that is a wrapper for the native ActionScript `TextField` object. You can use styles to customize the `TextInput` component; when an instance is disabled, its contents appear in a color represented by the `disabledColor` style. A `TextInput` component can also be formatted with HTML, or as a password field that disguises the text.

A `TextInput` component can be enabled or disabled in an application. In the disabled state, it doesn't receive mouse or keyboard input. When enabled, it follows the same focus, selection, and navigation rules as an ActionScript `TextField` object. When a `TextInput` instance has focus, you can also use the following keys to control it:

Key	Description
Arrow keys	Move the insertion point one character left and right.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

A live preview of each `TextInput` instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. Text is not selectable in the live preview, and you cannot enter text in the component instance on the Stage.

When you add the `TextInput` component to an application, you can use the Accessibility panel to make it accessible to screen readers.

Using the TextInput component

You can use a `TextInput` component wherever you need a single-line text field. If you need a multiline text field, use the [TextArea component](#). For example, you could use a `TextInput` component as a password field in a form. You could also set up a listener that checks if the field has enough characters when a user tabs out of the field. That listener could display an error message indicating that the proper number of characters must be entered.

TextInput parameters

You can set the following authoring parameters for each `TextInput` component instance in the Property inspector or the Component inspector (Window > Component Inspector menu option):

editable indicates whether the `TextInput` component is editable (`true`) or not (`false`). The default value is `true`.

password indicates whether the field is a password field (`true`) or not (`false`). The default value is `false`.

text specifies the contents of the `TextInput` component. You cannot enter carriage returns in the Property inspector or the Component inspector. The default value is "" (an empty string).

You can set the following additional parameters for each `TextInput` component instance in the Component inspector (Window > Component Inspector):

maxChars is the maximum number of characters that the text input field can contain. The default value is `null` (meaning unlimited).

restrict indicates the set of characters that a user can enter in the text input field. The default value is `undefined`. See [“TextInput.restrict” on page 1229](#).

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write `ActionScript` to control these and additional options for the `TextInput` component using its properties, methods, and events. For more information, see [“TextInput class” on page 1214](#).

Creating an application with the TextInput component

The following procedure explains how to add a TextInput component to an application while authoring. In this example, the component is a password field with an event listener that determines if the proper number of characters has been entered.

To create an application with the TextInput component:

1. Drag a TextInput component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name `my_ti`.
 - Leave the text parameter blank.
 - Set the editable parameter to `true`.
3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
/**
 * Requires:
 * - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.handleEvent = function (evt_obj:Object){
    if (evt_obj.type == "enter"){
        if (my_ti.length < 8) {
            trace("You must enter at least 8 characters");
        } else {
            trace("Thanks");
        }
    }
}
// Add listener.
my_ti.addEventListener("enter", tiListener);
```

This code sets up an enter event handler on the TextInput instance called `my_ti`. If the user types less than eight characters, the example displays the message: You must enter at least 8 characters. If the user enters eight or more characters, the example displays: Thanks.

4. After text is entered in the `my_ti` instance, you can get its value as follows:

```
var my_text:String = my_ti.text;
```

Customizing the TextInput component

You can transform a `TextInput` component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use `UIObject.setSize()` or any applicable properties and methods of the `TextInput` class.

When a `TextInput` component is resized, the border is resized to the new bounding box. The `TextInput` component doesn't use scroll bars, but the insertion point scrolls automatically as the user interacts with the text. The text field is then resized within the remaining area; there are no fixed-size elements in a `TextInput` component. If the `TextInput` component is too small to display the text, the text is clipped.

Using styles with the TextInput component

The `TextInput` component has its `backgroundColor` and `borderStyle` style properties defined on a class style declaration. Class styles override global styles; therefore, if you want to set the `backgroundColor` and `borderStyle` style properties, you must create a different custom style declaration or define it on the instance.

A `TextInput` component supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>	Both	The background color. The default color is white.
<code>borderStyle</code>	Both	The <code>TextInput</code> component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063 . The default border style is <code>"inset"</code> .
<code>marginLeft</code>	Both	A number indicating the left margin for text. The default value is 0.
<code>marginRight</code>	Both	A number indicating the right margin for text. The default value is 0.
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).

Style	Theme	Description
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `TextArea` and `TextInput` components both use the same styles and are often used in the same manner. Thus, by default they share the same class-level style declaration. For example, the following code sets a style on the `TextArea` declaration but it affects both `TextArea` and `TextInput` components.

```
_global.styles.TextArea.setStyle("disabledColor", 0xBBBBFF);
```

To separate the components and provide class-level styles for one and not the other, create a new style declaration.

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextInput = new CSSStyleDeclaration();
_global.styles.TextInput.setStyle("disabledColor", 0xFFBBBB);
```

Notice how this example does not check if `_global.styles.TextInput` existed before overwriting it; in this example, you know it exists and you want to overwrite it.

Using skins with the TextInput component

The TextArea component uses an instance of RectBorder for its border. For more information about skinning these visual elements, see [“RectBorder class” on page 1063](#).

TextInput class

Inheritance MovieClip > UIObject class > UIComponent class > TextInput

ActionScript Class Name mx.controls.TextInput

The properties of the TextInput class let you set the text content, formatting, and horizontal position at runtime. You can also indicate whether the field is editable, and whether it is a “password” field. You can also restrict the characters that a user can enter.

Setting a property of the TextInput class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The TextInput component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see [“FocusManager class” on page 721](#).

The TextInput component supports CSS styles and any additional HTML styles supported by Flash Player. For information about CSS support, see the W3C specification at www.w3.org/TR/REC-CSS2/.

You can manipulate the text string by using the string returned by the text object.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.TextInput.version);
```

NOTE

The code `trace(myTextInputInstance.version);` returns `undefined`.

Method summary for the TextInput class

There are no methods exclusive to the TextInput class.

Methods inherited from the UIObject class

The following table lists the methods the TextInput class inherits from the UIObject class.

When calling these methods from the TextInput object, use the form

TextInputInstance.methodName.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the TextInput class inherits from the UIComponent class. When calling these methods from the TextInput object, use the form

TextInputInstance.methodName.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the TextInput class

The following table lists properties of the TextInput class.

Property	Description
TextInput.editable	A Boolean value indicating whether the field is editable (<code>true</code>) or not (<code>false</code>).
TextInput.hPosition	The horizontal scrolling position of the text in the field.
TextInput.length	Read-only; the number of characters in a TextInput component.
TextInput.maxChars	The maximum number of characters that a user can enter in the text field.
TextInput.maxHPosition	Read-only; the maximum possible value for <code>TextField.hPosition</code> .
TextInput.password	A Boolean value that indicates whether the text field is a password field that hides the entered characters.
TextInput.restrict	Indicates which characters a user can enter in a text field.
TextInput.text	Sets the text content of a TextInput component.

Properties inherited from the UIObject class

The following table lists the properties the `TextInput` class inherits from the `UIObject` class.

When accessing these properties from the `TextInput` object, use the form

`TextInputInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the `TextInput` class inherits from the `UIComponent` class. When accessing these properties from the `TextInput` object, use the form

`TextInputInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the TextInput class

The following table lists events of the TextInput class.

Event	Description
<code>TextInput.change</code>	Broadcast when the TextInput field changes.
<code>TextInput.enter</code>	Broadcast when the Enter key is pressed.

Events inherited from the UIObject class

The following table lists the events the TextInput class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the TextInput class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

TextInput.change

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    //...
};
textInputInstance.addEventListener("change", listenerObject)
```

Usage 2:

```
on (change){
    //...
}
```

Description

Event; notifies listeners that text has changed. This event is broadcast after the text has changed. This event cannot be used to prevent certain characters from being added to the component's text field; for that purpose, use [TextInput.restrict](#). This event is triggered only by user input, not by programmatic change.

The first usage example uses an `on()` handler and must be attached directly to a `TextInput` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextInput`, sends “`_level0.myTextInput`” to the Output panel:

```
on (change){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*textInputInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

This example creates a listener for a *change* event on the `my_ti` `TextInput` instance. When a *change* event occurs, the example displays “Input has changed”.

You must first drag a `TextInput` component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
    trace("Input has changed");
};
// Add listener.
my_ti.addEventListener("change", tiListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

TextInput.editable

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.editable

Description

Property; a Boolean value that indicates whether the component is editable (*true*) or not (*false*). The default value is *true*.

Example

This example sets the *editable* property to a value of *false* for the *my_ti* *TextInput* instance. This prevents the user from entering text in the instance. You can set the property to *true* to make the *TextInput* instance editable.

You must first drag a *TextInput* component to the Stage and give it an instance name of *my_ti*; then add the following code to Frame 1.

```
/**  
 * Requires:  
 * - TextInput instance on Stage (instance name: my_ti)  
 */  
  
var my_ti:mx.controls.TextInput;  
  
my_ti.editable = false;
```

TextInput.enter

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.enter = function(eventObject:Object) {
    //...
};
textInputInstance.addEventListener("enter", listenerObject);
```

Usage 2:

```
on (enter) {
    //...
}
```

Description

Event; notifies listeners that the Enter key has been pressed.

The first usage example uses an `on()` handler and must be attached directly to a `TextInput` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextInput`, sends “_level0.myTextInput” to the Output panel:

```
on (enter){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`textInputInstance`) dispatches an event (in this case, `enter`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

This example creates a listener for an enter event on a TextInput instance called `my_ti`. When the enter event occurs, if the user entered fewer than eight characters, the example displays:

You must enter at least 8 characters. Otherwise, it displays Thanks!

You must first drag a TextInput component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.handleEvent = function (evt_obj:Object){
    if (evt_obj.type == "enter"){
        if (my_ti.length < 8) {
            trace("You must enter at least 8 characters");
        } else {
            trace("Thanks");
        }
    }
}
// Add listener.
my_ti.addEventListener("enter", tiListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

TextInput.hPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.hPosition

Description

Property; specifies how many pixels have been scrolled to accommodate the user's entry in the TextInput box. The default value is 0.

NOTE

The value changes for the same text on different computers because of monitor, screen size, and font characteristics.

Example

The following example creates a listener for a change event on the TextInput instance called `my_ti`. The listener accesses the `hPosition` property to display the current position for each character the user enters.

You must first drag a TextInput component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
    trace("hPosition = " + my_ti.hPosition);
};
// Add listener.
my_ti.addEventListener("change", tiListener);
```

TextInput.length

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.length

Description

Read-only property; a number that indicates the number of characters in a TextInput component. A character such as tab ("\t") counts as one character. The default value is 0.

Example

The following example creates a listener for the change event on the TextInput instance `my_ti`. The listener accesses the `length` property to display the length of the text in `my_ti` as the user enters text.

You must first drag a TextInput component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
    trace("Length of text: " + my_ti.length);
};
// Add listener.
my_ti.addEventListener("change", tiListener);
```

TextInput.maxChars

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.maxChars

Description

Property; the maximum number of characters that the text field can contain. A script may insert more text than the `maxChars` property allows; this property indicates only how much text a user can enter. If this property is `null`, there is no limit to the amount of text a user can enter. The default value is `null`.

Example

The following example limits to eight the number of characters a user can enter in the `TextInput` instance called `my_ti`. It also sets the `password` property, which hides the input characters by displaying an asterisk in place of the character that was entered.

You must first drag a `TextInput` component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

my_ti.maxChars = 8;
my_ti.password = true;
```

TextInput.maxHPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.maxHPosition

Description

Read-only property; the value of `maxHPosition` is the pixel position of the character that is visible when the pointer has been moved to the very right of the text. It's not the last character's pixel position. Rather, it's the pixel position all the way to the right of the last character in the `TextInput` field. The default value is 0.

You must first drag a `TextInput` component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

Example

The following code creates a listener for a change event on the `TextInput` instance called `my_ti`. When the change event occurs, the listener displays the current `hPosition` and `maxHPosition` values for each character that the user enters:

```
/**
 * Requires:
 * - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Create listener object.
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
    trace("hPosition: " + my_ti.hPosition + " of " + my_ti.maxHPosition);
};
// Add listener.
my_ti.addEventListener("change", tiListener);
```

TextInput.password

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.password

Description

Property; a Boolean value indicating whether the text field is a password field (`true`) or not (`false`). If this property is `true`, the text field is a password text field and hides the input characters. If this property is `false`, the text field is not a password text field. The default value is `false`.

Example

The following example sets the `password` property to display an asterisk in place of the character that the user enters in the `TextInput` instance called `my_ti`. It also sets `maxChars` to limit the maximum number of characters the user can enter to eight.

You must first drag a `TextInput` component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

my_ti.maxChars = 8;
my_ti.password = true;
```


TextInput.restrict

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.restrict

Description

Property; indicates the set of characters that a user can enter in the text field. The default value is undefined. If this property is null or an empty string (""), a user can enter any character. If this property is a string of characters, the user can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (-).

If the string begins with ^, all characters that follow the ^ are considered unacceptable characters. If the string does not begin with ^, the characters in the string are considered acceptable. The ^ can also be used as a toggle between acceptable and unacceptable characters.

For example, the following code allows A-Z except X and Q:

```
Ta.restrict = "A-Z^XQ";
```

You can use the backslash (\) to enter a hyphen (-), caret (^), or backslash (\) character, as shown here:

```
\^  
\-  
\\
```

When you enter the \ character in the Actions panel within double quotation marks, it has a special meaning for the Actions panel's double-quote interpreter. It signifies that the character following the \ should be treated as is. For example, you could use the following code to enter a single quotation mark:

```
var leftQuote = "\'";
```

The Actions panel's restrict interpreter also uses \ as an escape character. Therefore, you may think that the following should work:

```
myText.restrict = "0-9-\^\\";
```

However, since this expression is surrounded by double quotation marks, the following value is sent to the restrict interpreter: 0-9-^\, and the restrict interpreter doesn't understand this value.

Because you must enter this expression within double quotation marks, you must not only provide the expression for the restrict interpreter, but you must also escape the Actions panel's built-in interpreter for double quotation marks. To send the value `0-9\-\^\` to the restrict interpreter, you must enter the following code:

```
myText.restrict = "0-9\\-\\^\\\\";
```

The `restrict` property restricts only user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

The following example provides three different uses of the `restrict` property. The first usage restricts input to uppercase characters A through Z, spaces, and numbers. The second usage allows any characters except the lowercase characters a through z. The third usage allows only numbers, -, ^, and \.

You must first drag a `TextInput` component to the Stage and give it an instance name of `my_ti`; then add the code to Frame 1, using only one of the following `restrict` statements at a time.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

// Example 1: Allow only uppercase A-Z, spaces, and digits 0-9.
my_ti.restrict = "A-Z 0-9";

// Example 2: Allow everything EXCEPT lowercase a-z.
my_ti.restrict = "^a-z";

// Example 3: Allow only digits 0-9, dash (-), ^, and \
my_ti.restrict = "0-9\\-\\^\\\\";
```

TextInput.text

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.text

Description

Property; the text contents of a TextInput component. The default value is "" (an empty string).

Example

The following code places a string in the TextInput instance called `my_ti`, and then traces that string to the Output panel.

You must first drag a TextInput component to the Stage and give it an instance name of `my_ti`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - TextInput instance on Stage (instance name: my_ti)
 */

var my_ti:mx.controls.TextInput;

my_ti.text = "The Royal Nonesuch";
trace(my_ti.text); // "The Royal Nonesuch"
```


ActionScript Class Name mx.data.to.TransferObject

The TransferObject interface defines a set of methods that items managed by the DataSet component must implement. The `DataSet.itemClassName` property specifies the name of the transfer object class that is instantiated each time a new item is needed. You can also specify this property for a selected DataSet component using the Property inspector.

Method summary for the TransferObject interface

The following table lists methods of the TransferObject interface.

Method	Description
<code>TransferObject.clone()</code>	Creates a new instance of the transfer object.
<code>TransferObject.getPropertyData()</code>	Returns the data for this transfer object.
<code>TransferObject.setPropertyData()</code>	Sets the data for this transfer object.

TransferObject.clone()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
class itemClass implements mx.data.to.TransferObject {  
    function clone() {  
        // Your code here.  
    }  
}
```

Parameters

None.

Returns

A copy of the transfer object.

Description

Method; creates an instance of the transfer object. The implementation of this method creates a copy of the existing transfer object and its properties and then returns that object.

Example

The following function returns a copy of this transfer object with all of the properties set to the same values as the original:

```
class itemClass implements mx.data.to.TransferObject {  
    function clone():Object {  
        var copy:itemClass = new itemClass();  
        for (var p in this) {  
            copy[p]= this[p];  
        }  
        return(copy);  
    }  
}
```

TransferObject.getPropertyData()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
class itemClass implements mx.data.to.TransferObject {  
    function getPropertyData() {  
        // Your code here.  
    }  
}
```

Parameters

None.

Returns

An object.

Description

Method; returns the data for this transfer object. The implementation of this method can return an anonymous ActionScript object with properties and corresponding values.

Example

The following function returns an object named `internalData` that contains the properties and their values from the `Contact` object:

```
class Contact implements mx.data.to.TransferObject {  
    function getPropertyData():Object {  
        var internalData:Object = {name:name, readOnly:_readOnly, phone:phone,  
            zip:zip.zipPlus4};  
        return(internalData);  
    }  
}
```

TransferObject.setPropertyData()

Availability

Flash Player 7.

Edition

Flash MX 2004.

Usage

```
class yourClass implements TransferObject {
    function setPropertyData(propData) {
        // Your code here.
    }
}
```

Parameters

propData An object that contains the data assigned to this transfer object.

Returns

Nothing.

Description

Method; sets the data for this transfer object. The *propData* parameter is an object whose fields contain the data assigned by the DataSet component to this transfer object.

Example

The following function receives a *propData* parameter and applies the values of its properties to the properties of the Contact object:

```
class Contact implements mx.data.to.TransferObject {

    function setPropertyData(propData: Object):Void {
        _readOnly = propData.readOnly;
        phone = propData.phone;
        zip = new mx.data.types.ZipCode(data.zip);
    }

    public var name:String;
    public var phone:String;
    public var zip:ZipCode;
    private var _readOnly:Boolean; // indicates if immutable
}
```


ActionScript Class Name `mx.transitions.TransitionManager`

The `TransitionManager` class and the effect-defining transition-based classes allow you to quickly apply impressive transition animation effects to slides and movie clips.

As its name implies, the `TransitionManager` class manages transitions. It allows you to apply one of ten animation effects to slides and movie clips. When creating custom components of version 2 of the Macromedia Component Architecture, you can use `TransitionManager` to apply animation effects to movie clips in your component's visual interface. The transition effects are defined in a set of transition classes that all extend the base class `mx.transitions.Transition`. You apply transitions through an instance of a `TransitionManager` only; you do not instantiate them directly. The `TransitionManager` class implements animation events.

Using the TransitionManager class

To use the methods and properties of the `TransitionManager` class, you have two options for creating a new instance. The easiest is to call the `TransitionManager.start()` method, which creates a new `TransitionManager` instance, designates the target object, applies a transition with an easing method, and starts it in one call. The following code uses the `TransitionManager.start()` method:

```
mx.transitions.TransitionManager.start(myMovieClip_mc,  
    {type:mx.transitions.Zoom, direction:mx.transitions.Transition.IN,  
    duration:1, easing:mx.transitions.easing.Bounce.easeOut});
```

For more information about the `TransitionManager.start()` method, its use, and parameters, see [TransitionManager.start\(\) on page 1244](#).

You can also create a new instance of the `TransitionManager` class by using the `new` operator. You then designate the transition properties and start the transition effect in a second step by calling the `TransitionManager.startTransition()` method. The following code uses the `TransitionManager.startTransition()` method:

```
var myTransitionManager:mx.transitions.TransitionManager = new
    mx.transitions.TransitionManager(myMovieClip_mc);
myTransitionManager.startTransition({type:mx.transitions.Zoom,
    direction:Transition.IN, duration:1,
    easing:mx.transitions.easing.Bounce.easeOut});
```

TransitionManager class parameters

When you create a new instance of a `TransitionManager` class by using the `new` operator, you must designate a target movie clip in the `content` parameter for its constructor. The constructor for the `mx.transitions.TransitionManager` class has the following parameter name and type:

```
TransitionManager(content:MovieClip)
```

content is the movie clip object to which the `TransitionManager` instance applies a transition.

NOTE

If you create a `TransitionManager` instance by using the `new` operator, you must then designate the properties of the transition that you want to apply and follow with a call to start the transition using the `TransitionManager.startTransition()` method; otherwise, the transition is not applied to a movie clip or started. For details about the `TransitionManager.startTransition()` method, its use, and parameters, see [TransitionManager.startTransition\(\) on page 1246](#). A quick alternative to the two-step process of creating a `TransitionManager` instance is to simply call the `TransitionManager.start()` method; for more information, see [TransitionManager.start\(\) on page 1244](#). The `TransitionManager.start()` method allows you to create a `TransitionManager` instance, provide the target movie clip, and specify the transition properties in one call.

Specifying an easing class and method in a transition

When you create an instance of the `TransitionManager` class by using the `TransitionManager.start()` method, you use the easing property of the `transParam` parameter to specify a function or method that provides an easing calculation. For a full description of the available easing classes and methods see [“Specifying an easing class and method in a transition” on page 1238](#).

TransitionManager class summary

The following sections list the methods, properties, and events for the TransitionManager class.

Method summary for the TransitionManager class

The following table lists the methods of the TransitionManager class.

Method	Description
TransitionManager.start()	Creates a new TransitionManager instance, designates the target object, applies a transition, and starts the transition.
TransitionManager.startTransition()	Creates a transition instance and starts it.
TransitionManager.toString()	Returns the type of the TransitionManager as a string.

Property summary for the TransitionManager class

The following table lists the properties of the TransitionManager class.

Property	Description
TransitionManager.content	The movie clip instance to which TransitionManager is to apply a transition.
TransitionManager.contentAppearance	An object that contains the saved visual properties of the content (target movie clip) to which the transitions will be applied. This property is read-only.

Event summary for the TransitionManager class

The following table lists the events of the TransitionManager class.

Event	Description
TransitionManager.allTransitionsInDone	Broadcast by a TransitionManager instance when it completes a transition with a direction property of <code>mx.transitions.Transition.IN</code> .
TransitionManager.allTransitionsOutDone	Broadcast by a TransitionManager instance when it completes a transition with a direction property of <code>mx.transitions.Transition.OUT</code> .

TransitionManager.allTransitionsInDone

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.allTransitionsInDone = function(eventObj:Object) {
    // ...
};
transitionManagerInstance.addEventListener("allTransitionsInDone",
    listenerObject);
```

Description

Event; notifies listeners that the TransitionManager instance has completed all transitions that have a direction property of `mx.transitions.Transition.IN` and has removed them from the list of transitions it is to apply.

The usage example uses a dispatcher or listener event model. A TransitionManager instance (*transitionManagerInstance*) dispatches an event (in this case, `allTransitionsInDone`), and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. In this case, the `allTransitionsInDone` event provides a `target` property that contains the TransitionManager instance that fired the event, allowing you to use that instance and all its properties and methods within the code that receives the `allTransitionsInDone` event. For more information, see [Chapter 21, “EventDispatcher class,” on page 499](#).

Example

The following code assigns an object to listen for the `allTransitionsInDone` event and specifies the method to act as the handler for the event. When that method is called to handle the event, a transition with a direction property of `mx.transitions.Transition.IN` has already been completed.

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Iris, direction:Transition.IN,
    duration:1, easing:None.easeNone, startPoint:5, shape:Iris.CIRCLE});

var myListener:Object = new Object();
myListener.allTransitionsInDone = function(eventObj:Object) {
    trace("allTransitionsInDone event occurred.");
};
myTransitionManager.addEventListener("allTransitionsInDone", myListener);
```

See also

[Chapter 21, “EventDispatcher class,” on page 499](#)

TransitionManager.allTransitionsOutDone

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.allTransitionsOutDone = function(eventObj:Object) {
    // ...
};
transitionManagerInstance.addEventListener("allTransitionsOutDone",
    listenerObject);
```

Description

Event; notifies listeners that the `TransitionManager` instance has completed all transitions that have a direction property of “out” and has removed them from the list of transitions it is to apply.

The usage example uses a dispatcher or listener event model. A `TransitionManager` instance (`transitionManagerInstance`) dispatches an event (in this case, `allTransitionsOutDone`) and the event is handled by a function, also called a handler, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. In this case, the `allTransitionsInDone` event provides a `target` property that contains the instance of `TransitionManager` that fired the event, so you can use that instance and all its properties and methods within the code that receives the `allTransitionsInDone` event. For more information, see [Chapter 21, “EventDispatcher class,” on page 499](#).

Example

The following code assigns an object to listen for the `allTransitionsOutDone` event and specifies the method to act as the handler for the event. When that method is called to handle the event, a transition with a `direction` property of `mx.transitions.Transition.IN` has already completed.

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Iris, direction:Transition.OUT,
    duration:1, easing:None.easeNone,startPoint:5, shape:Iris.CIRCLE});

var myListener:Object = new Object();
myListener.allTransitionsOutDone = function(eventObj:Object) {
    trace("allTransitionsOutDone event occurred.");
};
myTransitionManager.addEventListener("allTransitionsOutDone", myListener);
```

See also

[Chapter 21, “EventDispatcher class,” on page 499](#)

TransitionManager.content

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

transitionManagerInstance.content

Description

Property; the movie clip instance to which TransitionManager is to apply a transition.

Example

The following example returns the movie clip object currently targeted by a TransitionManager instance:

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
var myMovieClip:MovieClip = myTransitionManager.content;
trace(myMovieClip._name);
```

TransitionManager.contentAppearance

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

transitionManagerInstance.contentAppearance

Description

Property (read-only); an object that contains a snapshot of the properties of the target movie clip of a `TransitionManager` instance before the transition is applied. This object is helpful for obtaining information about what property values you can expect the movie clip to return to after the transition completes. The object returned from

`TransitionManager.contentAppearance` contains a recording of the original corresponding settings of the target movie clips in the following properties: `_x`, `_y`, `_xscale`, `_yscale`, `_alpha`, `_rotation`, `_innerBounds`, `_outerBounds`, `_width`, `_height`, and `colorTransform`. These properties are saved, and the `TransitionManager.start()` or `TransitionManager.startTransition()` method is called.

Example

The following example calls `TransitionManager.contentAppearance()` to get the original property settings of the `TransitionManager` object's target movie clip before the transition is applied:

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Zoom, direction:Transition.OUT,
    duration:3, easing:Bounce.easeOut});

var myMovieClip:MovieClip = myTransitionManager.content;
var myOriginalMovieClipProps:Object =
    myTransitionManager.contentAppearance;

for (var prop in myOriginalMovieClipProps) {
    trace(myMovieClip._name + "." +prop+ " = "+myOriginalMovieClipProps[prop]);
}
```

TransitionManager.start()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
transitionManagerInstance.start(content, transParams)
```

Parameters

content The `MovieClip` object to which to apply the transition effect.

transParams A collection of parameters that are passed within an object.

The `transParams` object should contain a `type` parameter that indicates the transition effect class to be applied, followed by `direction`, `duration`, and `easing` parameters. In addition, you must include any parameters required by that transition effect class. For example, the `mx.transitions.Iris` transition effect class requires additional `startPoint` and `shape` parameters. So, in addition to the `type`, `duration`, and `easing` parameters that every transition requires, you would also add (to the `transParams` object) the `startPoint` and `shape` parameters that the `mx.transitions.Iris` effect requires. The following code adds `startPoint` and `shape` parameters to the `mx.transitions.Iris` effect:

```
{type:mx.transitions.Iris, direction:mx.transitions.Transition.IN,
  duration:5, easing:mx.transitions.easing.Bounce.easeOut,
  startPoint:5, shape:mx.transitions.Iris.CIRCLE}
```

To verify the additional required parameters for the transition class effect that you are specifying in the `transParam` object's `type` parameter, see the API for that transition class. For example, for more information about the `Blinds` transition class, see [“Blinds transition” on page 1250](#).

NOTE

The `transParams` object's `type` parameter should include the full class-package name of the classes specified for its parameters unless they are already imported by using the `import` statement. To avoid having to provide the full class-package name for all the `transParams` parameter collection, place the following `import` statements previously in your code to import all `mx.transitions` classes and all `mx.transitions.easing` classes:

```
import mx.transitions.*;
import mx.transitions.easing.*;
```

Returns

An instance of the `Transition` object that the `TransitionManager` instance is assigned to apply.

Description

Method; creates an instance of the `TransitionManager` class if one does not already exist, creates an instance of the specified transition class designated in the `transParams.type` parameter, and then starts the transition. The transition is applied to the slide or movie clip that is designated in the `content` parameter.

Example

The following code uses the `TransitionManager.start()` method to create an instance of `TransitionManager` and assigns an `Iris` transition to a movie clip called `img1_mc`. The `TransitionManager.start()` method contains two parameters. The first parameter, `content`, is the `MovieClip` object that the transition effect will be applied to. The second parameter for the `TransitionManager.start()` method, `transParam`, contains an object that holds a parameter collection. This object that contains a parameter collection first designates the type of transition effect with the `type` parameter, followed by the `direction`, `duration`, and `easing` parameters. The `type`, `direction`, `duration`, and `easing` parameters are required information for all `TransitionManager` effects. Following the `easing` parameter are any parameters that the transition type specifically requires. In the following example, the `Iris` transition is the type of transition, and the `Iris` transition requires the `startPoint` and `shape` parameters (for more information about the `Iris` transition parameters, see [“Iris transition” on page 1252](#)):

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Iris, direction:Transition.IN,
    duration:5, easing:Bounce.easeOut, startPoint:5, shape:Iris.CIRCLE});
```

TransitionManager.startTransition()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
transitionManagerInstance.startTransition(transParams)
```

Parameters

transParams A collection of parameters that are passed within an object.

The `transParams` object should contain a `type` parameter that indicates the transition effect class to apply, followed by `direction`, `duration`, and `easing` parameters. In addition, you must include any parameters required by the specified transition effect class. For example, the `mx.transitions.Iris` transition effect class requires additional `startPoint` and `shape` parameters. So, in addition to the `type`, `duration`, and `easing` parameters that every transition requires, you would also add (to the `transParams` object) the `startPoint` and `shape` parameters that the `mx.transitions.Iris` effect requires. The following code adds `startPoint` and `shape` parameters to the `mx.transitions.Iris` effect:

```
{type:mx.transitions.Iris, direction:mx.transitions.Transition.IN,
  duration:5, easing:mx.transitions.easing.Bounce.easeOut, startPoint:5,
  shape:mx.transitions.Iris.CIRCLE}
```

To verify the additional parameters required for the transition class effect that you are specifying in the `transParam` object's `type` parameter, see the API for that transition class. For example, for more information about the `Blinds` transition class, see [“Blinds transition” on page 1250](#).

NOTE

The `transParams` object's `type` parameter should include the full class-package name of the classes specified for its parameters unless they are already imported by using the `import` statement. To avoid having to provide the full class-package name for all the `transParams` parameter collection, place the following `import` statements previously in your code to import all `mx.transitions` classes and all `mx.transitions.easing` classes:

```
import mx.transitions.*;
import mx.transitions.easing.*;
```

Returns

An instance of the `Transition` object that the `TransitionManager` instance is assigned to apply.

Description

Method; creates and starts an instance of the specified transition class, which is applied to the slide or movie clip that the `TransitionManager` instance is assigned to affect. If a matching transition already exists, that transition is removed and a new transition is created and started.

Example

The following code imports the `TransitionManager` class and creates a new `TransitionManager` instance. Next, the `TransitionManager.startTransition()` method designates a `mx.transitions.Zoom` transition in its type parameter. The direction parameter indicates that the transition should move in the out direction by designating `mx.transitions.Transition.OUT`. The duration of the transition is 3 seconds. The easing is calculated by using the `mx.transitions.Bounce.easeOut()` method of the `Bounce` class. This effect causes the `img1_mc` movie clip to appear to zoom out in a bouncing motion until it disappears, with the entire effect lasting 3 seconds.

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Zoom, direction:Transition.OUT,
    duration:3, easing:Bounce.easeOut});
```

TransitionManager.toString()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

transitionManagerInstance.toString()

Returns

The following string is returned: "[TransitionManager]".

Description

Method; returns the `TransitionManager` object's type as a string.

Example

The following code instructs the `TransitionManager` instance to return a string indicating its type:

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
var myType:String = myTransitionManager.toString();
trace(myType);
```

Transition-based classes

Inheritance (Root class)

ActionScript Class Name mx.transitions.Transition

The Transition class is the base class for all transition classes. You do not use or access this class directly. It allows transition-based classes to share some common behaviors and properties that are accessed by an instance of the TransitionManager class. Transition-based classes define an effect that is applied over time to a movie clip or a slide.

Flash includes ten transitions that you can use to apply effects to movie clip objects. You can customize all the transitions by including optional easing methods, and most transitions accept several optional parameters that allow you to control particular aspects of its effect. *Easing* refers to gradual acceleration or deceleration during an animation, which makes your animations appear more realistic. For example, a ball might gradually increase its speed near the beginning of an animation but slow down before it arrives at a full stop at the end of the animation. Many equations exist for this acceleration and deceleration, which change the easing animation.

The transitions are used with the TransitionManager class. See [“TransitionManager class” on page 1237](#). You use the TransitionManager class to specify a transition and apply it to a movie clip object rather than calling it directly. For example, to apply a Zoom transition to a movie clip called img1_mc, you specify the Zoom transition class as the type parameter in `TransitionManager.start()`:

```
mx.transitions.TransitionManager.start(myMovieClip_mc,  
    {type:mx.transitions.Zoom, direction:mx.transitions.Transition.IN,  
    duration:1, easing:mx.transitions.easing.Bounce.easeOut});
```

Flash includes the following transitions:

Transition	Description
Blinds transition	Reveals the movie clip object by using appearing or disappearing rectangles.
Fade transition	Fades the movie clip object in or out.
Fly transition	Slides the movie clip object in from a specified direction.
Iris transition	Reveals or hides the movie clip object by using an animated mask of a square shape or a circle shape that zooms in or out.
Photo transition	Causes the movie clip object to appear or disappear like a photographic flash.
PixelDissolve transition	Reveals or hides the movie clip object by using randomly appearing or disappearing rectangles in a checkerboard pattern.
Rotate transition	Rotates the movie clip object.
Squeeze transition	Scales the movie clip object horizontally or vertically.
Wipe transition	Reveals or hides the movie clip object by using an animated mask of a shape that moves horizontally.
Zoom transition	Zooms the movie clip object in or out by scaling it in proportion.

NOTE

Transitions are available only in ActionScript 2.0.

Blinds transition

ActionScript Class Name `mx.transitions.Blinds`

Parameters

numStrips The number of masking strips in the Blinds effect. The recommended range is 1 to 50.

dimension An integer that indicates that the Blinds strips are to be vertical (0) or horizontal (1).

Description

A transition effect: Reveals the movie clip object by using appearing or disappearing rectangles.

This class is used by specifying `mx.transitions.Blinds` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the Blinds transition with ten `numStrips` and a `dimension` integer specified as vertical (0). The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` instance applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with no easing.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Blinds, direction:Transition.IN,
    duration:2, easing:None.easeNone, numStrips:10, dimension:0});
```

Fade transition

ActionScript Class Name `mx.transitions.Fade`

Description

A transition effect: Fades the movie clip object in or out.

This class is used by specifying `mx.transitions.Fade` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the Fade transition. The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` instance applies a direction of `mx.transitions.Transition.IN` over a duration of 3 seconds with no easing.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Fade, direction:Transition.IN,
    duration:3, easing:None.easeNone});
```

Fly transition

ActionScript Class Name `mx.transitions.Fly`

Parameters

startPoint An integer that indicates a starting position; the range is 1 to 9:

Top Left, 1; Top Center, 2; Top Right, 3; Left Center, 4; Center, 5; Right Center, 6;
Bottom Left, 7; Bottom Center, 8; Bottom Right, 9.

Description

A transition effect: Slides the movie clip object in from a specified direction.

This class is used by specifying `mx.transitions.Fly` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the `Fly` transition with a `startPoint` set to the bottom right (9). The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` instance applies a direction of `mx.transitions.Transition.IN` over a duration of 3 seconds with an `Elastic.easeOut` easing effect.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Fly, direction:Transition.IN,
    duration:3, easing:Elastic.easeOut, startPoint:9});
```

Iris transition

ActionScript Class Name `mx.transitions.Iris`

Parameters

startPoint An integer indicating a starting position; the range is 1 to 9:

Top Left, 1; Top Center, 2, Top Right, 3; Left Center, 4; Center, 5; Right Center, 6; Bottom Left, 7; Bottom Center, 8, Bottom Right, 9.

shape A mask shape of either `mx.transitions.Iris.SQUARE` (a square) or `mx.transitions.Iris.CIRCLE` (a circle).

Description

A transition effect: Reveals the movie clip object by using an animated mask of a square shape or a circle shape that zooms in or out.

This class is used by specifying `mx.transitions.Iris` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the Iris transition with a `startPoint` from the center (5) and a masking shape of `mx.transitions.Iris.CIRCLE`. The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with an easing of `Strong` with an emphasis on the `easeOut` by specifying the `mx.transitions.easing.Strong.easeOut` easing calculation method.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Iris, direction:Transition.IN,
    duration:2, easing:Strong.easeOut, startPoint:5, shape:Iris.CIRCLE});
```

Photo transition

ActionScript Class Name `mx.transitions.Photo`

Description

A transition effect: Makes the movie clip object appear or disappear like a photographic flash. This class is used by specifying `mx.transitions.Photo` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the Photo transition to a content target movie clip `img1_mc`. The `TransitionManager` class applies a direction of `mx.transitions.Transition.IN` over a duration of 1 second with no easing.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start (img1_mc, {type:Photo, direction:Transition.IN,
    duration:1, easing:None.easeNone});
```

PixelDissolve transition

ActionScript Class Name `mx.transitions.PixelDissolve`

Parameters

`xSections` An integer that indicates the number of masking rectangle sections along the horizontal axis. The recommended range is 1 to 50.

`ySections` An integer that indicates the number of masking rectangle sections along the vertical axis. The recommended range is 1 to 50.

Description

A transition effect: Reveals the movie clip object by using randomly appearing or disappearing rectangles in a checkerboard pattern.

This class is used by specifying `mx.transitions.PixelDissolve` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the `PixelDissolve` transition with ten `xSections` and ten `ySections`. The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` instance applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with no easing.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:PixelDissolve,
    direction:Transition.IN, duration:2, easing:None.easeNone, xSections:10,
    ySections:10});
```

Rotate transition

ActionScript Class Name `mx.transitions.Rotate`

Parameters

ccw A Boolean value: *false* for clockwise rotation; *true* for counter-clockwise rotation.

degrees An integer that indicates the number of degrees the object is to be rotated. The recommended range is 1 to 9999. For example, a *degrees* setting of 1080 would rotate the object completely three times.

Description

A transition effect: Rotates the movie clip object.

This class is used by specifying `mx.transitions.Rotate` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the `Rotate` transition clockwise 720 degrees (two full revolutions). The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` instance applies a direction of `mx.transitions.Transition.IN` over a duration of 3 seconds with an easing set to `Strong.easeInOut` so that the transition starts slowly, speeds up, and then ends slowly.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Rotate, direction:Transition.IN,
    duration:3, easing:Strong.easeInOut, ccw:false, degrees:720});
```

Squeeze transition

ActionScript Class Name `mx.transitions.Squeeze`

Parameters

dimension An integer that indicates the Squeeze effect should be horizontal (0) or vertical (1).

Description

A transition effect: Scales the movie clip object horizontally or vertically.

This class is used by specifying `mx.transitions.Squeeze` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the `Squeeze` transition with a `dimension` integer specified as vertical (1). The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with an `Elastic` easing effect in the direction of `easeOut`.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Squeeze, direction:Transition.IN,
    duration:2, easing:Elastic.easeOut, dimension:1});
```

Wipe transition

ActionScript Class Name `mx.transitions.Wipe`

Parameters

startPoint An integer that indicates a starting position. Range of 1 to 4 and 6 to 9:

Top Left, 1; Top Center, 2; Top Right, 3; Left Center, 4; Right Center, 6; Bottom Left, 7; Bottom Center, 8; Bottom Right, 9.

Description

A transition effect: Reveals or hides the movie clip object by using an animated mask of a shape that moves horizontally.

This class is used by specifying `mx.transitions.Wipe` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies the `Wipe` transition with a `startPoint` from the top left (1). The content target of the transition is the movie clip `img1_mc`. The `TransitionManager` applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with no easing.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Wipe, direction:Transition.IN,
    duration:2, easing:None.easeNone, startPoint:1});
```

Zoom transition

ActionScript Class Name `mx.transitions.Zoom`

Description

A transition effect: Zooms the movie clip object in or out by scaling it in proportion.

This class is used by specifying `mx.transitions.Zoom` as a `transObject.type` parameter for the `TransitionManager` class.

Example

The following code creates an instance of `TransitionManager` that applies a `Zoom` transition to the content target movie clip `img1_mc`. The `TransitionManager` applies a direction of `mx.transitions.Transition.IN` over a duration of 2 seconds with an `Elastic` type easing that starts fast and eases slowly at the finish.

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Zoom, direction:Transition.IN,
    duration:2, easing:Elastic.easeOut});
```

TreeDataProvider interface (Flash Professional only)

The `TreeDataProvider` interface is a set of properties and methods and does not need to be instantiated to be used. If a `Tree` class is packaged in a SWF file, all XML instances in the SWF file contain the `TreeDataProvider` interface. All nodes in a tree are XML objects that contain the `TreeDataProvider` interface.

It's best to use the `TreeDataProvider` methods to create XML for the `Tree.dataProvider` property, because only `TreeDataProvider` broadcasts events that refresh the tree's display. These are events that the `Tree` class handles; you do not need to write functions to handle these events. (The built-in XML class methods don't broadcast such events.)

Use the `TreeDataProvider` methods to control the data model and the data display. Use built-in XML class methods for read-only tasks such as traversing through the tree hierarchy.

You can select the property that holds the text to be displayed by specifying a `labelField` or `labelFunction` property. For example, the code `myTree.labelField = "firstName";` results in the value of the property `myTreeDP.attributes.fred` being queried for the display text.

Method summary for the `TreeDataProvider` interface

The following table lists the methods of the `TreeDataProvider` interface.

Method	Description
<code>TreeDataProvider.addNode()</code>	Adds a child node at the root of the tree.
<code>TreeDataProvider.addNodeAt()</code>	Adds a child node at a specified location on the parent node.
<code>TreeDataProvider.getNodeAt()</code>	Returns the specified child of a node.
<code>TreeDataProvider.removeTreeNode()</code>	Removes a node and all the node's descendants from the node's parent.
<code>TreeDataProvider.removeTreeNodeAt()</code>	Removes a node and all the node's descendants from the index position of the child node.

Property summary for the TreeDataProvider interface

The following table lists the properties of the TreeDataProvider interface.

Property	Description
TreeDataProvider.attributes.data	Specifies the data to associate with a node.
TreeDataProvider.attributes.label	Specifies the text to be displayed next to a node.

TreeDataProvider.addTreeNode()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
someNode.addTreeNode(label, data)
```

Usage 2:

```
someNode.addTreeNode(child)
```

Parameters

label A string that displays the node.

data An object of any type that is associated with the node.

child Any XMLNode object.

Returns

The added XML node.

Description

Method; adds a child node at the root of the tree. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt child node, which is an XMLNode object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the display of the tree.

Example

The first line of code in the following example locates the node to which to add a child. The second line adds a new node to a specified node.

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.addTreeNode("Inbox", 3);
```

The following code moves a node from one tree to the root of another tree:

```
myTreeNode.addTreeNode(mySecondTree.getTreeNodeAt(3));
```

TreeDataProvider.addTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
someNode.addTreeNodeAt(index, label, data)
```

Usage 2:

```
someNode.addTreeNodeAt(index, child)
```

Parameters

index An integer that indicates the index position (among the child nodes) at which the node should be added.

label A string that displays the node.

data An object of any type that is associated with the node.

child Any XMLNode object.

Returns

The added XML node.

Description

Method; adds a child node at the specified location in the parent node. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt child node, which is an XMLNode object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the display of the tree.

Example

The following code locates the node to which you will add a node and adds a new node as the second child of the root:

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.addTreeNodeAt(1, "Inbox", 3);
```

The following code moves a node from one tree to become the fourth child of the root of another tree:

```
myTreeNode.addTreeNodeAt(3, mySecondTree.getTreeNodeAt(3));
```

TreeDataProvider.attributes.data

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
someNode.attributes.data
```

Description

Property; specifies the data to associate with the node. This adds the value as an attribute in the XMLNode object. Setting this property does not refresh any tree displays. This property can be of any data type.

Example

The following code locates the node to adjust and sets its *data* property:

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.attributes.data = "hi"; // results in <node data = "hi">;
```

See also

[TreeDataProvider.attributes.label](#)

TreeDataProvider.attributes.label

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
someNode.attributes.label
```

Description

Property; a string that specifies the text displayed for the node. This is written to an attribute of the XMLNode object. Setting this property does not refresh any tree displays.

Example

The following code locates the node to adjust and sets its `label` property. The result of the following code is `<node label="Mail">`.

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.attributes.label = "Mail";
```

See also

[TreeDataProvider.attributes.data](#)

TreeDataProvider.getTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
someNode.getTreeNodeAt(index)
```

Parameters

index An integer representing the position of the child node in the current node.

Returns

The specified node.

Description

Method; returns the specified child node of the node.

Example

The following code locates a node and then retrieves the second child of `myTreeNode`:

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.getTreeNodeAt(1);
```

TreeDataProvider.removeTreeNode()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
someNode.removeTreeNode()
```

Parameters

None.

Returns

The removed XML node, or `undefined` if an error occurs.

Description

Method; removes the specified node, and any of its descendants, from the node's parent.

Example

The following code removes a node:

```
myTreeDP.firstChild.removeTreeNode();
```

TreeDataProvider.removeTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

someNode.removeTreeNodeAt(*index*)

Parameters

index An integer indicating the position of the node to be removed.

Returns

The removed XML node, or `undefined` if an error occurs.

Description

Method; removes a node (and all of its descendants) specified by the current node and index position of the child node. Calling this method refreshes the view.

Example

The following code removes the fourth child of the `myTreeDP.firstChild` node:

```
myTreeDP.firstChild.removeTreeNodeAt(3);
```


Tree component (Flash Professional only)

The Tree component allows a user to view hierarchical data. The tree appears in a box like the List component, but each item in a tree is called a *node* and can be either a *leaf* or a *branch*. By default, a leaf is represented by a text label beside a file icon, and a branch is represented by a text label beside a folder icon with an expander arrow (disclosure triangle) that a user can open to display child nodes. The children of a branch can be leaves or branches.

The data of a tree component must be provided from an XML data source. For more information, see [“Using the Tree component \(Flash Professional only\)” on page 1266](#).

When a Tree instance has focus either from clicking or tabbing, you can use the following keys to control it:

Key	Description
Down Arrow	Moves selection down one item.
Up Arrow	Moves selection up one item.
Right Arrow	Opens a selected branch node. If a branch is already open, moves to first child node.
Left Arrow	Closes a selected branch node. If on a leaf node of a closed branch node, moves to parent node.
Space	Opens or closes a selected branch node.
End	Moves selection to the bottom of the list.
Home	Moves selection to the top of the list.
Page Down	Moves selection down one page.
Page Up	Moves selection up one page.
Control	Allows multiple noncontiguous selections.
Shift	Allows multiple contiguous selections.

The Tree component cannot be made accessible to screen readers.

Using the Tree component (Flash Professional only)

The Tree component can be used to represent hierarchical data such as e-mail client folders, file browser panes, or category browsing systems for inventory. Most often, the data for a tree is retrieved from a server in the form of XML, but it can also be well-formed XML that is created during authoring in Flash. The best way to create XML for the tree is to use the `TreeDataProvider` interface. You can also use the `ActionScript XML` class or build an XML string. After you create an XML data source (or load one from an external source), you assign it to `Tree.dataProvider`.

The Tree component comprises two sets of APIs: the `Tree` class and the `TreeDataProvider` interface. The `Tree` class contains the visual configuration methods and properties. The `TreeDataProvider` interface lets you construct XML and add it to multiple tree instances. A `TreeDataProvider` object broadcasts changes to any trees that use it. In addition, any XML or `XMLNode` object that exists on the same frame as a tree or a menu is automatically given the `TreeDataProvider` methods and properties. For more information, see “[TreeDataProvider interface \(Flash Professional only\)](#)” on page 1257.

Formatting XML for the Tree component

The Tree component is designed to display hierarchical data structures using XML as the data model. It is important to understand the relationship of the XML data source to the Tree component.

Consider the following XML data source sample:

```
<node>
  <node label="Mail">
    <node label="INBOX"/>
    <node label="Personal Folder">
      <node label="Business" isBranch="true" />
      <node label="Demo" isBranch="true" />
      <node label="Personal" isBranch="true" />
      <node label="Saved Mail" isBranch="true" />
      <node label="bar" isBranch="true" />
    </node>
    <node label="Sent" isBranch="true" />
    <node label="Trash"/>
  </node>
</node>
```

NOTE

The `isBranch` attribute is read-only; you cannot set it directly. To set it, call `Tree.setIsBranch()`.

Nodes in the XML data source can have any name. Notice in the previous example that each node is named with the generic name `node`. The tree reads through the XML and builds the display hierarchy according to the nested relationship of the nodes.

Each XML node can be displayed as one of two types in the tree: branch or leaf. Branch nodes can contain multiple child nodes and appear as a folder icon with an expander arrow that allows users to open and close the folder. Leaf nodes appear as a file icon and cannot contain child nodes. Both leaves and branches can be roots; a root node appears at the top level of the tree and has no parent. The icons are customizable; for more information, see [“Using skins with the Tree component” on page 1278](#).

There are many ways to structure XML, but the Tree component cannot use all types of XML structures. Do not nest node attributes in a child node; each node should contain all its necessary attributes. Also, the attributes of each node should be consistent to be useful. For example, to describe a mailbox structure with a Tree component, use the same attributes on each node (message, data, time, attachments, and so on). This lets the tree know what it expects to render, and lets you loop through the hierarchy to compare data.

When a Tree displays a node, it uses the `label` attribute of the node by default as the text label. If any other attributes exist, they become additional properties of the node's attributes within the tree.

The actual root node is interpreted as the Tree component itself. This means that the first child (in the previous example, `<node label="Mail">`), is rendered as the root node in the tree view. This means that a tree can have multiple root folders. In the example, there is only one root folder displayed in the tree: Mail. However, if you were to add sibling nodes at that level in the XML, multiple root nodes would be displayed in the tree.

A data provider for a tree always wants a node that has children it can display. It displays the first child of the `XMLNode` object. When the XML is wrapped in an XML object, the structure looks like the following:

```
<XMLDocumentObject>
  <node>
    <node label="Mail">
      <node label="INBOX"/>
      <node label="Personal Folder">
        <node label="Business" isBranch="true" />
        <node label="Demo" isBranch="true" />
        <node label="Personal" isBranch="true" />
        <node label="Saved Mail" isBranch="true" />
        <node label="bar" isBranch="true" />
      </node>
      <node label="Sent" isBranch="true" />
      <node label="Trash"/>
    </node>
  </node>
</XMLDocumentObject>
```

Flash Player wraps the XML nodes in an extra document node, which is passed to the tree. When the tree tries to display the XML, it tries to display `<node>`, which doesn't have a label, so it doesn't display correctly.

To avoid this problem, the data provider for the Tree component should point at the `XMLDocumentObject`'s first child, as shown here:

```
myTree.dataProvider = myXML.firstChild;
```

Tree parameters

You can set the following authoring parameters for each Tree component instance in the Property inspector or the Component inspector:

multipleSelection is a Boolean value that indicates whether a user can select multiple items (`true`) or not (`false`). The default value is `false`.

rowHeight indicates the height of each row, in pixels. The default value is 20.

You can write ActionScript to control these and additional options for the Tree component using its properties, methods, and events. For more information, see [“Tree class \(Flash Professional only\)” on page 1278](#).

You cannot enter data parameters in the Property inspector or the Component inspector for the Tree component as you can with other components. For more information, see [“Using the Tree component \(Flash Professional only\)” on page 1266](#) and [“Creating an application with the Tree component” on page 1268](#).

Creating an application with the Tree component

The following procedures show how to use a Tree component to display mailboxes in an e-mail application.

The Tree component does not allow you to enter data parameters in the Property inspector or Component inspector. Because of the complexity of a Tree component's data structure, you must either import an XML object at runtime or build one in Flash while authoring. To create XML in Flash, you can use the `TreeDataProvider` interface, use the ActionScript XML object, or build an XML string. Each of these options is explained in the following procedures.

To add a Tree component to an application and load XML:

1. In Flash, select `File > New` and select `Flash Document`.
2. Save the document as `treeMenu fla`.
3. In the Components panel, double-click the Tree component to add it to the Stage.

4. Select the Tree instance. In the Property inspector, enter the instance name **menuTree**.
5. Select the Tree instance and press F8. Select Movie Clip, and enter the name **TreeNavMenu**.
6. Click the Advanced button, and select Export for ActionScript.
7. Type **TreeNavMenu** in the AS 2.0 Class text box and click OK.
8. Select File > New and select ActionScript File.
9. Save the file as **TreeNavMenu.as** in the same directory as **treeMenu.fla**.
10. In the Script window, enter the following code:

```
import mx.controls.Tree;

class TreeNavMenu extends MovieClip {
    var menuXML:XML;
    var menuTree:Tree;
    function TreeNavMenu() {
        // Set up the appearance of the tree and event handlers.
        menuTree.setStyle("fontFamily", "_sans");
        menuTree.setStyle("fontSize", 12);
        // Load the menu XML.
        var treeNavMenu = this;
        menuXML = new XML();
        menuXML.ignoreWhite = true;
        menuXML.load("TreeNavMenu.xml");
        menuXML.onLoad = function() {
            treeNavMenu.onMenuLoaded();
        };
    }
    function change(event:Object) {
        if (menuTree == event.target) {
            var node = menuTree.selectedItem;
            // If this is a branch, expand/collapse it.
            if (menuTree.getIsBranch(node)) {
                menuTree.setIsOpen(node, !menuTree.getIsOpen(node), true);
            }
            // If this is a hyperlink, jump to it.
            var url = node.attributes.url;
            if (url) {
                getURL(url, "_top");
            }
            // Clear any selection.
            menuTree.selectedNode = null;
        }
    }
    function onMenuLoaded() {
        menuTree.dataProvider = menuXML.firstChild;
        menuTree.addEventListener("change", this);
    }
}
```

This ActionScript sets up styles for the tree. An XML object is created to load the XML file that creates the tree's nodes. Then the `onLoad` event handler is defined to set the data provider to the contents of the XML file.

11. Create a new file called `TreeNavMenu.xml` in a text editor.

12. Enter the following code in the file:

```
<node>
  <node label="My Bookmarks">
    <node label="Macromedia Web site" url="http://www.macromedia.com" />
    <node label="MXNA blog aggregator" url="http://www.markme.com/mxna"
  />
  </node>
  <node label="Google" url="http://www.google.com" />
</node>
```

13. Save your documents and return to `treeMenu fla`. Select `Control > Test Movie` to test the application.

To load XML from an external file:

1. In Flash, select `File > New` and select `Flash Document`.
2. Drag an instance of the `Tree` component onto the Stage.
3. Select the `Tree` instance. In the `Property inspector`, enter the instance name `myTree`.
4. In the `Actions panel` on `Frame 1`, enter the following code:

```
var myTreeDP:XML = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("treeXML.xml");
myTreeDP.onLoad = function() {
  myTree.dataProvider = this.firstChild;
};
var treeListener:Object = new Object();
treeListener.change = function(evt:Object) {
  trace("selected node is: "+evt.target.selectedNode);
  trace("");
};
myTree.addEventListener("change", treeListener);
```

This code creates an XML object called `myTreeDP` and calls the `XML.load()` method to load an XML data source. The code then defines an `onLoad` event handler that sets the `dataProvider` property of the `myTree` instance to the new XML object when the XML loads. For more information about the XML object, see its entry in *ActionScript 2.0 Language Reference*.

5. Create a new file called `treeXML.xml` in a text editor.

6. Enter the following code in the file:

```
<node>
  <node label="Mail">
    <node label="INBOX"/>
    <node label="Personal Folder">
      <node label="Business" isBranch="true" />
      <node label="Demo" isBranch="true" />
      <node label="Personal" isBranch="true" />
      <node label="Saved Mail" isBranch="true" />
      <node label="bar" isBranch="true" />
    </node>
    <node label="Sent" isBranch="true" />
    <node label="Trash"/>
  </node>
</node>
```

7. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the change event handler send the data values to the Output panel.

To use the `TreeDataProvider` class to create XML in Flash while authoring:

- 1. In Flash, select File > New and select Flash Document.**
- 2. Drag an instance of the Tree component onto the Stage.**
- 3. Select the Tree instance and in the Property inspector, enter the instance name `myTree`.**
- 4. In the Actions panel on Frame 1, enter the following code:**

```
var myTreeDP:XML = new XML();
myTreeDP.addTreeNode("Local Folders", 0);
// Use XML.firstChild to nest child nodes below Local Folders.
var myTreeNode:XMLNode = myTreeDP.firstChild;
myTreeNode.addTreeNode("Inbox", 1);
myTreeNode.addTreeNode("Outbox", 2);
myTreeNode.addTreeNode("Sent Items", 3);
myTreeNode.addTreeNode("Deleted Items", 4);
// Assign the myTreeDP data source to the myTree component.
myTree.dataProvider = myTreeDP;
// Set each of the four child nodes to be branches.
for (var i = 0; i<myTreeNode.childNodes.length; i++) {
  var node:XMLNode = myTreeNode.getTreeNodeAt(i);
  myTree.setIsBranch(node, true);
}
```

This code creates an XML object called `myTreeDP`. Any XML object on the same frame as a Tree component automatically receives all the properties and methods of the `TreeDataProvider` interface. The second line of code creates a single root node called Local Folders. For detailed information about the rest of the code, see the comments (lines preceded with `//`) throughout the code.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the change event handler send the data values to the Output panel.

To use the ActionScript XML class to create XML:

1. In Flash, select File > New and select Flash Document.
2. Drag an instance of the Tree component onto the Stage.
3. Select the Tree instance. In the Property inspector, enter the instance name `myTree`.
4. In the Actions panel on Frame 1, enter the following code:

```
// Create an XML object.
var myTreeDP:XML = new XML();
// Create node values.
var myNode0:XMLNode = myTreeDP.createElement("node");
myNode0.attributes.label = "Local Folders";
myNode0.attributes.data = 0;
var myNode1:XMLNode = myTreeDP.createElement("node");
myNode1.attributes.label = "Inbox";
myNode1.attributes.data = 1;
var myNode2:XMLNode = myTreeDP.createElement("node");
myNode2.attributes.label = "Outbox";
myNode2.attributes.data = 2;
var myNode3:XMLNode = myTreeDP.createElement("node");
myNode3.attributes.label = "Sent Items";
myNode3.attributes.data = 3;
var myNode4:XMLNode = myTreeDP.createElement("node");
myNode4.attributes.label = "Deleted Items";
myNode4.attributes.data = 4;
// Assign nodes to the hierarchy in the XML tree.
myTreeDP.appendChild(myNode0);
myTreeDP.firstChild.appendChild(myNode1);
myTreeDP.firstChild.appendChild(myNode2);
myTreeDP.firstChild.appendChild(myNode3);
myTreeDP.firstChild.appendChild(myNode4);
// Assign the myTreeDP data source to the Tree component.
myTree.dataProvider = myTreeDP;
```

For more information about the XML object, see its entry in *ActionScript 2.0 Language Reference*.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the change event handler send the data values to the Output panel.

To use a well-formed string to create XML in Flash while authoring:

1. In Flash, select File > New and select Flash Document.
2. Drag an instance of the Tree component onto the Stage.
3. Select the Tree instance. In the Property inspector, enter the instance name **myTree**.
4. In the Actions panel on Frame 1, enter the following code:

```
var myTreeDP:XML = new XML("<node label='Local Folders'><node  
  label='Inbox' data='0'><node label='Outbox' data='1'></node>");  
myTree.dataProvider = myTreeDP;
```

This code creates the XML object `myTreeDP` and assigns it to the `dataProvider` property of `myTree`.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the change event handler send the data values to the Output panel.

Customizing the Tree component (Flash Professional only)

You can transform a Tree component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)). When a tree isn't wide enough to display the text of the nodes, the text is clipped.

Using styles with the Tree component

A Tree component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color of the list. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to "false". The default value is 0xDDDDDD (medium gray).
<code>depthColors</code>	Both	Sets the background colors for rows based on the depth of each node. The value is an array of colors where the first element is the background color for the root node, the second element is the background color for its children, and so on, continuing through the number of colors provided in the array. This style property is not set by default.
<code>borderStyle</code>	Both	The Tree component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See " RectBorder class " on page 1063. The default border style is "inset".
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.

Style	Theme	Description
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
textAlign	Both	The text alignment: either "left", "right", or "center". The default value is "left".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".
textIndent	Both	A number indicating the text indent. The default value is 0.
defaultLeafIcon	Both	The icon displayed in a Tree control for leaf nodes when no icon is specified for a particular node. The default value is "TreeNodeIcon", which is an image representing a piece of paper.
disclosureClosedIcon	Both	The icon displayed next to a closed folder node in a Tree component. The default value is "TreeDisclosureClosed", which is a gray arrow pointing to the right.
disclosureOpenIcon	Both	The icon displayed next to an open folder node in a Tree component. The default value is "TreeDisclosureOpen", which is a gray arrow pointing down.
folderClosedIcon	Both	The icon displayed for a closed folder node in a Tree component if no node-specific icon is set. The default value is "TreeFolderClosed", which is a yellow closed file folder image.
folderOpenIcon	Both	The icon displayed for an open folder node in a Tree component if no node-specific icon is set. The default value is "TreeFolderOpen", which is a yellow open file folder image.
indentation	Both	The number of pixels to indent each row of a Tree component. The default value is 17.
openDuration	Both	The duration, in milliseconds, of the expand and collapse animations. The default value is 250.

Style	Theme	Description
<code>openEasing</code>	Both	A reference to a tweening function that controls the expand and collapse animations. Defaults to sine in/out. For more information, see “Customizing component animations” in <i>Using Components</i> .
<code>repeatDelay</code>	Both	The number of milliseconds of delay between when a user first presses a button in the scrollbar and when the action begins to repeat. The default value is 500 (half a second).
<code>repeatInterval</code>	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button in the scrollbar. The default value is 35.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>0xE3FFD6</code> (bright green) with the Halo theme and <code>0xA9A9A9</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionColor</code>	Both	The background color of a selected row. The default value is <code>0xCDFFC1</code> (light green) with the Halo theme and <code>0xE9E9E9</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal state to a selected state or back from selected to normal, in milliseconds. The default value is 200.
<code>selectionDisabledColor</code>	Both	The background color of a selected row. The default value is <code>0xDDDDDD</code> (medium gray). Because the default value for this property is the same as the default for <code>backgroundDisabledColor</code> , the selection is not visible when the component is disabled unless one of these style properties is changed.

Style	Theme	Description
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. Applies only for the transition from a normal to a selected state. The default equation uses a sine in/out formula. For more information, see “Customizing component animations” in <i>Using Components</i> .
<code>textRollOverColor</code>	Both	The color of text when the pointer rolls over it. The default value is <code>0x2B333C</code> (dark gray). This style is important when you set <code>rollOverColor</code> , because the two must complement each other so that text is easily viewable during a rollover.
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two must complement each other so that text is easily viewable while selected.
<code>useRollOver</code>	Both	Determines whether rolling over a row activates highlighting. The default value is <code>true</code> .

For example, the following code creates a `Tree` instance `first_tr` using `UIObject.createClassObject()`, populates the tree using a data provider, and then uses `UIObject.setStyle()` to change the indentation of the nodes of the tree to 8 pixels. Drag a `Tree` component to the current document’s library and then add the following to the first frame of the main timeline:

```
import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 20);
first_tr.setSize(200, 100);
first_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folder'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /><node label='2nd
  Local Folder'><node label='Inbox' data='0' /><node label='Outbox'
  data='1' /></node></node>");
first_tr.dataProvider = trDP_xml;

first_tr.setStyle("indentation", 8);
```

Setting styles for all Tree components in a document

The `Tree` class inherits from the `List` class, which inherits from the `ScrollSelectList` class. The default class-level style properties are defined on the `ScrollSelectList` class, which the `Menu` component and all `List`-based components extend. You can set new default style values on this class directly, and the new settings are reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the `Tree` components only, you can create a new `CSSStyleDeclaration` instance and store it in `_global.styles.DataGrid`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Tree == undefined) {
    _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFF00AA);
```

When you create a new class-level style declaration, you lose all default values provided by the `ScrollSelectList` declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for` loop, as follows, to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Tree;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

Using skins with the Tree component

The `Tree` component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see [“RectBorder class” on page 1063](#) and [“Using skins with the UIScrollBar component” on page 1394](#).

Tree class (Flash Professional only)

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `ScrollSelectList` > [List component](#) > `Tree`

ActionScript Class Name `mx.controls.Tree`

The methods, properties, and events of the `Tree` class allow you to manage and manipulate `Tree` objects.

Method summary for the Tree class

The following table lists methods of the Tree class.

Method	Description
Tree.addNode()	Adds a node to a Tree instance.
Tree.addNodeAt()	Adds a node at a specific location in a Tree instance.
Tree.getDisplayIndex()	Returns the display index of a given node.
Tree.getIsBranch()	Specifies whether the folder is a branch (has a folder icon and an expander arrow).
Tree.getIsOpen()	Indicates whether a node is open or closed.
Tree.getNodeDisplayedAt()	Maps a display index of the tree onto the node that is displayed there.
Tree.getTreeNodeAt()	Returns a node on the root of the tree.
Tree.refresh()	Updates the tree.
Tree.removeAll()	Removes all nodes from a Tree instance and refreshes the tree.
Tree.removeTreeNodeAt()	Removes a node at a specified position and refreshes the tree.
Tree.setIcon()	Specifies an icon for the specified node.
Tree.setIsBranch()	Specifies whether a node is a branch (has a folder icon and expander arrow).
Tree.setIsOpen()	Opens or closes a node.

Methods inherited from the UIObject class

The following table lists the methods the Tree class inherits from the UIObject class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.

Method	Description
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Tree class inherits from the UIComponent class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the List class

The following table lists the methods the Tree class inherits from the List class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
<code>List.addItem()</code>	Adds an item to the end of the list.
<code>List.addItemAt()</code>	Adds an item to the list at the specified index. With the Tree component, it is better to use <code>Tree.addNodeAt()</code> .
<code>List.getItemAt()</code>	Returns the item at the specified index.
<code>List.removeAll()</code>	Removes all items from the list.
<code>List.removeItemAt()</code>	Removes the item at the specified index.
<code>List.replaceItemAt()</code>	Replaces the item at the specified index with another item.
<code>List.setPropertiesAt()</code>	Applies the specified properties to the specified item.
<code>List.sortItems()</code>	Sorts the items in the list according to the specified compare function.
<code>List.sortItemsBy()</code>	Sorts the items in the list according to a specified property.

Property summary for the Tree class

The following table lists properties of the Tree class.

Property	Description
Tree.dataProvider	Specifies an XML data source.
Tree.firstVisibleNode	Specifies the first node at the top of the display.
Tree.selectedNode	Specifies the selected node in a Tree instance.
Tree.selectedNodes	Specifies the selected nodes in a Tree instance.

Properties inherited from the UIObject class

The following table lists the properties the Tree class inherits from the UIObject class. When accessing these properties from the Tree object, use the form *TreeInstance.propertyName*.

Property	Description
UIObject.bottom	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
UIObject.height	Read-only; the height of the object, in pixels.
UIObject.left	Read-only; the left edge of the object, in pixels.
UIObject.right	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
UIObject.scaleX	A number indicating the scaling factor in the x direction of the object, relative to its parent.
UIObject.scaleY	A number indicating the scaling factor in the y direction of the object, relative to its parent.
UIObject.top	Read-only; the position of the top edge of the object, relative to its parent.
UIObject.visible	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
UIObject.width	Read-only; the width of the object, in pixels.
UIObject.x	Read-only; the left edge of the object, in pixels.
UIObject.y	Read-only; the top edge of the object, in pixels.

Properties inherited from the UIComponent class

The following table lists the properties the Tree class inherits from the UIComponent class. When accessing these properties from the Tree object, use the form

TreeInstance.propertyName.

Property	Description
UIComponent.enabled	Indicates whether the component can receive focus and input.
UIComponent.tabIndex	A number indicating the tab order for a component in a document.

Properties inherited from the List class

The following table lists the properties the Tree class inherits from the List class. When accessing these properties from the Tree object, use the form *TreeInstance.propertyName.*

Property	Description
List.cellRenderer	Assigns the class or symbol to use to display each row of the list.
List.dataProvider	The source of the list items.
List.hPosition	The horizontal position of the list.
List.hScrollPolicy	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
List.iconField	A field in each item to be used to specify icons.
List.iconFunction	A function that determines which icon to use.
List.labelField	Specifies a field of each item to be used as label text.
List.labelFunction	A function that determines which fields of each item to use for the label text.
List.length	The number of items in the list. This property is read-only.
List.maxHPosition	The number of pixels the list can scroll to the right, when List.hScrollPolicy is set to "on".
List.multipleSelection	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
List.rowCount	The number of rows that are at least partially visible in the list.
List.rowHeight	The pixel height of every row in the list.
List.selectable	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).

Property	Description
List.selectedIndex	The index of a selection in a single-selection list.
List.selectedIndices	An array of the selected items in a multiple-selection list.
List.selectedItem	The selected item in a single-selection list. This property is read-only.
List.selectedItems	The selected item objects in a multiple-selection list. This property is read-only.
List.vPosition	Scrolls the list so the topmost visible item is the number assigned.
List.vScrollPolicy	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Event summary for the Tree class

The following table lists events of the Tree class.

Event	Description
Tree.nodeClose	Broadcast when a node is closed by a user.
Tree.nodeOpen	Broadcast when a node is opened by a user.

Events inherited from the UIObject class

The following table lists the events the Tree class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Tree class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

Events inherited from the List class

The following table lists the events the Tree class inherits from the List class.

Event	Description
List.change	Broadcast whenever user interaction causes the selection to change.
List.itemRollOut	Broadcast when the pointer rolls over and then off list items.
List.itemRollOver	Broadcast when the pointer rolls over list items.
List.scroll	Broadcast when a list is scrolled.

Tree.addTreeNode()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
treeInstance.addTreeNode(label [, data])
```

Usage 2:

```
treeInstance.addTreeNode(child)
```

Parameters

label A string that displays the node, or an object with a label field (or whatever label field name is specified by the `labelField` property).

data An object of any type that is associated with the node. This parameter is optional.

child Any XMLNode object.

Returns

The added XML node.

Description

Method; adds a child node to the tree. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt child node, which is an XMLNode object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the view.

Example

The following example creates two Tree components with a node for each one, 1st Local Folders and 2nd Local Folders, respectively. Then it adds the tree node (2nd Local Folders) from the second Tree to the first Tree and also adds a new node, Inbox.

You must first add the component to the document library by dragging a Tree component to the Stage and then deleting it; then add the following code to Frame 1.

TIP

First try this example without the two `addTreeNode()` statements at the end; then try the full example.

```

/**
 * Requires:
 *   - Tree component in library
 */

import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 10);
first_tr.setSize(200, 100);

this.createClassObject(Tree, "second_tr", 20);
second_tr.setSize(200, 100);
second_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
first_tr.dataProvider = trDP_xml;

var trDP2_xml:XML = new XML("<node label='2nd Local Folders'><node
    label='Outbox' data='0' /><node label='Outbox' data='1' /></node>");
second_tr.dataProvider = trDP2_xml;

// Add the node from second_tr to first_tr.
first_tr.addTreeNode(second_tr.getTreeNodeAt(0));

// Add node to first_tr.
first_tr.addTreeNode("Inbox", "data");

```

Tree.addTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
treeInstance.addTreeNodeAt(index, label [, data])
```

Usage 2:

```
treeInstance.addTreeNodeAt(index, child)
```

Parameters

index The zero-based index position (among the child nodes) at which the node should be added.

label A string that contains the name of the node to add.

data An object of any type that is associated with the node. This parameter is optional.

child Any XMLNode object.

Returns

The added XML node.

Description

Method; adds a node at the specified location in the tree. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt XMLNode object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the view.

Example

The following example creates two Tree components with a node for each one: 1st Local Folders and 2nd Local Folders, respectively. It uses the first usage of addTreeNodeAt() to add a new node, Inbox, to the first Tree. It then uses the second usage of addTreeNodeAt() to add the 1st node ((getTreeNodeAt(0)) from the second Tree to the first Tree.

You must first add the component to the document library by dragging a Tree component to the Stage and then deleting it; then add the following code to Frame 1.

TIP

First try this example without the two addTreeNodeAt() statements at the end; then try the full example.

```

/**
 * Requires:
 *   - Tree component in library
 */

import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 10);
first_tr.setSize(200, 100);

this.createClassObject(Tree, "second_tr", 20);
second_tr.setSize(200, 100);
second_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
first_tr.dataProvider = trDP_xml;

var trDP2_xml:XML = new XML("<node label='2nd Local Folders'><node
    label='Outbox' data='0' /><node label='Outbox' data='1' /></node>");
second_tr.dataProvider = trDP2_xml;

// Add node to first_tr.
first_tr.addTreeNodeAt(1, "Inbox", "data");
// Add the node from second_tr to first_tr.
first_tr.addTreeNodeAt(2, second_tr.getTreeNodeAt(0));

```

Tree.dataProvider

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.dataProvider
```

Description

Property; either XML or a string. If `dataProvider` is an XML object, it is added directly to the tree. If `dataProvider` is a string, it must contain valid XML that is read by the tree and converted to an XML object.

You can either load XML from an external source at runtime or create it in Flash while authoring. To create XML, you can use either the `TreeDataProvider` methods, or the built-in ActionScript XML class methods and properties. You can also create a string that contains XML.

XML objects that are on the same frame as a `Tree` component automatically contain the `TreeDataProvider` methods and properties. You can use the ActionScript XML or `XMLNode` object.

Example

The following example uses the `dataProvider` property to add the contents of an XML file to the `my_tr` instance of the `Tree` component:

You must first add an instance of the `Tree` component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function(success:Boolean){
    my_tr.dataProvider = trDP_xml.firstChild;
}
trDP_xml.load("http://www.flash-mx.com/mm/xml/tree.xml");
```

NOTE

Most XML files contain white space. To make Flash ignore white space, you must set the `XML.ignoreWhite` property to `true`.

Tree.firstVisibleNode

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.firstVisibleNode = someNode
```

Description

Property; indicates the first node that is visible at the top of the tree display. Use this property to scroll the tree display to a desired position. If the specified node *someNode* is within a node that hasn't been expanded, setting `firstVisibleNode` has no effect. The default value is the first visible node or undefined if there is no visible node. The value of this property is an XMLNode object.

This property is an analogue to the `List.vPosition` property.

Example

The following example populates a Tree component called `my_tr` with six nodes that it creates from a string of XML text. It makes the last node visible in the Tree by assigning the last node (relative position 5) to the `firstVisibleNode` property. You can make other nodes visible by changing 5 to other values from 0 to 4.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node><node label='3rd Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='4th Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node><node label='5th Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='6th Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node>");
my_tr.dataProvider = trDP_xml;

// Set visible node to last node.
my_tr.firstVisibleNode = my_tr.getTreeNodeAt(5);
```

Tree.getDisplayIndex()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.getDisplayIndex(node)
```

Parameters

node An XMLNode object.

Returns

The index of the specified node, or `undefined` if the node is not currently displayed.

Description

Method; returns the display index of the node specified in the *node* parameter.

The display index indicates the item's position in the list of items that are visible in the tree window. For example, any children of a closed node are not in the display index. The list of display indices starts with 0 and proceeds through the visible items regardless of parent. In other words, the index is the row number, starting with 0, of the displayed rows.

Example

The following example adds six nodes to a Tree and calls `getDisplayIndex()` to display the position of the node that the user selects.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 140);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node><node label='3rd Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='4th Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node><node label='5th Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='6th Local Folders'><node label='Inbox' data='0' /><node
  label='Outbox' data='1' /></node>");
my_tr.dataProvider = trDP_xml;
my_tr.firstChild = my_tr.getTreeNodeAt(0);

my_tr.addEventListener("change", listChanged);
function listChanged(evt_obj:Object) {
    trace(my_tr.getDisplayIndex(evt_obj.target.selectedNode));
}
```


Tree.getIsBranch()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.getIsBranch(node)
```

Parameters

node An XMLNode object.

Returns

A Boolean value that indicates whether the node is a branch (*true*) or not (*false*).

Description

Method; indicates whether the specified node has a folder icon and expander arrow (and is therefore a branch). This is set automatically when children are added to the node. You only need to call [Tree.setIsBranch\(\)](#) to create empty folders.

Example

The following example creates two nodes in a tree and calls `isBranch()` to determine whether the second one is a branch.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='2' /><node
  label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

var isBranch:Boolean = my_tr.getIsBranch(my_tr.getTreeNodeAt(1));
trace("2nd node is a branch: " + isBranch);
```

See also

[Tree.setIsBranch\(\)](#)

Tree.getIsOpen()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.getIsOpen(node)
```

Parameters

node An XMLNode object.

Returns

A Boolean value that indicates whether the tree is open (`true`) or closed (`false`).

Description

Method; indicates whether the specified node is open or closed.

NOTE

Display indices change every time nodes open and close.

Example

The following example adds two nodes to a Tree called `my_tr`, opens the second node, and then calls `getIsOpen()` to display the state of the second node.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);
```

```
var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='2' /><node
  label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;
my_tr.setIsOpen(my_tr.getTreeNodeAt(1), true);
var isOpen:Boolean = my_tr.getIsOpen(my_tr.getTreeNodeAt(1));
trace("2nd node is a open: " + isOpen);
```

Tree.getNodeDisplayedAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.getNodeDisplayedAt(index)
```

Parameters

index An integer representing the display position in the viewable area of the tree. This number is zero-based; the node at the first position is 0, second position is 1, and so on.

Returns

The specified XMLNode object.

Description

Method; maps a display index of the tree onto the node that is displayed there. For example, if the fifth row of the tree showed a node that is eight levels deep into the hierarchy, that node would be returned by a call to `getNodeDisplayedAt(4)`.

The display index is an array of items that can be viewed in the tree window. For example, any children of a closed node are not in the display index. The display index starts with 0 and proceeds through the visible items regardless of parent. In other words, the display index is the row number, starting with 0, of the displayed rows.

NOTE

Display indices change every time nodes open and close.

Example

The following example adds a node to a `Tree` instance called `my_tr` and then calls the `getNodeDisplayedAt()` method to retrieve the node at display position 0 (zero). The example calls the `trace()` function to display the node.

You must first add an instance of the `Tree` component to the `Stage` and name it `my_tr`; then add the following code to `Frame 1`.

```
/**
 * Requires:
 *   - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label=\"1st Local Folders\"><node
  label=\"Inbox\" data=\"0\" />");
my_tr.dataProvider = trDP_xml;

trace(my_tr.getNodeDisplayedAt(0));
```

Tree.getTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.getTreeNodeAt(index)
```

Parameters

index The index number of a node.

Returns

An `XMLNode` object.

Description

Method; returns the specified node on the root of `myTree`.

Example

The following example adds a node to the Tree instance `my_tr`, and then calls the `getNodeAt()` method to return the first node in the tree.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 *   - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label=\"1st Local Folders\"><node
  label=\"Inbox\" data=\"0\" /><node label=\"Outbox\" data=\"1\" /></
  node>");
my_tr.dataProvider = trDP_xml;

trace(my_tr.getNodeAt(0));
```

Tree.nodeClose

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.nodeClose = function(eventObject:Object) {
    // Insert your code here.
};
treeInstance.addEventListener("nodeClose", listenerObject);
```

Description

Event; broadcast to all registered listeners when the nodes of a Tree component are closed by a user.

Version 2 components use a dispatcher/listener event model. The Tree component broadcasts a `nodeClose` event when one of its nodes is clicked closed; the event is handled by a function, also called a *handler*, that is attached to a listener object (*listenerObject*) that you create.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Tree.nodeClose` event's event object has one additional property: `node` (the XML node that closed).

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example adds two nodes to the `Tree` instance `my_tr`, and then creates two listener objects, one for `nodeOpen` events and one for `nodeClose` events. When these events occur, the listener function uses a trace statement to display the event and the affected node in the Output panel.

You must first add an instance of the `Tree` component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
    label='2nd Local Folders'><node label='Inbox' data='2' /><node
    label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Create listener object.
var trListener:Object = new Object();
trListener.nodeOpen = function(evt_obj:Object){
    trace("Node opened\n" + evt_obj.node);
    trace("\n");
}
trListener.nodeClose = function(evt_obj:Object){
    trace("Node closed\n" + evt_obj.node);
    trace("\n");
}
// Add listeners.
my_tr.addEventListener("nodeOpen", trListener);
my_tr.addEventListener("nodeClose", trListener);
```

Tree.nodeOpen

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
var listenerObject:Object = new Object();
listenerObject.nodeOpen = function(eventObject:Object) {
    // Insert your code here.
};
treeInstance.addEventListener("nodeOpen", listenerObject);
```

Description

Event; broadcast to all registered listeners when a user opens a node on a Tree component.

Version 2 components use a dispatcher/listener event model. The Tree component dispatches a `nodeOpen` event when a node is clicked open by a user; the event is handled by a function, also called a *handler*, that is attached to a listener object (*listenerObject*) that you create.

You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Tree.nodeOpen` event's event object has one additional property: `node` (the XML node that was opened).

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example adds two nodes to the Tree instance `my_tr`, and then creates two listener objects, one for `nodeOpen` events and one for `nodeClose` events. When these events occur, the listener functions call trace statements to display the event and the affected node in the Output panel.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
    label='2nd Local Folders'><node label='Inbox' data='2' /><node
    label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Create listener object.
var trListener:Object = new Object();
trListener.nodeOpen = function(evt_obj:Object){
    trace("Node opened\n" + evt_obj.node);
    trace("\n");
}
trListener.nodeClose = function(evt_obj:Object){
    trace("Node closed\n" + evt_obj.node);
    trace("\n");
}
// Add listeners.
my_tr.addEventListener("nodeOpen", trListener);
my_tr.addEventListener("nodeClose", trListener);
```

Tree.refresh()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.refresh()
```

Parameters

None.

Returns

Nothing.

Description

Method; updates the tree.

Example

The following example adds a node to a Tree instance called `my_tr` and creates listeners for a Refresh button and a Remove All button. Assuming the XML source for the data provider has changed, the user can click the Refresh button, and the code calls the `refresh()` method to update the tree contents.

You must first add an instance of the Tree component to the Stage and name it `my_tr`. Then add a button called `refresh_button`. Then add the following code to Frame 1 in the main timeline:

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 * - Button component on Stage (instance name: refresh_button)
 */

var my_tr:mx.controls.Tree;
var refresh_button:mx.controls.Button;
var removeAll_button:mx.controls.Button;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function() {
    my_tr.dataProvider = this.firstChild;
};
trDP_xml.load("http://yourXMLsourcehere");

function refreshListener(evt_obj:Object):Void {
    my_tr.refresh();
}
refresh_button.addEventListener("click", refreshListener);
```

Tree.removeAll()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all nodes and refreshes the tree.

Example

The following example adds a node to a Tree instance called `my_tr` and creates a listener for a Remove All button. When the user clicks the Remove All button, the code calls the `removeAll()` method to remove all nodes from the tree.

You first add an instance of the Tree component to the Stage and name it `my_tr` and add a button called `removeAll_button`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 * - Button component on Stage (instance name: refresh_button)
 * - Button component on Stage (instance name: removeAll_button)
 */

var my_tr:mx.controls.Tree;
var removeAll_button:mx.controls.Button;

my_tr.setSize(200, 100);
```

```
var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function() {
    my_tr.dataProvider = this.firstChild;
};
trDP_xml.load("http://www.flash-mx.com/mm/xml/tree.xml");

function removeAllListener(evt_obj:Object):Void {
    my_tr.removeAll();
}
removeAll_button.addEventListener("click", removeAllListener);
```

Tree.removeTreeNodeAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.removeTreeNodeAt(index)
```

Parameters

index The index number of a tree child. Each child of a tree is assigned a zero-based index in the order in which it was created.

Returns

An XMLNode object, or undefined if an error occurs.

Description

Method; removes a node (specified by its index position) on the root of the tree and refreshes the tree.

Example

The following example adds two nodes to a Tree instance and creates a listener for a change event on the tree. When a change event occurs, the listener functions call the `removeTreeNodeAt()` method to delete the selected node from the tree.

You first add an instance of the Tree component to the Stage and name it `my_tr` and then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='2' /><node
  label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

var treeListener:Object = new Object();
treeListener.change = function (evt_obj:Object) {
    my_tr.removeTreeNodeAt(my_tr.selectedIndex);
}
my_tr.addEventListener("change", treeListener);
```

Tree.selectedNode

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.selectedNode
```

Description

Property; specifies the selected node in a tree instance.

Example

The following example adds two nodes to a Tree instance and sets the second node to the selected state.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Tree component on Stage (instance name: my_tr)
 */
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='2' /><node
  label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Select the second node.
my_tr.selectedNode = my_tr.getTreeNodeAt(1);
```

See also

[Tree.selectedNodes](#)

Tree.selectedNodes

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.selectedNodes
```

Description

Property; specifies the selected nodes in a tree instance.

Example

The following example adds three nodes to a Tree instance and sets the first two to the selected state.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1:

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
  label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
  label='2nd Local Folders'><node label='Inbox' data='2' /><node
  label='Outbox' data='3' /></node><node label='3rd Local Folders'><node
  label='Inbox' data='2' /><node label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Allow multiple selections.
my_tr.multipleSelection = true;

// Select first and second node.
my_tr.selectedNodes = [my_tr.getTreeNodeAt(0), my_tr.getTreeNodeAt(1)];
```

See also

[Tree.selectedNode](#)

Tree.setIcon()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.setIcon(node, linkID [, linkID2])
```

Parameters

node An XML node.

linkID The linkage identifier of a symbol to be used as an icon beside the node. This parameter is used for leaf nodes and for the closed state of branch nodes.

linkID2 For a branch node, the linkage identifier of a symbol to be used as an icon that represents the open state of the node. This parameter is optional.

Returns

Nothing.

Description

Method; specifies an icon for the specified node. This method takes one ID parameter (*linkID*) for leaf nodes and two ID parameters (*linkID* and *linkID2*) for branch nodes (the closed and open icons). For leaf nodes, the second parameter is ignored. For branch nodes, if you omit *linkID2*, the icon is used for both the closed and open states.

Example

The following example adds two nodes to a Tree instance called `my_tr` and calls the `setIcon()` function to specify an icon in the library with a Linkage ID of `imageIcon` for the second node.

You must first add an instance of the Tree component to the Stage and name it `my_tr` and add an icon to the library with a Linkage ID of `imageIcon`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 * - Library item with Linkage ID of imageIcon
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
    label='2nd Local Folders'><node label='Inbox' data='2' /><node
    label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Set movieclip as icon for 2nd node.
my_tr.setIcon(my_tr.getTreeNodeAt(1), "imageIcon");
```

Tree.setIsBranch()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.setIsBranch(node, isBranch)
```

Parameters

node An XML node.

isBranch A Boolean value indicating whether the node is (`true`) or is not (`false`) a branch.

Returns

Nothing.

Description

Method; specifies whether the node has a folder icon and expander arrow and either has children or can have children. A node is automatically set as a branch when it has children; you only need to call `setIsBranch()` when you want to create an empty folder. You may want to create branches that don't yet have children if, for example, you only want child nodes to load when a user opens a folder.

Calling `setIsBranch()` refreshes any views.

Example

The following example adds a single node to a `Tree` instance called `my_tr` and calls `setIsBranch()` to make it a branch without children.

You must first add an instance of the Tree component to the Stage and name it `my_tr`; then add the following code to Frame 1.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='Inbox' data='0'/>");
my_tr.dataProvider = trDP_xml;

// Set 1st node to be branch.
my_tr.setIsBranch(my_tr.getTreeNodeAt(0), true);
```

Tree.setIsOpen()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
treeInstance.setIsOpen(node, open [, animate [, fireEvent]])
```

Parameters

node An XML node.

open A Boolean value that opens a node (`true`) or closes it (`false`).

animate A Boolean value that determines whether the opening transition is animated (`true`) or not (`false`). This parameter is optional.

fireEvent A Boolean value that determines whether the `nodeOpen` and `nodeClose` events are dispatched (`true`) or not (`false`) when the tree node is opened or closed. This parameter is optional. The default value is `false`.

Returns

Nothing.

Description

Method; opens or closes a node.

Example

The following example creates two nodes in a `Tree` instance called `my_tr` and calls the `setIsOpen()` method to open the second node.

```
/**
 * Requires:
 * - Tree component on Stage (instance name: my_tr)
 */
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
    label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
    label='2nd Local Folders'><node label='Inbox' data='2' /><node
    label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// Set 2nd node open.
my_tr.setIsOpen(my_tr.getTreeNodeAt(1), true);
```

Inheritance (Root class)

ActionScript Class Name mx.transitions.Tween

The Tween class lets you use ActionScript to move, resize, and fade movie clips easily on the Stage by specifying a property of the target movie clip to be tween animated over a number of frames or seconds. The Tween class also lets you specify a variety of easing methods. *Easing* refers to gradual acceleration or deceleration during an animation, which helps your animations appear more realistic. For example, the options on a drop-down list component you create might gradually increase their speed near the beginning of an animation as the options appear, but slow down before the options come to a full stop at the end of the animation as the list is extended. Flash provides many easing methods that contain equations for this acceleration and deceleration, which change the easing animation accordingly.

The Tween class also invokes event handlers so your code may respond when an animation starts, stops, or resumes or increments its tweened property value. For example, you can start a second tweened animation when the first tween invokes its `Tween.onMotionStopped` event handler, indicating that the first tween has stopped.

Method summary for the Tween class

The following table lists methods of the Tween class:

Method	Description
<code>Tween.yoyo()</code>	Instructs the tweened animation to continue from its current value to a new value.
<code>Tween.ffforward()</code>	Forwards the tweened animation directly to the end of the animation.
<code>Tween.nextFrame()</code>	Forwards the tweened animation to the next frame.
<code>Tween.prevFrame()</code>	Directs the tweened animation to the frame previous to the current frame.

Method	Description
<code>Tween.resume()</code>	Resumes a tweened animation from its stopped point in the animation.
<code>Tween.rewind()</code>	Rewinds a tweened animation to the beginning of the tweened animation.
<code>Tween.start()</code>	Starts the tweened animation from the beginning.
<code>Tween.stop()</code>	Stops the tweened animation at its current position.
<code>Tween.toString()</code>	Returns the class name, "[Tween]".
<code>Tween.yoyo()</code>	Instructs the tweened animation to play in reverse from its last direction of tweened property increments.

Property summary for the Tween class

The following table lists properties of the Tween class.

Property	Description
<code>Tween.duration</code>	The duration of the tweened animation in frames or seconds. Read-only.
<code>Tween.finish</code>	The last tweened value for the end of the tweened animation. Read-only.
<code>Tween.FPS</code>	The number of frames per second of the tweened animation. Read-only.
<code>Tween.position</code>	The current value of the target movie clip's property being tweened. Read-only.
<code>Tween.time</code>	The current time within the duration of the animation. Read-only.

Event handler summary for the Tween class

The following table lists event handlers of the Tween class.

Event	Description
<code>Tween.onMotionChanged</code>	Event handler; invoked with each change in the tweened object's property that is being animated.
<code>Tween.onMotionFinished</code>	Event handler; invoked when the Tween object finishes its animation.

Event	Description
<code>Tween.onMotionResumed</code>	Event handler; invoked when the <code>Tween.resume()</code> method is called, causing the tweened animation to resume.
<code>Tween.onMotionStarted</code>	Event handler; invoked when the <code>Tween.start()</code> method is called, causing the tweened animation to start.
<code>Tween.onMotionStopped</code>	Event handler; invoked when the <code>Tween.stop()</code> method is called, causing the tweened animation to stop.

Using the Tween class

To use the methods and properties of the Tween class, you use the `new` operator to create a new instance of the class. For example, to apply an instance of a tween to a movie clip object on the Stage called `myMovieClip_mc`, you use the following code to create a new instance of `mx.transitions.Tween`:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(myMovieClip_mc, "_x",
    mx.transitions.easing.Elastic.easeOut, 0, 300, 3, true);
```

Tween class parameters

When you create a new instance of a Tween class, you pass several parameters. You must indicate the target movie clip object, what property of the movie clip the tween is to affect, the range over which the object is to be tweened, and an easing method to use to calculate the tweened property.

The constructor for the `mx.transitions.Tween` class has the following parameter names and types:

```
Tween( obj:Object, prop:String, func:Function, begin:Number, finish:Number,
    duration:Number, useSeconds:Boolean )
```

obj The movie clip object that the Tween instance targets.

prop A string name of a property in `obj` to which the values are to be tweened.

func The easing method that calculates an easing effect for the tweened object's property values. See [“About easing classes and methods” on page 1314](#)

begin A number indicating the starting value of `prop` (the target object property to be tweened).

finish A number indicating the ending value of `prop` (the target object property to be tweened).

duration A number indicating the length of time of the tween motion. If omitted, the duration is set to infinity by default.

useSeconds A Boolean value indicating to use seconds if `true` or frames if `false` in relation to the value specified in the `duration` parameter.

About easing classes and methods

When you create an instance of the Tween class, you use the `func` parameter to specify a function or method that provides an easing calculation. Flash provides five easing classes, each with three methods that indicate which part of the transitional motion to apply the easing effect to: at the beginning of the animation, the end, or both. In addition, a `None` easing class with an `easeNone` method is available for designating that no easing be used.

The following classes and components use the easing classes and methods:

- The `mx.transitions.Tween` class for easing effects on a tweened animation
- The `mx.transitions.TransitionManager` class for easing effects on transitions. See [Chapter 48, “TransitionManager class,” on page 1237](#).
- Some components in version 2 of the Macromedia Component Architecture. See [“Applying easing methods to components” on page 1315](#)

The six easing calculation classes are described in the following table:

Easing Class	Description
Back	Extends the animation once beyond the transition range at one or both ends to give the effect of being pulled back from beyond its range.
Bounce	Adds a bouncing effect within the transition range at one or both ends. The number of bounces relates to the duration—longer durations produce more bounces.
Elastic	Adds an elastic effect that falls outside the transition range at one or both ends. The amount of elasticity is unaffected by the duration.
Regular	Adds slower movement at one or both ends. This feature lets you add a speeding up effect, a slowing down effect, or both.
Strong	Adds slower movement at one or both ends. This effect is similar to the Regular easing class, but it’s more pronounced.
None	Adds an equal movement from start to end without effects, slowing, or speeding up. This transition is also called a linear transition.

These six easing calculation classes each have three easing methods, which indicate at what part of the animation to apply the easing effect. In addition, the None easing class has a fourth easing method: `easeNone`. The easing methods are described in the following table:

Method	Description
<code>easeIn</code>	Provides the easing effect at the beginning of the transition.
<code>easeOut</code>	Provides the easing effect at the end of the transition.
<code>easeInOut</code>	Provides the easing effect at the beginning and end of the transition.
<code>easeNone</code>	Indicates no easing calculation is to be used. Provided only in the None easing class.

Applying easing methods to components

Another use for the various easing methods is to apply them on version 2 of the Macromedia Component Architecture. You can apply the easing methods only to the following version 2 components: `Accordion`, `ComboBox`, `DataGrid`, `List`, `Menu`, and `Tree`. Each component uses the easing methods to allow different customizations. For example, the `Accordion`, `ComboBox`, and `Tree` components let you select an easing class to use for their respective open and close animations. In contrast, the `Menu` component lets you define only the number of milliseconds that the animation lasts.

Applying easing methods to an Accordion component

This section describes how to add an `Accordion` component to a Flash document, add a few child slides, and change the default easing method and duration. If you decide to use this code in a project, reduce the value of the `openDuration` property to avoid annoying users with animations that are too slow when they open and close the `Accordion` component's child panes.

To apply a different easing method to the Accordion component:

1. Create a new Flash document and save it as `accordion fla`.
2. Drag a copy of the `Accordion` component onto the Stage.
3. Open the Property inspector, and type `my_acc` into the Instance Name text box.
4. Insert a new layer above Layer 1, and name it *actions*.

5. Add the following ActionScript to Frame 1 of the actions layer:

```
import mx.core.View;
import mx.transitions.easing.*;
my_acc.createChild(View, "studio_view", {label:"Studio"});
my_acc.createChild(View, "dreamweaver_view", {label:"Dreamweaver"});
my_acc.createChild(View, "flash_view", {label:"Flash"});
my_acc.createChild(View, "coldfusion_view", {label:"ColdFusion"});
my_acc.createChild(View, "contribute_view", {label:"Contribute"});
my_acc.setStyle("openEasing", Bounce.easeOut);
my_acc.setStyle("openDuration", 3500);
```

This code imports the easing classes, so you can type `Bounce.easeOut` instead of referring to each of the classes with fully qualified names such as

`mx.transitions.easing.Bounce.easeOut`. Next, the code adds five new child panes to the Accordion component (Studio, Dreamweaver, Flash, ColdFusion, and Contribute).

The final two lines of code set the easing style from the default easing method to `Bounce.easeOut` and set the length of the animation to 3500 milliseconds (3.5 seconds).

6. Save the document, and then select **Control > Test Movie** to preview the file in the test environment.

Click each of the different header (title) bars to view the modified animations and switch between each pane.

If you want to increase the animation speed, decrease `openDuration` from 3500 milliseconds to a smaller number. The default duration for the animation is 250 milliseconds (one fourth of a second).

Applying easing methods to the ComboBox component

The process to change the default easing method on a ComboBox component is similar to the example in [“Applying easing methods to an Accordion component” on page 1315](#) where you modify the Accordion component’s animation. In the following example, you use ActionScript to dynamically add the component to the Stage at runtime.

To apply easing methods to a ComboBox component:

1. Create a new Flash document and save it as `combobox fla`.
2. Drag a copy of the ComboBox component from the Components panel to the current document’s library.

NOTE

The component appears in the library (not on the Stage) and is available to the SWF file at runtime.

3. Insert a new layer and rename it actions.

Make sure the actions layer is above Layer 1.

4. Add the following ActionScript to Frame 1 of the actions layer:

```
import mx.transitions.easing.*;
this.createClassObject(mx.controls.ComboBox, "my_cb", 20);
var product_array:Array = new Array("Studio", "Dreamweaver", "Flash",
    "ColdFusion", "Contribute", "Breeze", "Director", "Flex");
my_cb.dataProvider = product_array;
my_cb.move(10, 10);
my_cb.setSize(140, 22);
my_cb.setStyle("openDuration", 2000);
my_cb.setStyle("openEasing", Elastic.easeOut);
```

After you import each of the easing methods, which occurs in the first line of code, the `createClassObject()` method creates an instance of the `ComboBox` component. The keyword `this` in the second line of code refers to the main of the SWF file. This line of code puts the component on the Stage at runtime and gives it the instance name `my_cb`.

Next, you create an array named `product_array` that contains a list of Macromedia software. You use this array in the following line of code to set the `dataProvider` property to the array of product names. Then you use the `setSize()` method to resize the component instance, set `openDuration` to 2000 milliseconds (2 seconds), and change the easing method to `Elastic.easeOut`.

NOTE

As with earlier examples, you import the easing classes, which let you use the shortened version of the class name instead of using the fully qualified class name of `mx.transitions.easing.Elastic.easeOut`.

5. Save the current document, and select Control > Test Movie to view the document in the test environment.
6. Click the `ComboBox` component on the Stage to use the specified easing class to animate your drop-down list of product names.

NOTE

Use an easing method such as `Elastic` or `Bounce` for your `ComboBox` or `Accordion` components with care. Some users might find it distracting if your options take a long time to stop moving before they can read and select from the menu. Test your individual applications and settings, and decide whether the easing methods enhance or detract from your Flash document.

Animating the DataGrid component

Flash 8 also lets you tweak the animations you use when you select items in a component (such as the `DataGrid`, `Tree`, `ComboBox`, or `List` components). Although the animations are subtle, in some cases you want to control small details or increase the speed of the animation.

To add easing to the DataGrid component:

1. Create a new Flash document and save it as `datagrid fla`.
2. Drag an instance of the DataGrid component onto the Stage, and give it the name `my_dg`.
3. Insert a new layer and rename it `actions`.

Make sure you place the actions layer above Layer 1.

4. Add the following ActionScript to the actions layer:

```
import mx.transitions.easing.*;
my_dg.setSize(320, 240);
my_dg.addColumn("product");
my_dg.getColumnAt(0).width = 304;
my_dg.rowHeight = 60;
my_dg.addItem({product:"Studio"});
my_dg.addItem({product:"Dreamweaver"});
my_dg.addItem({product:"Flash"});
my_dg.setStyle("selectionEasing", Elastic.easeInOut);
my_dg.setStyle("selectionDuration", 1000);
```

This ActionScript imports the easing classes and resizes the component instance on the Stage to 320 pixels (width) by 240 pixels (height). Next, you create a new column named *product* and resize the column to 304 pixels (width). The data grid itself is 320 pixels wide, although the scroll bar is 16 pixels wide, which leaves a difference of 304 pixels. Then you set the row height to 60 pixels, which makes the easing animations easier to see.

The next three lines of ActionScript add items to the data grid instance so you can click and see the animations. Finally, the `selectionEasing` and `selectionDuration` properties are set using the `setStyle()` method. The easing method is set to `Elastic.easeInOut` and the duration is set to 1000 milliseconds (one second, which is five times longer than the default value of 200 milliseconds).

5. Save the document and select `Control > Test Movie` to view the result in the test environment.

When you click an item in the DataGrid instance, you see the selection ease in and out using the elastic effect. The animation should be easy to see because the duration is significantly increased.

NOTE

You can also use the same properties (`selectionEasing` and `selectionDuration`) with the `ComboBox`, `List`, and `Tree` components.

Tween.continueTo()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.continueTo(finish, duration)
```

Parameters

finish A number indicating the ending value of the target object property that is to be tweened.

duration A number indicating the length of time or number of frames for the tween motion; duration is measured in length of time if the Tween.start() useSeconds parameter is set to true, or measured in frames if it is set to false. For more information on the useSeconds parameter, see [Tween.start\(\) on page 1334](#).

Returns

Nothing.

Description

Method; instructs the tweened animation to continue tweening from its current animation point to a new finish and duration point.

Example

In this example a handler is triggered by the onMotionFinished event and tells a Tween instance to continue its animation with new finish and duration values by calling the Tween.continueTo() method. A movie clip instance named img1_mc is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
    mx.transitions.easing.Elastic.easeOut,0, 200, 3, true);
myTween.onMotionFinished = function() {
    var myFinish:Number = 100;
    var myDuration:Number = 5;
    myTween.continueTo( myFinish, myDuration );
};
```

Tween.duration

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.duration

Description

Property (read-only); a number indicating the duration of the tweened animation in frames or seconds. This property is set as a parameter when creating a new Tween instance or when calling the [Tween.yoyo\(\)](#) method.

Example

The following example traces the current `duration` setting of a Tween object by getting the `Tween.duration` property. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
    mx.transitions.easing.Strong.easeOut,0, Stage.height, 50, false);
var theDuration:Number = myTween.duration;
trace(theDuration);
```

Tween.fforward()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.fforward()

Returns

Nothing.

Description

Method; forwards the tweened animation directly to the final value of the tweened animation.

Example

In this example, `Tween.fforward()` is called to forward a tweened animation directly to its final value, immediately triggering the `onMotionFinished` event. A handler for the `Tween.onMotionFinished` event calls the `Tween.yoyo()` method. The tweened animation therefore visibly starts with the reversing effect of the `Tween.yoyo()` method, since the initial animation was skipped to its end. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.Elastic.easeOut,0, Stage.width - img1_mc._width, 8,
    true);

myTween.fforward();

myTween.onMotionFinished = function() {
    myTween.yoyo();
};
```

Tween.finish

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.finish

Description

Property (read-only); a number indicating the ending value of the target object property that is to be tweened. This property is set as a parameter when creating a new Tween instance or when calling the `Tween.yoyo()` method.

Example

The following example returns the current `finish` setting of a Tween instance. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
    mx.transitions.easing.Strong.easeOut,0, Stage.height - img1_mc._height,
    50, false);
var myFinish:Number = myTween.finish;
trace(myFinish);
```

Tween.FPS

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.FPS

Description

Property; the number of frames per second calculated into the tweened animation. By default the current Stage frame rate is used to calculate the tweened animation. Setting this property recalculates the number of increments in the animated property that is displayed each second to the `Tween.FPS` property rather than the current Stage frame rate. Setting the `Tween.FPS` property does not change the actual frame rate of the Stage.

NOTE

The `Tween.FPS` property returns undefined unless it is first set explicitly.

Example

The following example creates two tweened animations set at two different FPS settings. The current FPS settings of both Tween instances are displayed. A movie clip instance named `img1_mc`, and a movie clip instance named `img2_mc` are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween1:Tween = new Tween(img1_mc, "_y",
    mx.transitions.easing.Strong.easeOut,0, Stage.height - img1_mc._height,
    400, false);
myTween1.FPS = 1;
var myFPS1:Number = myTween1.FPS;
trace("myTween1.FPS:" + myFPS1);

var myTween2:Tween = new Tween(img2_mc, "_y",
    mx.transitions.easing.Strong.easeOut,0, Stage.height - img2_mc._height,
    400, false);
myTween2.FPS = 12;
var myFPS2:Number = myTween2.FPS;
trace("myTween2.FPS:" + myFPS2);
```

Tween.nextFrame()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.nextFrame()

Returns

Nothing.

Description

Method; forwards the tweened animation to the next frame of an animation that was stopped. Use this method to forward a frame at a time of a tweened animation after you use the [Tween.stop\(\)](#) method to stop it.

NOTE

This method may be used only on frame-based tweens. A tween is set to frame based at its creation by setting the `useSeconds` parameter to `false`.

Example

This example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within a handler triggered by the `Tween.onMotionFinished` event. Clicking a button called `forwardByFrame_btn` calls the `Tween.stop()` method to stop the animation, followed by calling the `Tween.nextFrame()` method. Clicking the button during the tweened animation has the effect of stopping the animation and then moving forward by only a single frame. When you create the `Tween` instance, the `useSeconds` parameter is declared `false` to make the tween frame based. This process is required to use the `Tween.nextFrame()` method. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width, 60, false);

myTween.onMotionFinished = function() {
    myTween.start();
};

forwardByFrame_btn.onRelease = function() {
    myTween.stop();
    myTween.nextFrame();
};
```

Tween.onMotionChanged

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.onMotionChanged = function() {
    // ...
};
```


Description

Event handler; invoked with each change in the tweened object property that is being animated. Handling this event allows your code to react as the target movie clip's property that is being tweened increments to the next value.

Example

In this example, a tween is applied to the `img1_mc` movie clip. With each increment of the tween to the `_x` property of the movie clip, the `onMotionChanged` event handler is invoked and displays a trace message indicating the tweened movie clip's new position. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.Elastic.easeOut,0, Stage.width-img1_mc._width, 3,
    true);

myTween.onMotionChanged = function() {
    trace( this.position );
};
```

Tween.onMotionFinished

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.onMotionFinished = function() {
    // ...
};
```

Description

Event handler; invoked when the animation reaches the end of its duration. Handling this event allows your code to react at the point at which the tweened animation is finished.

Example

In the following example, a tween is applied to the `img1_mc` movie clip. When the tween reaches the end of its animation, it invokes the `onMotionFinished` event handler which calls the `Tween.yoyo()` method. The tween is therefore able to complete its animation before the `Tween.yoyo()` method is called to reverse the animation. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.Elastic.easeOut,0, Stage.width-img1_mc._width, 3,
    true);
myTween.FPS = 30;
myTween.onMotionFinished = function() {
    myTween.yoyo();
};
```

Tween.onMotionResumed

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.onMotionResumed = function() {
    // ...
};
```

Description

Event handler; invoked when the `Tween.resume()` method is called. Handling this event allows your code to react at the point at which the tweened animation was resumed.

Example

The following example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within an `onMotionFinished` event handler. Clicking a button called `stopTween_btn` calls the `Tween.stop()` method to stop the tweened animation at its current value. Clicking a button called `resumeTween_btn` calls the `Tween.resume()` method to resume the tweened animation from its stopping point. When the `Tween.resume()` method is called, the Tween instance invokes the `onMotionResumed` handler. A movie clip instance named `img1_mc`, a movie clip instance named `stopTween_btn`, and a movie clip instance named `resumeTween_btn`, are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width, 3, true);

myTween.onMotionFinished = function() {
    myTween.start();
};

myTween.onMotionResumed = function() {
    trace("onMotionResumed");
};

stopTween_btn.onRelease = function() {
    myTween.stop();
};

resumeTween_btn.onRelease = function() {
    myTween.resume();
};
```

Tween.onMotionStarted

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.onMotionStarted = function() {
    // ...
};
```

Description

Event handler; invoked when the animation starts again during or after completing its animation. This event handler is not invoked at the initial start of a tweened animation. Calling the `Tween.start()`, `Tween.yoyo()` or `Tween.yoyo()` method to restart a finished animation or restart during an animation invokes the `onMotionStarted` event handler. Handling this event allows your code to react at the point at which the tweened animation was started again sometime after its initial start.

Example

The following example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within the `Tween.onMotionFinished` event handler. When the `Tween.start()` method is called, the Tween instance invokes the `Tween.onMotionStarted` event handler. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width, 4, true);

myTween.onMotionFinished = function() {
    myTween.start();
};

myTween.onMotionStarted = function() {
    trace("onMotionStarted");
};
```

Tween.onMotionStopped

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.onMotionStopped = function() {
    // ...
};
```

Description

Event handler; invoked when the tweened animation completes to the end of its animation or when the `Tween.stop()` method is called. Handling this event allows your code to react at the point at which the tweened animation was stopped.

Example

The following example applies a tweened animation to the `img1_mc` movie clip. When the Tween instance finishes its animation, the Tween instance invokes the `Tween.onMotionStopped` event handler. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width-img1_mc._width, 3,
    true);

myTween.onMotionStopped = function() {
    trace("onMotionStopped");
};
```

Tween.position

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.position

Description

Property (read-only); the current value of the target object property being tweened. This value updates with each drawn frame of the tweened animation.

Example

The following example traces a Tween object's current `Tween.position` and the `Tween.position` value that the tween position ends with at the last frame of the tweened animation. A movie clip instance named `img1_mc` is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new mx.transitions.Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, 0, Stage.width-img1_mc._width, 3,
    true);
myTween.onMotionChanged = function() {
    var myPosition:Number = myTween.position;
    var myFinish:Number = myTween.finish;
    trace(myPosition + " : " + myFinish);
};
```

Tween.prevFrame()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.prevFrame()

Returns

Nothing.

Description

Method; plays the previous frame of the tweened animation from the current stopping point of an animation that was stopped. Use this method to play a tweened animation backwards one frame at a time after you use the `Tween.stop()` method to stop it.

NOTE

This method may be used only on frame-based tweens. A tween is set to frame based at its creation by setting the `Tween.start()` `useSeconds` parameter to `false`. For more information on the `useSeconds` parameter, see [Tween.start\(\) on page 1334](#).

Example

This example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within a handler triggered by the `onMotionFinished` event. Clicking a button called `forwardByFrame_btn` calls the `Tween.stop()` method to stop the animation, followed by calling the `Tween.prevFrame()` method. Clicking the button during the tweened animation stops the animation and then reverses it by only a single frame. Clicking the `resumeTween_btn` button calls the `Tween.resume()` method, and the tweened animation resumes. When you create the `Tween` instance, the `useSeconds` parameter is declared `false` to make the tween frame-based. This process is required to use the `Tween.nextFrame()` method. A movie clip instance named `img1_mc`, a movie clip instance named `resumeTween_btn` and a movie clip instance named `reverseByFrame_btn` are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, -img1_mc._width, Stage.width, 50,
    false);

myTween.onMotionFinished = function() {
    myTween.start();
};

reverseByFrame_btn.onRelease = function() {
    myTween.stop();
    myTween.prevFrame();
};

resumeTween_btn.onRelease = function() {
    myTween.resume();
};
```

Tween.resume()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.resume()
```

Returns

Nothing.

Description

Method; resumes the play of a tweened animation that has been stopped. Use this method to continue a tweened animation after you have stopped it by using the `Tween.stop()` method.

NOTE

This method may be used only on frame-based tweens. A tween is set to be frame based at its creation by setting the `useSeconds` parameter to `false`.

Example

This example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within a handler triggered by the `onMotionFinished` event. Clicking the `stopTween_btn` button calls the `Tween.stop()` method to stop the tweened animation at its current value. Clicking the `resumeTween_btn` button calls the `Tween.resume()` method to resume the tweened animation from its stopping point. A movie clip instance named `img1_mc`, a movie clip instance named `stopTween_btn`, and a movie clip instance named `resumeTween_btn`, are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, -img1_mc._width, Stage.width, 3,
    true);

myTween.onMotionFinished = function() {
    myTween.start();
};

stopTween_btn.onRelease = function() {
    myTween.stop();
};

resumeTween_btn.onRelease = function() {
    myTween.resume();
};
```


Tween.rewind()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.rewind()
```

Parameters

None.

Returns

Nothing.

Description

Method; moves the play of a tweened animation back to its starting value. If `Tween.rewind()` is called while the tweened animation is still playing, the animation rewinds to its starting value and continues playing. If `Tween.rewind()` is called while the tweened animation has been stopped or has finished its animation, the tweened animation rewinds to its starting value and remains stopped. Use this method to rewind a tweened animation to its starting point after you have stopped it by using the `Tween.stop()` method or to rewind a tweened animation during its play.

Example

The following example applies a tweened animation to the `img1_mc` movie clip. The animation is looped to play repeatedly from its starting point by calling the `Tween.start()` method from within a handler triggered by the `Tween.onMotionFinished` event. Clicking the `rewindTween_btn` button calls the `Tween.rewind()` method to rewind the tweened animation to its starting point. A movie clip instance named `img1_mc`, a movie clip instance named `stopTween_btn`, a movie clip instance named `rewindTween_btn` and a movie clip instance named `resumeTween_btn`, are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, img1_mc._width, Stage.width, 8,
    true);

myTween.onMotionFinished = function() {
    myTween.start();
};

stopTween_btn.onRelease = function() {
    myTween.stop();
};

rewindTween_btn.onRelease = function() {
    myTween.rewind();
};

resumeTween_btn.onRelease = function() {
    myTween.resume();
};
```

Tween.start()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.start()
```

Returns

Nothing.

Description

Method; starts the play of a tweened animation from its starting point. This method is used for re-starting a Tween from the beginning of its animation after it stops or has completed its animation.

Example

This example creates a new Tween object that animates the `_x` property of the `img1_mc` movie clip. After the tweened animation is complete and calls the `Tween.onMotionFinished` event handler, the tween is played again by calling the `Tween.start()` method. The result is a movie clip that moves across the Stage from left to right and then starts the animation over again when it reaches the end of the Stage. A movie clip instance named `img1_mc`, is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width, 50, false);

myTween.onMotionFinished = function() {
    myTween.start();
};
```

Tween.stop()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.stop()
```

Returns

Nothing.

Description

Method; stops the play of a tweened animation at its current value.

Example

The following example applies a tweened animation to the `_x` property of the `img1_mc` movie clip. A `stopTween_btn` movie clip's `onRelease()` handler calls the `Tween.stop()` method to stop the tweened animation and a `resumeTween_btn` movie clip's `onRelease()` handler calls the `Tween.resume()` method to resume the animation from a stopped position. A movie clip instance named `img1_mc`, a movie clip instance named `stopTween_btn`, and a movie clip instance named `resumeTween_btn`, are required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, img1_mc._width, Stage.width, 8,
    true);

myTween.onMotionFinished = function() {
    myTween.start();
};

stopTween_btn.onRelease = function() {
    myTween.stop();
};

resumeTween_btn.onRelease = function() {
    myTween.resume();
};
```

Tween.time

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

`tweenInstance.time`

Description

Property (read-only); the current number of seconds that have passed within the duration of the animation if the `useSeconds` parameter was set to `true` when creating the `Tween` instance. If the `useSeconds` parameter of the animation was set to `false`, `Tween.time` returns the current number of frames that have passed in the `Tween` object animation.

Example

The following example returns the time value of a Tween instance. A movie clip instance named `img1_mc`, is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new mx.transitions.Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone, 0, Stage.width-img1_mc._width, 10,
    false);
myTween.onMotionChanged = function() {
    var myCurrentTime:Number = myTween.time;
    var myCurrentDuration:Number = myTween.duration;
    trace(myCurrentTime + " of " + myCurrentDuration);
};
```

Tween.toString()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

tweenInstance.toString()

Returns

The following string is returned: "[Tween]".

Description

Method; returns the class name, "[Tween]".

Example

In the following example a Tween object is identified by calling the `Tween.toString()` method to return "[Tween]", identifying the object's class name. A movie clip instance named `img1_mc`, is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_alpha",
    mx.transitions.easing.None.easeNone,0, 100, 50, false);
var theClassName:String = myTween.toString();
trace(theClassName);
```

Tween.yoyo()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
tweenInstance.yoyo()
```

Returns

Nothing.

Description

Method; instructs the tweened animation to play in reverse from its last direction of tweened property increments. If this method is called before a Tween object's animation is complete, the animation abruptly jumps to the end of its play and then plays in a reverse direction from that point. You can achieve an effect of an animation completing its entire play and then reversing its entire play by calling the `Tween.yoyo()` method within a [Tween.onMotionFinished](#) event handler. This process ensures that the reverse effect of the `Tween.yoyo` method does not begin until the current tweened animation is complete. See [Tween.onMotionFinished on page 1325](#).

Example

In the following example, a handler is triggered by the [Tween.onMotionFinished](#) event and tells the Tween instance to animate the `img1_mc` movie clip in a reverse direction by calling the `Tween.yoyo()` method. The result is a movie clip that moves from the left of the Stage to the right and then reverses direction, moving from right to left in an animation loop. A movie clip instance named `img1_mc`, is required on the Stage for this example:

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
    mx.transitions.easing.None.easeNone,0, Stage.width, 4, true);
myTween.onMotionFinished = function() {
    myTween.yoyo();
};
```

The UIComponent class does not represent a visual component; it contains methods, properties, and events that allow Macromedia components to share some common behavior. All version 2 Macromedia Component Architecture components extend UIComponent. The UIComponent class lets you do the following:

- Receive focus and keyboard input
- Enable and disable components
- Resize by layout

To use the methods and properties of UIComponent, you call them directly from whichever component you are using. For example, to call `UIComponent.setFocus()` from the `RadioButton` component, you would write the following code:

```
myRadioButton.setFocus();
```

You only need to create an instance of UIComponent if you are using version 2 of the Macromedia Component Architecture to create a new component. Even in that case, UIComponent is often created implicitly by other subclasses such as `Button`. If you do need to create an instance of UIComponent, use the following code:

```
class MyComponent extends mx.core.UIComponent;
```

UIComponent class (API)

Inheritance `MovieClip` > [UIObject class](#) > `UIComponent`

ActionScript Class Name `mx.core.UIComponent`

The methods, properties, and events of the UIComponent class allow you to control the common behavior of Flash visual components.

Method summary for the UIComponent class

The following table lists methods of the UIComponent class.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the UIObject class

The following table lists the methods the UIComponent class inherits from the UIObject class. When calling these methods from the UIComponent object, use the form *UIComponentInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the `UIComponent` class

The following table lists properties of the `UIComponent` class.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `UIObject` class

The following table lists the properties the `UIComponent` class inherits from the `UIObject` class. When accessing these properties from the `UIComponent` object, use the form `UIComponentInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the UIComponent class

The following table lists events of the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the UIObject class

The following table lists the events the UIComponent class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

UIComponent.enabled

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.enabled

Description

Property; indicates whether the component can (`true`) or cannot (`false`) accept focus and mouse input. The default value is `true`.

Example

The following example sets the `enabled` property of a `CheckBox` component to `false`:

```
checkBoxInstance.enabled = false;
```

UIComponent.focusIn

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.focusIn = function(eventObj:Object) {
    //...
};
componentInstance.addEventListener("focusIn", listenerObject);
```

Usage 2:

```
on (focusIn) {
    // ...
}
```

Description

Event; notifies listeners that the object has received keyboard focus.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `focusIn`), and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following code disables a Button component, `btn`, while a user types the TextInput component, `txt`, and enables the button when the user clicks it:

```
var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
    _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
    _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);
```

See also

[EventDispatcher.addEventListener\(\)](#), [UIComponent.focusOut](#)

UIComponent.focusOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
on(focusOut){
    ...
}
listenerObject = new Object();
listenerObject.focusOut = function(eventObject){
    ...
}
componentInstance.addEventListener("focusOut", listenerObject)
```

Description

Event; notifies listeners that the object has lost keyboard focus.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `focusOut`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following code disables a Button component, btn, while a user types the TextInput component, txt, and enables the button when the user clicks it:

```
var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
    _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
    _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);
```

See also

[EventDispatcher.addEventListener\(\)](#), [UIComponent.focusIn](#)

UIComponent.getFocus()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getFocus();
```

Parameters

None.

Returns

A reference to the object that currently has focus.

Description

Method; returns a reference to the object that has keyboard focus.

Example

The following code returns a reference to the object that has focus and assigns it to the `tmp` variable:

```
var tmp = checkbox.getFocus();
```

UIComponent.keyDown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
on(keyDown){  
    ...  
}  
listenerObject = new Object();  
listenerObject.keyDown = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("keyDown", listenerObject)
```

Description

Event; notifies listeners when a key is pressed. This is a very low-level event that you should not use unless necessary, because it can affect system performance.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `keyDown`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following code makes an icon blink when a key is pressed:

```
formListener.handleEvent = function(eventObj)
{
    form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyDown", formListener);
```

UIComponent.keyUp

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
on(keyUp){
    ...
}
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
    ...
}
componentInstance.addEventListener("keyUp", listenerObject)
```

Description

Event; notifies listeners when a key is released. This is a low-level event that you should not use unless necessary, because it can affect system performance.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `keyUp`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following code makes an icon blink when a key is released:

```
formListener.handleEvent = function(eventObj)
{
    form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyUp", formListener);
```

UIComponent.setFocus()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setFocus();
```

Parameters

None.

Returns

Nothing.

Description

Method; sets the focus to this component instance. The instance with focus receives all keyboard input.

Example

The following code gives focus to the checkbox instance:

```
checkbox.setFocus();
```

UIComponent.tabIndex

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

instance.tabIndex

Description

Property; a number indicating the tabbing order for a component in a document.

Example

The following code sets the value of `tmp` to the `tabIndex` property of the checkbox instance:

```
var tmp = checkbox.tabIndex;
```

ActionScript Class Name mx.events.UIEventDispatcher

Inheritance [EventDispatcher class](#) > UIEventDispatcher

The UIEventDispatcher class is mixed in to the UIComponent class and allows components to emit certain events.

If you want an object that doesn't inherit from UIComponent to dispatch certain events, you can use UIEventDispatcher.

Method summary for the UIEventDispatcher class

The following table lists the method of the UIEventDispatcher class.

Method	Description
UIEventDispatcher.removeEventListener()	Removes a registered listener from a component instance. This method overrides the <code>eventDispatcher.removeEventListener()</code> method.

Methods inherited from the EventDispatcher class

The following table lists the methods the UIEventDispatcher class inherits from the EventDispatcher class. When calling these methods from the UIEventDispatcher object, use the form *UIEventDispatcherInstance.methodName*.

Method	Description
EventDispatcher.addEventListener()	Registers a listener to a component instance.
EventDispatcher.dispatchEvent()	Dispatches an event to all registered listeners.

Event summary for the UIEventDispatcher class

The following table lists events of the UIEventDispatcher class.

Method	Description
UIEventDispatcher.keyDown	Broadcast when a key is pressed.
UIEventDispatcher.keyUp	Broadcast when a pressed key is released.
UIEventDispatcher.load	Broadcast when a component loads into Flash Player.
UIEventDispatcher.mouseDown	Broadcast when the mouse is pressed.
UIEventDispatcher.mouseOut	Broadcast when the mouse is moved off a component instance.
UIEventDispatcher.mouseOver	Broadcast when the mouse is moved over a component instance.
UIEventDispatcher.mouseUp	Broadcast when the mouse is pressed and released.
UIEventDispatcher.unload	Broadcast when a component is unloaded from Flash Player.

UIEventDispatcher.keyDown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.keyDown = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("keyDown", listenerObject)
```

Description

Event; broadcast to all registered listeners when a key is pressed and the Flash application has focus.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.keyUp

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("keyUp", listenerObject)
```

Description

Event; broadcast to all registered listeners when a key that was pressed is released and the Flash application has focus.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.load

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.load = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("load", listenerObject)
```

Description

Event; broadcast to all registered listeners when a component is loaded into Flash Player.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseDown

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.mouseDown = function(eventObject){  
    // Insert your code here.  
}  
componentInstance.addEventListener("mouseDown", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is pressed.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseOut

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseOut = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("mouseOut", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is moved off a component instance.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseOver

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseOver = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("mouseOver", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is moved over a component instance.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseUp

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseUp = function(eventObject){
    // Insert your code here.
}
componentInstance.addEventListener("mouseUp", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is pressed and released.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“EventDispatcher class” on page 499](#).

UIEventDispatcher.removeEventListener()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
componentInstance.removeEventListener(event, listener)
```

Parameters

event A string that is the name of the event.

listener A reference to a listener object or function.

Returns

Nothing.

Description

Method; unregisters a listener object from a component instance that is broadcasting an event. This method overrides the `EventDispatcher.removeEventListener()` event found in the `EventDispatcher` base class.

UIEventDispatcher.unload

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.unload = function(eventObject){  
    // Insert your code here.  
}  
componentInstance.addEventListener("unload", listenerObject)
```

Description

Event; broadcast to all registered listeners when a component is unloaded from Flash Player.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

Inheritance MovieClip > UIObject

ActionScript Class Name mx.core.UIObject

UIObject is the base class for all version 2 of the Macromedia Component Architecture components; it is not a visual component. The UIObject class wraps the ActionScript MovieClip object and contains functions and properties that allow version 2 components to share some common behavior. Wrapping the MovieClip class allows Macromedia to add new events and extend functionality in the future without breaking content. Wrapping the MovieClip class also allows users who aren't familiar with the traditional Flash concepts of "movie" and "frame" to use properties, methods, and events to create component-based applications without learning those concepts.

The UIObject class implements the following:

- Styles
- Events
- Resize by scaling

To use the methods and properties of the UIObject class, you call them directly from whichever component you are using. For example, to call the `UIObject.setSize()` method from the RadioButton component, you would write the following code:

```
myRadioButton.setSize(30, 30);
```

You only need to create an instance of UIObject if you are using version 2 of the Macromedia Component Architecture to create a new component. Even in that case, UIObject is often created implicitly by other subclasses like Button. If you do need to create an instance of UIObject, use the following code:

```
class MyComponent extends UIObject;
```

Method summary for the UIObject class

The following table lists methods of the UIObject class.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createLabel()</code>	Creates a TextField subobject, for use when creating components.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the UIObject class

The following table lists properties of the UIObject class.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.

Property	Description
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the UIObject class

The following table lists events of the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

UIObject.bottom

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.bottom

Description

Property (read-only); a number indicating the bottom position of the object, in pixels, relative to its parent's bottom. To set this property, call `UIObject.move()`.

Example

This example moves the check box so it aligns under the bottom edge of the list box:

```
myCheckbox.move(myCheckbox.x, form.height - listbox.bottom);
```

UIObject.createClassObject()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.createClassObject(className, instanceName, depth,  
                                     initObject)
```

Parameters

className An object indicating the class of the new instance.

instanceName A string indicating the instance name of the new instance.

depth A number indicating the depth of the new instance.

initObject An object containing initialization properties for the new instance.

Returns

A `UIObject` object that is an instance of the specified class.

Description

Method; creates an instance of a component at runtime. Use the `import` statement and specify the class package name before you call this method. In addition, the component must be in the FLA file's library.

Example

The following code imports the assets of the Button component and then makes a subobject of the Button component:

```
import mx.controls.Button;
createClassObject(Button,"button2",5,{label:"Test Button"});
```

The following example creates a CheckBox object:

```
import mx.controls.CheckBox;
form.createClassObject(CheckBox, "cb", 0, {label:"Check this"});
```

You can also use the following syntax to specify the class package name:

```
createClassObject(mx.controls.Button, "button2", 5, {label:"Test Button"});
```

UIObject.createLabel()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
createLabel(name, depth, text)
```

Parameters

name A string for the instance name.

depth A number indicating the depth of the new instance.

text The text for the label.

Returns

A TextField object.

Description

Method; creates a `TextField` subobject. Used by most components to get a lightweight text object to display text in the component while inheriting sizing and style methods and properties of the component. This method is used to create new components. The `TextField` that is created is the same as a `TextField` object created using

`MovieClip.createTextField()`, but has the added benefit of inheriting useful properties and methods from the parent `UIObject`.

A `TextField` that uses `UIObject.createLabel()` to create within a component can take advantage of the following inherited `UIObject` methods to set sizing and styles within the context of the parent `UIObject`:

- `TextField.getPreferredHeight() : Number`
- `TextField.getPreferredWidth() : Number`
- `TextField.setStyle(styleName : String, value)`
- `TextField.setSize(width : Number, height : Number)`
- `TextField.setValue(text : String)`

NOTE

`TextFields` created with `UIObject.createLabel()` have an initial `TextField._visible` property of `false`. This property is used to avoid flickering that may occur while `UIObject.setSize()` is called by the parent component. The `TextField._visible` property is set to `true` when the `UIObject.draw()` is called after the parent component's children objects are resized.

For more information, see the `MultilineCell.as` file example in [“Simple cell renderer example”](#) on page 112.

Example

The following example creates a `TextField` instance called `multilineLabel` within a component's `UIComponent.createChildren()` method:

```
public function createChildren():Void {
    var myTextField_txt:TextField = this.createLabel("multilineLabel", 900,
        "Hello World");
    // Set the fontSize style attribute of the TextField.
    myTextField_txt.setStyle("fontSize", 18);
    // Set the TextField's initial size.
    myTextField_txt.setSize(myTextField_txt.getPreferredWidth(),
        myTextField_txt.getPreferredHeight());
    // Set the TextField's initial location in the center of the Stage.
    myTextField_txt._x = (Stage.width/2) - (myTextField_txt._width/2);
    myTextField_txt._y = (Stage.height/2) - (myTextField_txt._height/2);
}
```


UIObject.createObject()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.createObject(linkageName, instanceName, depth,  
    initObject)
```

Parameters

linkageName A string indicating the linkage identifier of a symbol in the library.

instanceName A string indicating the instance name of the new instance.

depth A number indicating the depth of the new instance.

initObject An object containing initialization properties for the new instance.

Returns

A UIObject object that is an instance of the symbol.

Description

Method; creates a subobject on an object. This method is generally used only by component developers or advanced developers.

Example

The following example creates a CheckBox instance on the form object:

```
form.createObject("CheckBox", "sym1", 0);
```

UIObject.destroyObject()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
destroyObject(instanceName)
```

Parameters

instanceName A string indicating the instance name of the object to be destroyed.

Returns

Nothing.

Description

Method; destroys a component instance.

Example

The following example removes the `TextInput` instance `my_ti` when the button is clicked. With a `Button` and a `TextInput` component in the current document's library, add the following code to the first frame of the main timeline:

```
//Create textinput and button instances
this.createClassObject(mx.controls.TextInput, "my_ti", 1, {text:"Hello
World"});
this.createClassObject(mx.controls.Button, "my_button", 2, {label:"My
Button"});
//Shift button to be below text input
my_button.move(my_ti.left, Stage.height - my_ti.bottom);

//Create Listener Object for button click
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object){
    destroyObject("my_ti");
}
//Add Listener
my_button.addEventListener("click", buttonListener);
```

UIObject.doLater()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.doLater(target, "function")
```

Parameters

target A reference to a timeline that contains the specified function.

function A string indicating a function name to be called after a frame within the component movie clip has passed (so the component's properties set in the Property or Component inspector are available).

Returns

Nothing.

Description

Method; calls a user-defined function only after the component has finished setting all of its properties from the Property inspector or Component inspector. All version 2 components that inherit from UIObject have the `doLater()` method.

Component properties set in the Property inspector or Component inspector may not be immediately available to ActionScript in the timeline. For example, attempting to trace the `label` property from a CheckBox component using ActionScript on the first frame of your SWF file fails without notification, even though the component appears on the Stage as expected.

Although properties that are set in a class or a frame script are available immediately, most properties assigned in the Property inspector or Component inspector are not set until the next frame within the component itself.

Although any approach that delays access of the property will resolve this problem, the simplest and most direct solution is to use the `doLater()` method.

Example

The following example shows how the `doLater()` method is used:

```
// doLater() is called from the component instance
myCheckBox.doLater(this, "delay");

// The function or method called from doLater().

function delay() {
    trace(myCheckBox.label); // The property can now be traced
    // any additional statements go here
}
```

UIObject.draw

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.draw = function(eventObject:Object) {
    // ...
};
componentInstance.addEventListener("draw", listenerObject);
```

Usage 2:

```
on (draw) {
    // ...
}
```

Description

Event; notifies listeners that the object is about to draw its graphics. This is a low-level event that you should not use unless necessary because it can affect system performance.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *draw*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following code redraws the object `form2` when the `form` object is drawn:

```
formListener.draw = function(eventObj:Object) {  
    form2.redraw(true);  
}  
form.addEventListener("draw", formListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

UIObject.getStyle()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getStyle(propertyName)
```

Parameters

propertyName A string indicating the name of the style property (for example, "fontWeight", "borderStyle", and so on).

Returns

The value of the style property. The value can be of any data type.

Description

Method; gets the style property from the style declaration or object. If the style property is an inheriting style, the ancestors of the object may be the source of the style value.

For a list of the styles supported by each component, see the individual component entries. See also “Using global, custom, and class styles in the same document” in *Using Components*.

Example

The following code sets the `ib` instance's `fontWeight` style property to bold if the `cb` instance's `fontWeight` style property is bold:

```
if (cb.getStyle("fontWeight") == "bold") {  
    ib.setStyle("fontWeight", "bold");  
};
```

UIObject.height

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.height

Description

Property (read-only); a number indicating the height of the object, in pixels. To change the height property, call [UIObject.setSize\(\)](#).

Example

The following example increases the check box height:

```
myCheckbox.setSize(myCheckbox.width, myCheckbox.height + 10);
```

UIObject.hide

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.hide = function(eventObject:Object) {
    // ...
};
componentInstance.addEventListener("hide", listenerObject);
```

Usage 2:

```
on (hide) {
    // ...
}
```

Description

Event; broadcast when the object's `visible` property is changed from `true` to `false`.

Example

The following handler displays a message in the Output panel when the object it's attached to becomes invisible.

```
on (hide) {  
    trace("I've become invisible.");  
}
```

See also

[UIObject.reveal](#)

UIObject.invalidate()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.invalidate()
```

Returns

Nothing.

Description

Method; marks the object so it is redrawn on the next frame interval.

This method is primarily useful to developers of new custom components. A custom component is likely to support a number of operations that change the component's appearance.

Often, the best way to build a component is to centralize the logic for updating the component's appearance in the `draw()` method. If the component has a `draw()` method, you can call `invalidate()` on the component to redraw it. (For information on defining a `draw()` method, see "Defining the `draw()` method" in *Using Components*.)

All operations that change the component's appearance can call `invalidate()` instead of redrawing the component themselves. This has some advantages: code isn't duplicated unnecessarily, and multiple changes can easily be batched up into one redraw, instead of causing multiple, redundant redraws.

Example

The following example marks the `ProgressBar` instance `pBar` for redrawing:

```
pBar.invalidate();
```

UIObject.left

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.left
```

Description

Property (read-only); a number indicating the left edge of the object, in pixels, relative to its parent. To set this property, call `UIObject.move()`.

UIObject.load

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.load = function(eventObject:Object) {
    // ...
};
componentInstance.addEventListener("load", listenerObject);
```


Usage 2:

```
on (load) {  
    //...  
}
```

Description

Event; notifies listeners that the subobject for this object is being created.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `load`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following example creates an instance of `MySymbol` after the `form` instance is loaded:

```
var formListener:Object = new Object();  
formListener.load = function(eventObj:Object) {  
    form.createObject("MySymbol", "sym1", 0);  
};  
form.addEventListener("load", formListener);
```

UIObject.move

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.move = function(eventObject:Object):Void {
    // ...
};
componentInstance.addEventListener("move", listenerObject);
```

Usage 2:

```
on (move) {
    // ...
}
```

Description

Event; notifies listeners that the object has moved.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *move*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following example calls the `move()` method to reposition a `Button` component, `my_button`, from its current position to the upper-left corner (10,10) of the Stage:

```
var my_button:mx.controls.Button;
my_button.addEventListener("move", doMove);
function doMove(evt_obj:Object):Void {
    trace(evt_obj.target + " moved from {oldX:" + evt_obj.oldX + ", oldY:" +
        evt_obj.oldY + "} to {x:" + evt_obj.target.x + ", y:" + evt_obj.target.y
        + "}");
}
my_button.move(10, 10);
```

UIObject.move()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.move(x, y, noEvent)

Parameters

x A number that indicates the position of the object's upper left corner, relative to its parent.

y A number that indicates the position of the object's upper left corner, relative to its parent.

noEvent A Boolean value that indicates whether the move event should be dispatched.

Returns

Nothing.

Description

Method; moves the object to the requested position. Pass only integer values to [UIObject.move\(\)](#), or the component may appear fuzzy.

Example

The following example moves the check box 10 pixels to the right:

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

The following example calls the `move()` method to reposition a Button component, `my_button`, from its current position to the upper-left corner (10,10) of the Stage:

```
var my_button:mx.controls.Button;
my_button.addEventListener("move", doMove);
function doMove(evt_obj:Object):Void {
    trace(evt_obj.target + " moved from {oldX:" + evt_obj.oldX + ", oldY:" +
        evt_obj.oldY + "} to {x:" + evt_obj.target.x + ", y:" + evt_obj.target.y
        + "}");
}
my_button.move(10, 10);
```

UIObject.redraw()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.redraw(*always*)

Parameters

always A Boolean value. If `true`, the method draws the object, even if `invalidate()` wasn't called. If `false`, the method draws the object only if `invalidate()` was called.

Returns

Nothing.

Description

Method; forces validation of the object so that it is drawn in the current frame.

Example

The following example creates a check box and a button and draws them because other scripts are not expected to modify the form:

```
form.createClassObject(mx.controls.CheckBox, "cb", 0);  
form.createClassObject(mx.controls.Button, "b", 1);  
form.redraw(true)
```

UIObject.resize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object) {
    // ...
};
componentInstance.addEventListener("resize", listenerObject);
```

Usage 2:

```
on (resize) {
    // ...
}
```

Description

Event; notifies listeners that an object has been resized.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *resize*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following example calls the `setSize()` method to make `sym1` half the width and a fourth of the height when `form` is moved:

```
var formListener:Object = new Object();
formListener.resize = function(eventObj:Object):Void {
    form.sym1.setSize(sym1.width / 2, sym1.height / 4);
};
form.addEventListener("resize", formListener);
```

The following example calls the `setSize()` method to resize a `Button` component, `my_button`, to 200 pixels wide by 100 pixels high:

```
var my_button:mx.controls.Button;

my_button.addEventListener("resize", doSize);
function doSize(evt_obj:Object):Void {
    trace(evt_obj.target + " resized from {oldWidth:" + evt_obj.oldWidth + ",
        oldHeight:" + evt_obj.oldHeight + "} to {width:" + evt_obj.target.width +
        ", height:" + evt_obj.target.height + "}");
}
my_button.setSize(200, 100);
```

UIObject.reveal

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.reveal = function(eventObject:Object) {
    // ...
};
componentInstance.addEventListener("reveal", listenerObject);
```

Usage 2:

```
on (reveal) {
    // ...
}
```

Description

Event; broadcast when the object's `visible` property changes from `false` to `true`.

Example

The following handler displays a message in the Output panel when the object it's attached to becomes visible.

```
on (reveal) {  
    trace("I've become visible.");  
}
```

See also

[UIObject.hide](#)

UIObject.right

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.right

Description

Property (read-only); a number indicating the right edge of the object, in pixels, relative to its parent's right edge. To set this property, call [UIObject.move\(\)](#).

Example

The following example moves the check box so it aligns under the right edge of the list box:

```
myCheckbox.move(form.width - listbox.right, myCheckbox.y);
```

UIObject.scaleX

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.scaleX

Description

Property; a number indicating the scaling factor in the x direction of the object, relative to its parent.

Example

The following example makes the check box twice as wide and sets the `tmp` variable to the horizontal scale factor:

```
checkbox.scaleX = 200;  
var tmp:Number = checkbox.scaleX;
```

UIObject.scaleY

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.scaleY

Description

Property; a number indicating the scaling factor in the y direction of the object, relative to its parent.

Example

The following example makes the check box twice as high and sets the `tmp` variable to the vertical scale factor:

```
checkbox.scaleY = 200;  
var tmp:Number = checkbox.scaleY;
```


UIObject.setSize()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.setSize(width, height, noEvent)

Parameters

width A number that indicates the width of the object, in pixels.

height A number that indicates the height of the object, in pixels.

noEvent A Boolean value that indicates whether the resize event should be dispatched.

Returns

Nothing.

Description

Method; resizes the object to the requested size. You should pass only integer values to `UIObject.setSize()`, or the component may appear fuzzy. This method (as with all methods and properties of `UIObject`) is available from any component instance.

When you call this method on a `ComboBox` instance, the combo box is resized and the `rowHeight` property of the contained list also changes.

NOTE

Some components allow you to modify height or width dimensions only. For example, the `CheckBox` and `RadioButton` components do not allow you to modify the height.

Example

The following example resizes the `pBar` component instance to 100 pixels wide and 100 pixels high:

```
pBar.setSize(100, 100);
```

The following example calls the `setSize()` method to resize the `my_button` Button component to 200 pixels wide by 100 pixels high:

```
var my_button:mx.controls.Button;

my_button.addEventListener("resize", doSize);
function doSize(evt_obj:Object):Void {
    trace(evt_obj.target + " resized from {oldWidth:" + evt_obj.oldWidth + ",
    oldHeight:" + evt_obj.oldHeight + "} to {width:" + evt_obj.target.width +
    ", height:" + evt_obj.target.height + "}");
}
my_button.setSize(200, 100);
```

UIObject.setSkin()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setSkin(id, linkageName)
```

Parameters

id A number indicating the depth of the skin within the component.

linkageName A string indicating an asset in the library.

Returns

A reference to the movie clip (skin) that was attached.

Description

Method; sets a skin in the component instance. Use this method in a component's class file when you are creating a component. For more information, see "About assigning skins" in *Using Components*.

You cannot use this method to set a component's skins at runtime (for example, the way you set a component's styles at runtime).

Example

This example is a code snippet from the class file of a new component, called Shape. It creates a variable, `themeShape` and sets it to the Linkage identifier of the skin. In the `createChildren()` method, the `setSkin()` method is called and passed the id 1 and the variable that holds the linkage identifier of the skin:

```
class Shape extends UIComponent {  
  
    static var symbolName:String = "Shape";  
    static var symbolOwner:Object = Shape;  
    var className:String = "Shape";  
  
    var themeShape:String = "circle_skin"  
  
    function Shape() {  
    }  
  
    function init(Void):Void {  
        super.init();  
    }  
  
    function createChildren():Void {  
        setSkin(1, themeShape);  
        super.createChildren();  
    }  
}
```

UIObject.setStyle()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.setStyle(propertyName, value)

Parameters

propertyName A string indicating the name of the style property. Supported styles vary depending on the component. Each component has a different set of styles that you can set. For example, [“Customizing the TextArea component” on page 1180](#) shows a table of styles, including `fontWeight`. So, for a `TextArea` component, you can use `fontWeight` as the *propertyName* parameter.

value The value of the property. If the value is a string, it must be enclosed in quotation marks.

Returns

Nothing.

Description

Method; sets the style property on the style declaration or object. If the style property is an inheriting style, the children of the object are notified of the new value.

For a list of the styles that each component supports, see individual component entries. For example, Button component styles are listed in [“Using styles with the Button component” on page 94](#).

To enhance performance, you can change styles before they are loaded, calculated, and applied to the objects in your SWF file. If you can change styles before the styles are loaded and calculated, you do not have to call `setStyle`.

Macromedia recommends that you set properties on each object because objects are instantiated to improve performance when you use styles. When you dynamically attach instances to the Stage, set properties in the `initObj` parameter for the call that you make to `UIObject.createClassObject()`, as shown in the following ActionScript:

```
createClassObject(ComponentClass, "myInstance", 0, {styleName:"myStyle",  
color:0x99CCFF});
```

NOTE

This example uses the `myStyle` custom style declaration. To change multiple properties, or change properties for multiple component instances, create a custom style declaration. Flash renders a component using a custom style declaration faster than it renders a component using `UIObject.setStyle()` for multiple properties. For more information, see [“Setting custom styles for groups of components” in *Using Components*](#).

Example

The following code sets the `fontWeight` style property of the `cb` check box instance to bold:

```
cb.setStyle("fontWeight", "bold");
```

UIObject.top

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.top

Description

Property (read-only); a number indicating the top edge of the object, in pixels, relative to its parent. To set this property, call [UIObject.move\(\)](#).

UIObject.unload

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.unload = function(eventObject:Object):Void {
    // ...
};
componentInstance.addEventListener("unload", listenerObject);
```

Usage 2:

```
on (unload) {
    // ...
}
```

Description

Event; notifies listeners that the subobjects of this object are being unloaded.

The first usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `unload`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a component instance.

Example

The following example deletes `sym1` when the `unload` event is triggered:

```
function doUnload():Void {
    form.destroyObject(sym1);
}
form.addEventListener("unload", doUnload);
```

UIObject.visible

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.visible

Description

Property; a Boolean value indicating whether the object is visible (`true`) or not (`false`).

Example

The following example makes the `myLoader` loader instance visible:

```
myLoader.visible = true;
```

UIObject.width

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.width

Description

Property (read-only); a number indicating the width of the object, in pixels. To change the width, call `UIObject.setSize()`.

Example

The following example makes the check box wider:

```
myCheckbox.setSize(myCheckbox.width + 10, myCheckbox.height);
```

UIObject.x

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.x

Description

Property (read-only); a number indicating the left edge of the object, in pixels. To set this property, call `UIObject.move()`.

Example

The following example moves the check box 10 pixels to the right:

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

UIObject.y

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.y

Description

Property (read-only); a number indicating the top edge of the object, in pixels. To set this property, call [UIObject.move\(\)](#).

Example

The following example moves the check box down 10 pixels:

```
myCheckbox.move(myCheckbox.x, myCheckbox.y + 10);
```


The UIScrollBar component allows you to add a scroll bar to a text field. You can add a scroll bar to a text field while authoring, or at runtime with ActionScript.

The UIScrollBar component functions like any other scroll bar. It contains arrow buttons at either end and a scroll track and scroll box (thumb) in between. It can be attached to any edge of a text field and used both vertically and horizontally.

Using the UIScrollBar component

To use the UIScrollBar component, verify that object snapping is turned on (View > Snapping > Snap to Objects). Then create a text input field on the Stage and drag the UIScrollBar component from the Components panel to any quadrant of the text field's bounding box.

If the length of the scroll bar is smaller than the combined size of its scroll arrows, it is not displayed correctly. One of the arrow buttons becomes hidden behind the other. Flash does not provide error checking for this. In this case it is a good idea to hide the scroll bar with ActionScript. If the scroll bar is sized so that there is not enough room for the scroll box (thumb), Flash makes the scroll box invisible.

Unlike many other components, the UIScrollBar component can receive continuous mouse input, such as when the user holds the mouse button down, rather than requiring repeated clicks.

There is no keyboard interaction with the UIScrollBar component.

UIScrollBar parameters

You can set the following authoring parameters for each UIScrollBar instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

_targetInstanceName indicates the name of the text field instance that the UIScrollBar component is attached to.

horizontal indicates whether the scroll bar is oriented horizontally (`true`) or vertically (`false`). The default value is `false`.

You can set the following additional parameters for each UIScrollBar component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can write ActionScript to control these and additional options for a UIScrollBar component using its properties, methods, and events. For more information, see [“UIScrollBar class” on page 1395](#).

Creating an application with the UIScrollBar component

The following procedure explains how to add a UIScrollBar component to an application while authoring.

To create an application with the `UIScrollBar` component:

1. Create a dynamic text field and give it an instance name `myText` in the Property inspector.
2. In the Property inspector, set the Line Type of the text input field to Multiline or to Multiline No Wrap if you plan to use the scroll bar horizontally.
3. In Frame 1, use `ActionScript` to add enough text to the field so that users have to scroll to view it all. You could write the following code:

```
myText.text="When the moon is in the seventh house and Jupiter aligns  
with Mars, then peace will guide the planet and love will rule the  
stars."
```

NOTE

Make sure that the text field on the Stage is small enough that you need to scroll it to see all the text. If it isn't, the scroll bar does not appear or may appear simply as two lines with no thumb grip (the part you drag to scroll the content).

4. Verify that object snapping is turned on (View > Snapping > Snap to Objects).
5. Drag a `UIScrollBar` instance from the Components panel onto the text input field near the side you want to attach it to. The component must overlap with the text field when you release the mouse in order for it to be properly bound to the field.

The `_targetInstanceName` property of the component is automatically populated with the text field instance name in the Property and Component inspectors. If it does not appear in the Parameters tab, you may not have overlapped the `UIScrollBar` instance enough.

6. Select Control > Test Movie.

The application runs, and the scroll bar scrolls the contents of the text field.

You can also create a `UIScrollBar` component instance and associate it with a text field at runtime with `ActionScript`.

The following code creates a vertically oriented `UIScrollBar` instance and attaches it to the right side of a text field instance named `my_txt` and sets the size of the scroll bar to match the size of the text field:

```
/**
 * Requires:
 * - UIScrollBar component in library
 */
// Create text field.
this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field for the scroll bar.
my_sb.setScrollTarget(my_txt);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

Customizing the UIScrollBar component

You can transform a UIScrollBar component horizontally and vertically while authoring and at runtime. However, a vertical UIScrollBar does not allow you to modify the width, and a horizontal UIScrollBar does not allow you to modify the height. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the UIScrollBar class.

NOTE

If you use the `UIObject.setSize()` method, you can change only the height or the width of the instance, depending on whether the instance is a horizontal or a vertical scroll bar. Therefore the `setSize()` method ignores either the `height` or the `width` parameter.

Note, however, that with the Halo theme, the width of a vertically oriented scroll bar must be 16 pixels, and the height of a horizontally oriented scroll bar must also be 16 pixels. These dimensions are determined strictly by the current theme used with the scroll bar. Only the dimension of a scroll bar that corresponds to its length can be changed.

You can customize the appearance of a UIScrollBar instance by using styles and skins.

Using styles with the UIScrollBar component

The UIScrollBar component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>scrollTrackColor</code>	Sample	The background color for the scroll track. The default value is <code>0xCCCCCC</code> (light gray).
<code>symbolColor</code>	Sample	The color of the up and down scroll arrows. The default value is <code>0x000000</code> (black).
<code>symbolDisabledColor</code>	Sample	The color of the up and down scroll arrows in a disabled scroll bar. The default value is <code>0x848384</code> (dark gray).

Using skins with the UIScrollBar component

The UIScrollBar component uses 13 skins for the track, scroll box (thumb), and buttons. To customize these skin elements, edit the symbols in the Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/States folder. For more information, see “About skinning components” in *Using Components*.

Both horizontal and vertical scroll bars use the same vertical skins, and when displaying a horizontal scroll bar the UIScrollBar component rotates the skins as appropriate.

The UIScrollBar component supports the following skin properties.

Property	Description
upArrowUpName	The up (normal) state of the up and left buttons. The default value is ScrollUpArrowUp.
upArrowOverName	The rollover state of the up and left buttons. The default value is ScrollUpArrowOver.
upArrowDownName	The pressed state of the up and left buttons. The default value is ScrollUpArrowDown.
downArrowUpName	The up (normal) state of the down and right buttons. The default value is ScrollDownArrowUp.
downArrowOverName	The rollover state of the down and right buttons. The default value is ScrollDownArrowOver.
downArrowDownName	The pressed state of the down and right buttons. The default value is ScrollDownArrowDown.
scrollTrackName	The symbol used for the scroll bar’s track (background). The default value is ScrollTrack.
scrollTrackOverName	The symbol used for the scroll track (background) when rolled over. The default value is undefined.
scrollTrackDownName	The symbol used for the scroll track (background) when pressed. The default value is undefined.
thumbTopName	The top and left caps of the scroll box (thumb). The default value is ScrollThumbTopUp.
thumbMiddleName	The middle (expandable) part of the thumb. The default value is ScrollThumbMiddleUp.
thumbBottomName	The bottom and right caps of the thumb. The default value is ScrollThumbBottomUp.
thumbGripName	The grip displayed in front of the thumb. The default value is ScrollThumbGripUp.

The following example demonstrates how to put a thin blank line in the middle of the scroll track.

To create movie clip symbols for UIScrollBar skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in *Using Components*.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the ScrollBar Assets folder to the library for your document.
4. Expand the ScrollBar Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the ScrollTrack symbol.
6. Customize the symbol as desired.
For example, draw a black rectangle in the middle of the track using a 1 x 4 rectangle at (8,0).
7. Repeat steps 5-6 for all symbols you want to customize.
For example, draw the same line on the ScrollTrackDisabled symbol.
8. Click the Back button to return to the main timeline.
9. Create an input type TextField instance on the Stage.
10. Drag a UIScrollBar component to the TextField instance.
11. Select Control > Test Movie.

UIScrollBar class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > ScrollBar > UIScrollBar

ActionScript Class Name mx.controls.UIScrollBar

The properties of the UIScrollBar class let you adjust the scroll position and the amount of scrolling that occurs when the user clicks the scroll arrows or the scroll track.

Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.UIScrollBar.version);
```

NOTE

The code `trace(myUIScrollBarInstance.version);` returns `undefined`.

Method summary for the `UIScrollBar` class

The following table lists the method of the `UIScrollBar` class.

Method	Description
<code>UIScrollBar.setScrollProperties()</code>	Sets the scroll range of the scroll bar and the size of the text field that the scroll bar is attached to.
<code>UIScrollBar.setScrollTarget()</code>	Assigns the scroll bar to a text field.

Methods inherited from the `UIObject` class

The following table lists the methods the `UIScrollBar` class inherits from the `UIObject` class. When calling these methods from the `UIScrollBar` object, use the form `UIScrollBarInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it is redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.

Method	Description
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the UIScrollBar class inherits from the UIComponent class. When calling these methods from the UIScrollBar object, use the form *UIScrollBarInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the UIScrollBar class

The following table lists properties of the UIScrollBar class.

Property	Description
UIScrollBar.lineScrollSize	The number of lines or pixels scrolled when the user clicks the arrow buttons of the scroll bar.
UIScrollBar.pageScrollSize	The number of lines or pixels scrolled when the user clicks the track of the scroll bar.
UIScrollBar.scrollPosition	The current scroll position of the scroll bar.
UIScrollBar._targetInstanceName	The instance name of the text field associated with the UIScrollBar instance.
UIScrollBar.horizontal	A Boolean value indicating whether the scroll bar is oriented vertically (<code>false</code>), the default, or horizontally (<code>true</code>).

Properties inherited from the UIObject class

The following table lists the properties the UIScrollView class inherits from the UIObject class. When accessing these properties from the UIScrollView object, use the form *UIScrollViewInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only; the position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the UICComponent class

The following table lists the properties the UIScrollView class inherits from the UICComponent class. When accessing these properties from the UIScrollView object, use the form *UIScrollViewInstance.propertyName*.

Property	Description
<code>UICComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UICComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the UIScrollView class

The following table lists the event of the UIScrollView class.

Event	Description
UIScrollView.scroll	Broadcast when any part of the scroll bar is clicked.

Events inherited from the UIObject class

The following table lists the events the UIScrollView class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the UIScrollView class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

UIScrollBar.horizontal

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.horizontal
```

Description

Property; indicates whether the scroll bar is oriented vertically (`false`) or horizontally (`true`).

This property can be tested and set. The default value is `false`.

Example

The following example uses the `horizontal` property to set the scroll bar named `my_sb` to a horizontal orientation and displays the text in the `TextField` component `my_txt`:

```
/**
 * Requires:
 * - UIScrollBar component in library
 */
// Create the text field.
this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = false;

my_txt.text = "Mary had a little lamb whose fleece " +
"was white as snow and everywhere that Mary went the " +
"lamb was sure to go."

// Create scroll bar.
this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);
my_sb.horizontal = true;

// Set the target text field for the scroll bar.
my_sb.setScrollTarget(my_txt);
// Size it to match the text field.
my_sb.setSize(my_txt._width, 16);

// Move it to the bottom of the text field.
my_sb.move(my_txt._x, my_txt._y + my_txt._height);
```

UIScrollBar.lineScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollBarInstance.lineScrollSize

Description

Property; gets or sets the number of lines or pixels scrolled when the user clicks the arrow buttons of the UIScrollBar component. If the scroll bar is oriented vertically, the value is a number of lines. If the scroll bar is oriented horizontally, the value is a number of pixels.

The default value is 1.

Example

The following example creates a scroll bar to scroll text in a text field, which it loads from a web page. The example sets the `lineScrollSize` property to scroll two lines at a time for each click of an arrow button:

```
/**
 * Requires:
 *   - UIScrollBar component in library
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field.
my_sb.setScrollTarget(my_txt);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Scroll 2 lines per click on scroll arrow.
my_sb.lineScrollSize = 2;

// Scroll 5 lines per click on scroll track.
my_sb.pageScrollSize = 5;
```

```
// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

UIScrollBar.pageScrollSize

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollBarInstance.pageScrollSize

Description

Property; gets or sets the number of lines or pixels scrolled when the user clicks the track of the UIScrollBar component. If the scroll bar is oriented vertically, the value is a number of lines. If the scroll bar is oriented horizontally, the value is a number of pixels.

You can also set this value by passing a *pageSize* parameter with the [UIScrollBar.setScrollTarget\(\)](#) method.

Example

The following example creates a scroll bar to scroll text in a text field that it loads from a web page. The example sets the `pageScrollSize` property to scroll five lines of text each time the user clicks the scroll track:

```
/**
 * Requires:
 * - UIScrollBar component in library
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field.
```

```

my_sb.setScrollTarget(my_txt);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Scroll 2 lines per click of scroll arrow.
my_sb.lineScrollSize = 2;

// Scroll 5 lines per click of scroll track.
my_sb.pageScrollSize = 5;

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

UIScrollBar.scroll

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```

var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
    // ...
};
scrollBarInstance.addEventListener("scroll", listenerObject)

```

Usage 2:

```

on (scroll) {
    // ...
}

```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the scroll bar. The `UISearchBar.scrollPosition` property and the scroll bar's onscreen image are updated before this event is broadcast.

The first usage example uses a dispatcher/listener event model, in which the script is placed on a frame in the timeline that contains the component instance. A component instance (*scrollBarInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event occurs. When the event occurs, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see [EventDispatcher.addEventListener\(\)](#)) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

In addition to the normal properties of the event object (`type` and `target`), the event object for the `scroll` event includes a third property named `direction`. The `direction` property contains a string describing which way the scroll bar is oriented. The possible values for the `direction` property are `vertical` (the default) and `horizontal`.

For more information about the `type` and `target` event object properties, see [“Event objects” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `UISearchBar` component instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `UISearchBar` component instance `myUISearchBarComponent`, sends “`_level0.myUISearchBarComponent`” to the Output panel:

```
on (scroll) {  
    trace(this);  
}
```


Example

The following example implements Usage 1 and creates a listener object called `sbListener` with a `scroll` event handler:

```
/**
 * Requires:
 * - UIScrollBar component in library
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field.
my_sb.setScrollTarget(my_txt);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Create listener object.
var sbListener:Object = new Object();
sbListener.scroll = function(evt_obj:Object){
    // Insert code to handle the "scroll" event.
    trace("text is scrolling");
}
// Add listener.
my_sb.addEventListener("scroll", sbListener);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

The following code implements Usage 2. The code is attached to the `UIScrollBar` component instance and sends a message to the Output panel when the user clicks the scroll bar. The `on()` handler must be attached directly to the `UIScrollBar` instance.

```
on (scroll) {
    trace("UIScrollBar component was clicked");
}
```

UIScrollBar.scrollPosition

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollBarInstance.scrollPosition

Description

Property; gets or sets the current scroll position of the scroll box (thumb) when a new `scrollPosition` value is set. The value of `scrollPosition` depends on whether the `UIScrollBar` instance is being used for vertical or horizontal scrolling.

Set the scrolling of the scroll bar target instance separately, using the following syntax:

```
my_scrollbar._targetInstanceName.scroll = 20;
```

If the `UIScrollBar` instance is being used for vertical scrolling (the most common use), the value of `scrollPosition` is an integer with a range that begins with 0 and ends with a number that is equal to the total number of lines in the text field divided by the number of lines that can be displayed in the text field simultaneously. If `scrollPosition` is set to a number greater than this range, the text field simply scrolls to the end of the text.

To set the scroll box (thumb) to the first line, set `scrollPosition` to 0.

To set the scroll box (thumb) to the end, set `scrollPosition` to the number of lines of text in the text field minus 1. You can determine the number of lines by retrieving the value of the `maxscroll` property of the text field.

If the `UIScrollBar` instance is being used for horizontal scrolling, the value of `scrollPosition` is an integer value ranging from 0 to the width of the text field, in pixels. You can determine the width of the text field in pixels by getting the value of the `maxhscroll` property of the text field.

The default value of `scrollPosition` is 0.

Example

The following example sets the text to position 20:

```
/**
 * Requires:
 * - UIScrollBar and Button components in library
 */
this.createTextField("my_txt", 10, 10, 20, 200, 100);
this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);
this.createClassObject(mx.controls.Button, "my_bt", 30, {label: "Scroll"});

my_txt.wordWrap = true;
my_bt.move(300, 100);

// Set the target text field.
my_sb.setScrollTarget(my_txt);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();

my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};

my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

var scroll_listener = new Object();
scroll_listener.click = function() {
    my_sb.scrollPosition = 20;
    my_txt.scroll = 20;
};
my_bt.addEventListener("click", scroll_listener);
```

UIScrollBar.setScrollProperties()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.setScrollProperties(pageSize, minPos, maxPos)
```

Parameters

pageSize The number of items that can be viewed in the display area. This parameter sets the size of the text field's bounding box. If the scroll bar is vertical, this value is a number of lines of text; if the scroll bar is horizontal, this value is a number of pixels.

minPos This parameter refers to the lowest numbered line of text when the scroll bar is used vertically, or the lowest numbered pixel in the text field's bounding box when the scroll bar is used horizontally. The value is usually 0.

maxPos This value refers to the highest numbered line of text when the scroll bar is used vertically, or the highest numbered pixel in the text field's bounding box when the scroll bar is used horizontally.

Description

Method; sets the scroll range of the scroll bar and the size of the text field that the scroll bar is attached to. This method is primarily useful when you attach a UIScrollBar component to a text field at runtime (using [UIScrollBar.setScrollTarget\(\)](#)) rather than while authoring, and the assignment doesn't cause the text field to broadcast change events. If you use the `replaceText` method to set the text of the text field, you must use `setScrollProperties()` to cause an update of the scroll bars.

The `minPos` and `maxPos` values are used together by the UIScrollBar component to determine the scroll range for the scroll bar and the associated text field.

Example

The following example sets up a UIScrollBar component to display 10 lines of text at a time in the text field out of a range of 0 to 99 lines:

```
my_sb.setScrollProperties(10, 0, 99);
```

UIScrollBar.setScrollTarget()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollBarInstance.setScrollTarget(textInstance)

Parameters

textInstance The text field to assign to the scroll bar.

Description

Method; assigns a UIScrollBar component to a text field instance. If you need to associate a text field and a UIScrollBar component at runtime, use this method.

Example

The following example creates a scroll bar to scroll text in a text field, which it loads from a web page. The example calls the `setScrollTarget()` method to associate the scroll bar `my_sb` with the text field `my_txt`.

```
/**
 * Requires:
 *   - UIScrollBar component in library
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field.
my_sb.setScrollTarget(my_txt);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Scroll 2 lines per click of scroll arrow.
my_sb.lineScrollSize = 2;

// Scroll 5 lines per click of scroll track.
```

```
my_sb.pageSize = 5;

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

UIScrollBar._targetInstanceName

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

scrollBarInstance._targetInstanceName

Description

Property; indicates the instance name of the text field associated with a UIScrollBar component. This property can be tested and set. However, it should not be used to create an association between a text field and a scroll bar. Use [UIScrollBar.setScrollTarget\(\)](#) instead.

Example

The following example creates a scroll bar to scroll text in a text field, which it loads from a web page. The example calls the `trace()` function to display the value of the `targetInstanceName` property.

```
/**
 * Requires:
 * - UIScrollBar component in library
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// Set the target text field.
my_sb.setScrollTarget(my_txt);

trace(my_sb._targetInstanceName);

// Size it to match the text field.
my_sb.setSize(16, my_txt._height);

// Move it next to the text field.
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// Set scroll properties.
my_sb.setScrollProperties(10, 0, 99);

// Load text to display and define onData handler.
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src != undefined) {
        my_txt.text = src;
        my_txt.condenseWhite = true;
    } else {
        my_txt.text = "Error loading text.";
    }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```


Web service classes (Flash Professional only)

The web service classes, which are found in the `mx.services` package, let you access web services that use Simple Object Access Protocol (SOAP). This API is not the same as the `WebServiceConnector` component API. The web service API is a set of classes that can you use only in ActionScript code, and is common to various Macromedia products. In contrast, the `WebServiceConnector` component is an API unique to Flash and provides an ActionScript interface to the visual `WebServiceConnector` component.

The following table lists the classes in the `mx.services` package. These classes are closely integrated, so when first learning about this package, you may want to read the information in the order in which it is presented in the table.

Class	Description
WebService class (Flash Professional only)	Using a Web Service Definition Language (WSDL) file that defines the web service, constructs a new <code>WebService</code> object for calling web service methods and handling callbacks from the web service.
PendingCall class (Flash Professional only)	Object returned from a web service method call that you implement to handle the call's results and faults.
Log class (Flash Professional only)	Optional object used to record activity related to a <code>WebService</code> object.
SOAPCall class (Flash Professional only)	Advanced class that contains information about the web service operation, and provides control over certain behaviors.

Making web service classes available at runtime (Flash Professional only)

In order to make the web service classes available at runtime, the `WebServiceConnector` component must be in your FLA file's library. This component contains the runtime classes that let you work with web services. For details on adding these classes to your FLA file, see Chapter 16, “Data Integration (Flash Professional Only),” in *Using Flash*.

NOTE

These classes are automatically made available to your Flash document when you add a `WebServiceConnector` component to your FLA file.

Log class (Flash Professional only)

ActionScript Class Name `mx.services.Log`

The `Log` class is part of the `mx.services` package and is used with the `WebService` class (see “[WebService class \(Flash Professional only\)](#)” on page 1437). For an overview of the classes in the `mx.data.services` package, see “[Web service classes \(Flash Professional only\)](#)” on page 1413.

You can create a new `Log` object to record activity related to a `WebService` object. To execute code when messages are sent to a `Log` object, use the `Log.onLog()` callback function. There is no log file; the logging mechanism is whatever you have used in the `onLog()` callback function, such as sending the log messages to a `trace()` statement.

The constructor for this class creates a `Log` object that can be passed as an optional parameter to the `WebService` constructor (see “[WebService class \(Flash Professional only\)](#)” on page 1437).

Method summary for the Log class

The following table lists methods of the PendingCall class.

Method	Description
Log.getDateString()	Returns the current date and time as a string in the following format: mm/dd hh:mm:ss used by Log messages.
Log.logInfo()	Generates a <code>Log.onLog</code> event with a designated log level and a designated message.
Log.logDebug()	Generates a <code>Log.onLog</code> event with a log level of <code>Log.DEBUG</code> and a designated message.

Property summary for the Log object

The following table lists properties of the PendingCall class.

Property	Description
Log.level	The category of information that you want to record in the log.
Log.name	A string name identifying the Log object; included in every <code>Log.onLog</code> event message.

Callback summary for the Log object

The following table lists the callback of the Log object.

Callback	Description
Log.onLog()	Called by Flash Player when a log message is sent to a log file.

Constructor for the Log class

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebSvcLog = new Log([logLevel] [, logName]);
```

Parameters

logLevel A level to indicate the category of information that you want to record in the log. Four log levels are available:

- `Log.BRIEF` The log records primary life-cycle event and error notifications. This is the default value.
- `Log.VERBOSE` The log records all life-cycle event and error notifications.
- `Log.DEBUG` The log records metrics and fine-grained events and errors.
- `Log.NONE` The log records nothing. Can be used to temporarily turn off `Log.onLog` events.

logName Optional name that is included with each log message. If you are using multiple Log objects, you can use the log name to determine which log recorded a given message.

Returns

Nothing.

Description

Constructor; creates a Log object. After you create the Log object, you can pass it to a web service to get messages.

Example

You can call the new Log constructor to return a Log object to pass to your web service:

```
// Creates a new log object.
import mx.services.*;
myWebSvcLog = new Log();
myWebSvcLog.onLog = function(msg : String) : Void
{
    myTrace(txt)
}
```

You then pass this Log object as a parameter to the WebService constructor:

```
myWebSvc = new WebService("http://www.myco.com/info.wsdl", myWebSvcLog);
```

As the web services code executes and messages are sent to the Log object, the `onLog()` function of your Log object is called. This is the only place to put code that displays the log messages if you want to see them in real time.

The following are examples of log messages:

```
7/30 15:22:43 [INFO] SOAP: Decoding PendingCall response
7/30 15:22:43 [DEBUG] SOAP: Decoding SOAP response envelope
7/30 15:22:43 [DEBUG] SOAP: Decoding SOAP response body
7/30 15:22:44 [INFO] SOAP: Decoded SOAP response into result [16 millis]
7/30 15:22:46 [INFO] SOAP: Received SOAP response from network [6469 millis]
7/30 15:22:46 [INFO] SOAP: Parsed SOAP response XML [15 millis]
7/30 15:22:46 [INFO] SOAP: Decoding PendingCall response
7/30 15:22:46 [DEBUG] SOAP: Decoding SOAP response envelope
7/30 15:22:46 [DEBUG] SOAP: Decoding SOAP response body
7/30 15:22:46 [INFO] SOAP: Decoded SOAP response into result [16 millis]
```

Log.getDateString()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebSvcLog.getDateAsString()
```

Parameters

None.

Returns

The current date and time as a string in the following format: mm/dd hh:mm:ss.

Description

Function; returns the current date and time as a string in the following format: mm/dd hh:mm:ss. You can use `Log.getDateAsString()` to get the date in the same format that is provided in a log message, or you can record only the date string in a `log.onLog` event handler for use with custom log handling.

Example

The following example creates a new `Log` object, passes it to a new `WebService` object, and handles the log messages, using `Log.getDateString()` to get the time of the log.

```
import mx.services.*;
// Creates a new Log object.
myWebSvcLog = new Log(Log.BRIEF, "myLog");
// Passes the Log object to the web service.
myWebService = new WebService(wsdlURI, myWebSvcLog);
// Handles incoming log messages.
myWebSvcLog.onLog = function(message : String) : Void
{
    trace("A Log Event Occurred At This Time: "+ this.getDateString());
}
```

Log.logInfo()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebSvcLog.logInfo(myMessageString)
```

Parameters

msg A message of type `String` that you want to appear in the resulting log event message.

level A level to indicate the category of information that you want to record in the log.

Four log levels are available:

- `Log.BRIEF` The log records primary life-cycle event and error notifications. This is the default value.
- `Log.VERBOSE` The log records all life-cycle event and error notifications.
- `Log.DEBUG` The log records metrics and fine-grained events and errors.
- `Log.NONE` The log records nothing. Can be used to temporarily turn off `Log.onLog` events.

Returns

Nothing.

Description

Function; generates a log message set by the `msg` parameter at a log level set by the `level` parameter. This method provides a way to create your own log events with any log level.

Example

The following example creates a new `Log` object. An `onLog` event with a message indicating the start of a new log is generated by calling `Log.logDebug()`.

```
import mx.services.*;
// Creates a new Log object.
myWebSvcLog = new Log(Log.VERBOSE, "myLog");

// Handles incoming log messages.
myWebSvcLog.onLog = function(message : String) : Void
{
    trace(message);
}
myWebSvcLog.logInfo("New Log Started");
// Passes the Log object to the web service.
myWebService = new WebService(wsdlURI, myWebSvcLog);
```

Log.logDebug()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebSvcLog.logDebug(msg)
```

Parameters

msg A log message string. The string you provide in this parameter appears as the log message in the resulting log event.

Returns

Nothing.

Description

Function; generates a log message containing `msg` and the message type indicator of `[debug]`. This method provides a way to create your own log events with `[debug]` in the log message, which will be viewable only with a log level setting of `Log.DEBUG`.

The following string is an examples of a debug level log message generated by

```
Log.logDebug():  
12/18 23:20:17 [DEBUG] myLog: My log message
```

Example

The following example creates a new `Log` object. An `onLog` event with a message indicating the start of a new log is generated by calling `Log.logDebug()`.

```
import mx.services.*;  
// Creates a new Log object.  
myWebSvcLog = new Log(Log.DEBUG, "myLog");  
  
// Handles incoming log messages.  
myWebSvcLog.onLog = function(message : String) : Void  
{  
    trace(message);  
}  
// Generates a log message with a log level of Log.DEBUG.  
myWebSvcLog.logDebug("New Log Started");  
// Passes the Log object to the web service.  
myWebService = new WebService(wsdlURI, myWebSvcLog);
```

Log.level

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myLevel_Number = myWebSvcLog.level
```


Description

Property; indicates the category of information that you want to record in the log. Four log levels are available:

- `Log.BRIEF` The log records primary life-cycle event and error notifications. This is the default value. A `Log.level` property set to `Log.BRIEF` returns the number 0.
- `Log.VERBOSE` The log records all life-cycle event and error notifications. A `Log.level` property set to `Log.VERBOSE` returns the number 1.
- `Log.DEBUG` The log records metrics and fine-grained events and errors. A `Log.level` property set to `Log.DEBUG` returns the number 2.
- `Log.NONE` The log records nothing. Can be used to temporarily turn off `Log.onLog` events. A `Log.level` property set to `Log.NONE` returns the number -1.

Although you can set this property directly, usually the `Log.level` property is set as a parameter when you create a new Log object. See [“Log class \(Flash Professional only\)” on page 1414.](#))

Example

The following example creates a new Log object with a `Log.level` property of `Log.DEBUG`. The current `Log.level` property is traced. Then the Log object's `Log.level` property is set to `Log.VERBOSE`.

```
import mx.services.*;
// Creates a new Log object.
myWebSvcLog = new Log(Log.DEBUG, "myLog");
trace("myWebSvcLog.level: "+ myWebSvcLog.level);

// Now change the Log object's level.
myWebSvcLog.level = Log.VERBOSE;
trace("myWebSvcLog.level: "+ myWebSvcLog.level);
```

Log.name

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebServiceName = myWebSvcLog.name
```

Description

Property; a string identifying the Log instance; included in every `Log.onLog` event message. This property can be both get and set. It is usually set when creating a new Log object. See [“Log class \(Flash Professional only\)” on page 1414.](#))

Example

The following example creates a new Log object with a `Log.level` property of `Log.VERBOSE` and a name of “myLog”. The current `Log.name` property is traced. Then the Log object’s `Log.name` property is set to “myNewLogName”.

```
import mx.services.*;
// Creates a new Log object.
myWebSvcLog = new Log(Log.VERBOSE, "myLog");
trace("myWebSvcLog.level: "+ myWebSvcLog.level);

// Sets a new name for the Log object.
myWebSvcLog.name = "myNewLogName";
trace("myWebSvcLog.name: " + myWebSvcLog.name);
```

Log.onLog()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebSvcLog.onLog = function(message)
```

Parameters

message The log message passed to the handler. For example:

```
“7/30 15:22:43 [INFO] SOAP: Decoding PendingCall response”
```

Returns

Nothing.

Description

Callback function; called by Flash Player when a log message is sent to a log file. This function is a good place to put code that records or displays the log messages, such as a `trace` command. (For information about the structure of the log, see [“Log class \(Flash Professional only\)” on page 1414.](#))

Example

The following example creates a new `Log` object, passes it to a new `WebService` object, and handles the log messages:

```
import mx.services.*;
// Creates a new Log object.
myWebSvcLog = new Log();
// Passes the Log object to the web service.
myWebService = new WebService(wsdlURI, myWebSvcLog);
// Handles incoming log messages.
myWebSvcLog.onLog = function(message : String) : Void
{
    mytrace("myWebSvcLog.message: " + message);
}
```

PendingCall class (Flash Professional only)

ActionScript Class Name `mx.services.PendingCall`

The `PendingCall` class is part of the `mx.services` package and is used with the `WebService` class. For an overview of the classes in the `mx.services` package, see [“Web service classes \(Flash Professional only\)” on page 1413](#).

You don't create a `PendingCall` object or use a constructor function; instead, when you call a method on a `WebService` object, the `WebService` method returns a `PendingCall` object. You use the `PendingCall.onResult` and `PendingCall.onFault` callback functions to handle the asynchronous response from the web service method. If the web service method returns a fault, Flash Player calls `PendingCall.onFault` and passes a `SOAPFault` object that represents the XML SOAP fault returned by the server or web service. A `SOAPFault` object is not constructed directly by you, but is returned as the result of a failure. This object is an ActionScript mapping of the `SOAPFault` XML type.

If the web service invocation is successful, Flash Player calls `PendingCall.onResult` and passes a result object. The result object is the XML response from the web service, decoded or deserialized into ActionScript. For more information about the `WebService` object, see [“WebService class \(Flash Professional only\)” on page 1437](#).

The `PendingCall` object also gives you access to multiple output parameters when the web service method returns more than one result. The “return value” referred to in this API is simply the first (or only) result; to gain access to all of the results, you can use the “get output” functions. For example, if the return value delivered to you in the parameter to the `onResult` callback is not the only result you want to access, you can use `getOutputValues()` (which returns an array) or `getOutputValue()` (which returns an individual value) to get the ActionScript decoded values.

You can also access the `SOAPParameter` object directly. The `SOAPParameter` object is an ActionScript object with two properties: `value` (the output parameter’s ActionScript value) and `element` (the output parameter’s XML value). The following functions return a `SOAPParameter` object, or an array of `SOAPParameter` objects: `getOutputParameters()`, `getOutputParameterByName()`, and `getOutputParameter()`.

Method summary for the `PendingCall` class

The following table lists methods of the `PendingCall` class.

Method	Description
<code>PendingCall.getOutputParameter()</code>	Retrieves a <code>SOAPParameter</code> object by index.
<code>PendingCall.getOutputParameterByName()</code>	Retrieves a <code>SOAPParameter</code> object by name.
<code>PendingCall.getOutputParameters()</code>	Retrieves an array of <code>SOAPParameter</code> objects.
<code>PendingCall.getOutputValue()</code>	Retrieves the output value according to the specified index.
<code>PendingCall.getOutputValues()</code>	Retrieves an array of all the output values.

Property summary for the `PendingCall` object

The following table lists properties of the `PendingCall` class.

Property	Description
<code>PendingCall.myCall</code>	The <code>SOAPCall</code> operation descriptor for the <code>PendingCall</code> operation.
<code>PendingCall.request</code>	The SOAP request in raw XML format.
<code>PendingCall.response</code>	The SOAP response in raw XML format.

Callback summary for the PendingCall object

The following table lists the callbacks of the PendingCall class.

Callback	Description
PendingCall.onFault	Called by Flash Player when a web service method has failed and returned an error.
PendingCall.onResult	Called when a method has succeeded and returned a result.

PendingCall.getOutputParameter()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCall.getOutputParameter(index)
```

Parameters

index The zero-based index of the parameter.

Returns

A SOAPParameter object with two properties: `value` (the output parameter's ActionScript value) and `element` (the output parameter's XML value).

Description

Function; gets an additional output parameter of the SOAPParameter object, which contains the value and the XML element. SOAP RPC calls may return multiple output parameters. The first (or only) return value is always delivered in the `result` parameter of the `onResult` callback function, but to get access to the other return values, you must use functions such as `getOutputParameter()` and `getOutputValue()`. The `getOutputParameter()` function returns the *n*th output parameter as a SOAPParameter object.

Example

Given the SOAP descriptor file below, `getOutputParameter(1)` would return a `SOAPParameter` object with `value="Hi there!"` and `element=the <outParam2> XMLNode.`

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">>true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

See also

[PendingCall.getOutputParameterByName\(\)](#), [PendingCall.getOutputParameters\(\)](#), [PendingCall.getOutputValue\(\)](#), [PendingCall.getOutputValues\(\)](#)

PendingCall.getOutputParameterByName()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCall.getOutputParameterByName(var localName)
```

Parameters

localName The local name of the parameter. In other words, the name of an XML element, stripped of any namespace information. For example, the local name of both of the following elements is `bob`:

```
<bob abc="123">
<xsd:bob def="ghi">
```

Returns

A `SOAPParameter` object with two properties: `value` (the output parameter's `ActionScript` value) and `element` (the output parameter's XML value).

Description

Function; gets any output parameter as a SOAPParameter object, which contains the value and the XML element. SOAP RPC calls may return multiple output parameters. The first (or only) return value is always delivered in the *result* parameter of the *onResult* callback function, but to get access to the other return values, you must use functions such as `getOutputParameterByName()`. This function returns the output parameter with the name *localName*.

Example

Given the SOAP descriptor file below, `getOutputParameterByName("outParam2")` would return a SOAPParameter object with `value="Hi there!"` and `element=the <outParam2> XMLNode`.

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">>true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

See also

[PendingCall.getOutputParameter\(\)](#), [PendingCall.getOutputParameters\(\)](#),
[PendingCall.getOutputValue\(\)](#), [PendingCall.getOutputValues\(\)](#)

PendingCall.getOutputParameters()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCall.getOutputParameters()
```

Parameters

None.

Returns

A SOAPParameter object with two properties: `value` (the output parameter's ActionScript value) and `element` (the output parameter's XML value).

Description

Function; gets additional output parameters of the SOAPParameter object, which contains the values and the XML elements. SOAP RPC calls may return multiple output parameters. The first (or only) return value is always delivered in the `result` parameter of the `onResult` callback function, but to get access to the other return values, you must use functions such as `getOutputParameters()` and `getOutputValues()`.

See also

[PendingCall.getOutputParameterByName\(\)](#), [PendingCall.getOutputParameter\(\)](#), [PendingCall.getOutputValue\(\)](#), [PendingCall.getOutputValues\(\)](#)

PendingCall.getOutputValue()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCall.getOutputValue(var index)
```

Parameters

index The index of an output parameter. The first parameter is index 0.

Returns

The *n*th output parameter.

Description

Function; gets the decoded ActionScript value of an individual output parameter. SOAP RPC calls may return multiple output parameters. The first (or only) return value is always delivered in the `result` parameter of the `onResult` callback function, but to get access to the other return values, you must use functions such as `getOutputValue()` and `getOutputParameter()`. The `getOutputValue()` function returns the *n*th output parameter.

Example

Given the SOAP descriptor file below, `getOutputValue(2)` would return `true`.

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

See also

[PendingCall.getOutputParameterByName\(\)](#), [PendingCall.getOutputParameter\(\)](#),
[PendingCall.getOutputParameters\(\)](#), [PendingCall.getOutputValues\(\)](#)

PendingCall.getOutputValues()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCall.getOutputValues()
```

Parameters

None.

Returns

An array of all output parameters' decoded values.

Description

Function; gets the decoded ActionScript value of all output parameters. SOAP RPC calls can return multiple output parameters. The first (or only) return value is always delivered in the *result* parameter of the *onResult* callback function, but to get access to the other return values, you must use functions such as `getOutputValues()` and `getOutputParameters()`.

See also

[PendingCall.getOutputParameterByName\(\)](#), [PendingCall.getOutputParameter\(\)](#), [PendingCall.getOutputParameters\(\)](#), [PendingCall.getOutputValue\(\)](#)

PendingCall.myCall

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

PendingCall.myCall

Description

Property; the SOAPCall object corresponding to the PendingCall operation. The SOAPCall object contains information about the web service operation, and provides control over certain behaviors. For more information, see [“SOAPCall class \(Flash Professional only\)” on page 1434](#).

Example

The following `onResult` callback traces the name of the SOAPCall operation.

```
callback.onResult = function(result)
{
    // Check my operation name.
    trace("My operation name is " + this.myCall.name);
}
```

PendingCall.onFault

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCallObj.onFault = function(fault)
{
    // Your code here.
}
```

Parameters

fault Decoded ActionScript object version of the SOAPFault object with properties. If the error information came from a server in the form of XML, the SOAPFault object is the decoded ActionScript version of that XML.

The type of error object returned to `PendingCall.onFault` is a SOAPFault object. It is not constructed directly by you, but is returned as the result of a failure. This object is an ActionScript mapping of the SOAPFault XML type.

SOAPFault property	Description
<code>faultcode</code>	String; a short string describing the error.
<code>faultstring</code>	String; the human-readable description of the error.
<code>detail</code>	String; the application-specific information associated with the error, such as a stack trace or other information returned by the web service engine.
<code>element</code>	XML; the XML object representing the XML version of the fault.
<code>faultactor</code>	String; the source of the fault (optional if an intermediary is not involved).

Returns

Nothing.

Description

Callback function; you provide this function, which Flash Player calls when a web service method has failed and returned an error. The *fault* parameter is an ActionScript SOAPFault object.

This is a good place to put code that handles any faults, for example, by telling the user that the server isn't available or to contact technical support, if appropriate.

Example

The following example handles errors returned from the web service method.

```
// Handles any error returned from the use of a web service method.
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onFault = function(fault)
{
    // Catches the SOAP fault.
    DebugOutputField.text = fault.faultstring;

    // Add code to handle any faults, for example, by telling the
    // user that the server isn't available or to contact technical
    // support.
}
}
```

PendingCall.onResult

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myPendingCallObj.onResult = function(result)
{
    // Your code here.
}
}
```

Parameters

result Decoded ActionScript object version of the XML result returned by a web service method called with `myPendingCallObj = myWebService.methodName(params)`.

Returns

Nothing.

Description

Callback function; you provide this function, which Flash Player calls when a web service method succeeds and returns a result. The result is a decoded ActionScript object version of the XML returned by the operation. In this function, include code that takes appropriate action based on the result. To return the raw XML instead of the decoded result, access the `PendingCall.response` property.

Example

The following example handles results returned from the web service method.

```
// Handles results returned from the use of a web service method.
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onResult = function(result)
{
    // Catches the result and handles it for this application.
    ResultOutputField.text = result;
}
```

See also

[PendingCall.response](#)

PendingCall.request

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
rawXML = myPendingCallback.request;
```

Description

Property; contains the raw XML form of the current request sent with `myPendingCallback = myWebService.methodName()`. Normally, you would not have any use for `PendingCall.request`, but you can use it if you are interested in the SOAP communications that are sent over the network. To get the ActionScript version of the results of the request, use `myPendingCallback.onResult`.

PendingCall.response

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
rawXML = myPendingCallback.response;
```

Description

Property; contains the raw XML form of the response to the most recent web service method call sent with `myPendingCallback = myWebService.methodName()`. Normally, you would not have any use for `PendingCall.response`, but you can use it if you are interested in the SOAP communications that are sent over the network. To get the corresponding ActionScript version of the results of the request, use `myPendingCallback.onResult`.

SOAPCall class (Flash Professional only)

ActionScript Class Name `mx.services.SOAPCall`

The `SOAPCall` class is part of the `mx.services` package and is an advanced class to be used with the `WebService` class (see [“WebService class \(Flash Professional only\)” on page 1437](#)). For an overview of the classes in the `mx.data.services` package, see [“Web service classes \(Flash Professional only\)” on page 1413](#).

The `SOAPCall` object is not constructed by you. Instead, when you call a method on a `WebService` object, the `WebService` object returns a `PendingCall` object. To access the associated `SOAPCall` object, use `myPendingCall.myCall`.

When you create a new `WebService` object, it contains the methods that correspond to operations in the WSDL URL you pass in. Behind the scenes, a `SOAPCall` object is created for each operation in the WSDL as well. The `SOAPCall` object is the descriptor of the operation, and as such contains all the information about that operation (how the XML should look on the network, the operation style, and so on). It also provides control over certain behaviors. You can get the `SOAPCall` object for a given operation by using the `WebService.getCall()` function. There is a single `SOAPCall` for each operation, shared by all active calls to that operation. Once you have the `SOAPCall` object, you can customize the descriptor by doing the following:

- Turning on/off decoding of the XML response
- Turning on/off the delay of converting SOAP arrays into ActionScript objects
- Changing the concurrency configuration for a given operation

Property summary for the SOAPCall object

The following table lists the properties of the SOAPCall object.

Property	Description
SOAPCall.concurrency	The number of concurrent requests.
SOAPCall.doDecoding	Turns the decoding of the XML response on or off.
SOAPCall.doLazyDecoding	Turns “lazy decoding” (the delay of turning SOAP arrays into ActionScript objects) on or off.

SOAPCall.concurrency

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`SOAPCall.concurrency`

Description

Property; the number of concurrent requests. Possible values are listed in the table below:

Value	Description
<code>SOAPCall.Multiple_Concurrency</code>	Allows multiple active calls.
<code>SOAPCall.Single_Concurrency</code>	Allows only one call at a time by causing a fault after one is active.
<code>SOAPCall.Last_Concurrency</code>	Allows only one call by cancelling previous ones.

SOAPCall.doDecoding

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`SOAPCall.doDecoding`

Description

Property; turns decoding of the XML response on (`true`) or off (`false`). By default, the XML response is converted (decoded) into ActionScript objects. If you want just the XML, set `SOAPCall.doDecoding` to `false`.

SOAPCall.doLazyDecoding

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`SOAPCall.doLazyDecoding`

Description

Property; turns “lazy decoding” of arrays on (`true`) or off (`false`). By default, a “lazy decoding” algorithm is used to delay turning SOAP arrays into ActionScript objects until the last moment; this makes functions execute much more quickly when returning large data sets. This means any arrays you receive from the remote location are `ArrayProxy` objects. Then when you access a particular index (`foo[5]`), that element is automatically decoded if necessary. You can turn this behavior off (which causes all arrays to be fully decoded) by setting `SOAPCall.doLazyDecoding` to `false`.

WebService class (Flash Professional only)

ActionScript Class Name mx.services.WebService

The WebService class is part of the mx.services package and is used with the Log, PendingCall, and SOAPCall classes. For an overview of the classes in the mx.services package, see [“Web service classes \(Flash Professional only\)” on page 1413](#).

NOTE

The WebService class is not the same as the WebServiceConnector class. The WebServiceConnector class provides an ActionScript interface to the visual WebServiceConnector component.

The WebService object acts as a local reference to a remote web service. When you create a new WebService object, the WSDL file that defines the web service gets downloaded, parsed, and placed in the object. You can then call the methods of the web service directly on the WebService object and handle any callbacks from the web service. When the WSDL has been successfully processed and the WebService object is ready, the `WebService.onLoad` callback is invoked. If there is a problem loading the WSDL, the `WebService.onFault` callback is invoked.

When you call a method on a WebService object, the return value is a callback object. The object type of the callback returned from all web service methods is `PendingCall`. These objects are normally not constructed by you, but instead are constructed automatically as a result of the `WebServiceObject.webServiceMethodName()` method that was called. These objects are not the result of the WebService call, which occurs later. Instead, the `PendingCall` object represents the call in progress. When the WebService operation finishes executing (usually several seconds after a method is called), the various `PendingCall` data fields are filled in, and the `PendingCall.onResult` or `PendingCall.onFault` callback you provide is called. For more information about the `PendingCall` object, see [“PendingCall class \(Flash Professional only\)” on page 1423](#).

Flash Player queues any calls you make before the WSDL is parsed, and attempts to execute them after parsing the WSDL. This is because the WSDL contains information that is necessary for correctly encoding and sending a SOAP request. Function calls that you make after the WSDL has been parsed do not need to be queued; they are executed immediately. If a queued call doesn't match the name of any of the operations defined in the WSDL, Flash Player returns a fault to the callback object you were given when you originally made the call. The WebServices API, included under the mx.services package, consists of the WebService class, the Log class, the PendingCall class, and the PendingCall class.

To make the web service classes available at runtime, you must have the `WebServiceConnector` component in your FLA file's library. If you're using ActionScript only to access a web service at runtime, you must add this component manually to your document's library. For information on how to add this component to your document, see Chapter 16, "Data Integration (Flash Professional Only)," in *Using Flash*.

Method summary for the `WebService` object

The following table lists methods of the `WebService` object.

Method	Description
WebService.call()	Gets the <code>SOAPCall</code> object for a given operation.
WebService.myMethodName()	Invokes a specific web service operation defined by the WSDL.

Callback summary for the `WebService` object

The following table lists the callbacks of the `WebService` object.

Callback	Description
WebService.onFault	Called when an error occurs during WSDL parsing.
WebService.onLoad	Called when the web service has successfully loaded and parsed its WSDL file.

Supported types (Flash Professional only)

The web services classes support a subset of XML schema types (data types) as defined in the tables below.

Complex data types and the SOAP-encoded array type are also supported, and these may be composed of other complex types, arrays, or built-in XML schema types:

- "Numeric Simple types" on page 1439
- "Date and Time Simple types" on page 1439
- "Name and String Simple types" on page 1440
- "Boolean type" on page 1440
- "Object types" on page 1440
- "Supported XML schema object elements" on page 1440

Numeric Simple types

XML schema type	ActionScript binding
decimal	Number
integer	Number
negativeInteger	Number
nonNegativeInteger	Number
positiveInteger	Number
long	Number
int	Number
short	Number
byte	Number
unsignedLong	Number
unsignedShort	Number
unsignedInt	Number
unsignedByte	Number
float	Number
double	Number

Date and Time Simple types

XML schema type	ActionScript binding
date	Date object
datetime	Date object
duration	Date object
gDay	Date object
gMonth	Date object
gMonthDay	Date object
gYear	Date object
gYearMonth	Date object
time	Date object

Name and String Simple types

XML schema type	ActionScript binding
string	ActionScript string
normalizedString	ActionScript string
QName	mx.services.Qname object

Boolean type

XML schema type	ActionScript binding
Boolean	Boolean

Object types

XML schema type	ActionScript binding
Any	XML object
Complex Type	ActionScript object composed of properties of any supported type
Array	ActionScript array composed of any supported object or type

Supported XML schema object elements

The following schema description illustrates the supported XML schema object elements:

```
schema
  complexType
    complexContent
      restriction
    sequence | simpleContent
      restriction
  element
    complexType | simpleType
```

WebService security (Flash Professional only)

The methods and callbacks of the `WebService` class conform to the Flash Player security model. For more information on the Flash Player security model, see “Understanding Security” in *Learning ActionScript 2.0 in Flash*.

User authentication and authorization The authentication and authorization rules are the same for the `WebService` API as they are for any XML network operation from Flash. SOAP itself does not specify any means of authentication and authorization. For example, when the underlying HTTP transport returns an HTTP BASIC response in the HTTP headers, the browser responds by presenting a dialog box and subsequently attaching the user’s input to the HTTP headers in subsequent messages. This mechanism exists at a level lower than SOAP and is part of the Flash HTTP authentication design.

Message integrity Message-level security involves the encryption of the SOAP messages themselves, at a conceptual layer above the network packets on which the SOAP messages are delivered.

Transport security The underlying network transport for Flash Player SOAP web services is always HTTP `POST`. Therefore, any means of security that can be applied at the Flash HTTP transport layer—such as SSL—is supported through web services invocations from Flash. SSL/HTTPS provides the most common form of transport security for SOAP messaging, and use of HTTP BASIC authentication, coupled with SSL at the transport layer, is the most common form of security for websites today.

Constructor for the `WebService` class

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myWebServiceObject = new WebService(wsdlURI [, logObject]);
```

Parameters

wsdURI The URI of the web service WSDL file.

logObject An optional parameter specifying the name of the Log object for this web service. If this parameter is used, the Log object must be created first. For more information, see [“Log class \(Flash Professional only\)” on page 1414](#).

Returns

Nothing.

Description

Constructor function; creates a `WebService` object. When you call `new WebService()`, you provide a WSDL URL, and Flash Player returns a `WebService` object. The constructor can optionally accept a proxy URI and a Log object.

If you want, you can use two callbacks for the `WebService` object. Flash Player calls `webServiceObject.onLoad` when it finishes parsing the WSDL file and the object is complete. This is a good place to put code you want to execute only after the WSDL file has been completely parsed. For example, you might choose to put your first web service method call in this function.

Flash Player calls `webServiceObject.onFault` when an error occurs in finding or parsing the WSDL file. This is a good place to put debugging code and code that tells users that the server is unavailable, that they should try again later, or similar information. For more information, see the individual entries for these functions.

Invoking a web service operation You invoke a web service operation as a method directly available on the web service. For example, if your web service has the method `getCompanyInfo(tickerSymbol)`, you would invoke the method in the following manner:
`myPendingCallObject = myWebServiceObject.getCompanyInfo(tickerSymbol);`

In this example, the callback object is named `myPendingCallObject`. All method invocations are asynchronous, and return a callback object of type `PendingCall`. (*Asynchronous* means that the results of the web service call are not available immediately.)

Consider the following call:

```
x = stockService.getQuote("macr");
```

When you make this call, the object `x` is not the result of `getQuote`; it's a `PendingCall` object. The actual results are only available later, when the web service operation completes. Your ActionScript code is notified by a call to the `onResult` callback function.

Handling the PendingCall object This callback object is a PendingCall object that you use for handling the results and errors from the web service method that was called (see [“PendingCall class \(Flash Professional only\)” on page 1423](#)). Here is an example:

```
MyPendingCallObject = myWebServiceObject.myMethodName(param1, ..., paramN);
MyPendingCallObject.onResult = function(result)
{
    OutputField.text = result
}
MyPendingCallObject.onFault = function(fault)
{
    DebugField.text = fault.faultCode + "," + fault.faultstring;

    // Add code to handle any faults, for example, by telling the
    // user that the server isn't available or to contact technical
    // support.
}
```

WebService.getCall()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
getCall(var operationName)
```

Parameters

operationName The web service operation of the corresponding SOAPCall object that you want to retrieve.

Returns

A SOAPCall object.

Description

Function; when you create a new `WebService` object, it contains the methods corresponding to operations in the WSDL URL you pass in. Behind the scenes, a `SOAPCall` object is created for each operation in the WSDL as well. The `SOAPCall` object is the descriptor of the operation, and as such contains all the information about that operation (how the XML should look on the network, the operation style, and so on). It also provides control over certain behaviors. You can get the `SOAPCall` object for a given operation by using the `getCall()` method. There is a single `SOAPCall` object for each operation, shared by all active calls to that operation. Once you have the `SOAPCall` object, you can change the operator descriptor by using the `SOAPCall` class; see [“SOAPCall class \(Flash Professional only\)” on page 1434](#).

WebService.myMethodName()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
callbackObj = myWebServiceObject.myMethodName(param1, ... paramN);
```

Parameters

param1, ... *paramN* Various parameters, depending on the web service method that is called.

Returns

A `PendingCall` object to which you can attach a function for handling results and errors on the invocation. For more information, see [“PendingCall class \(Flash Professional only\)” on page 1423](#).

The callback invoked when the response comes back from the `WebService` method is `PendingCall.onResult` or `PendingCall.onFault`. By uniquely identifying your callback objects, you can manage multiple `onResult` callbacks, as in the following example:

```
myWebService = new WebService("http://www.myCompany.com/myService.wsdl");
callback1 = myWebService.getWeather("02451");
callback1.onResult = function(result)
{
    // do something
}
callback2 = myWebService.getDetailedWeather("02451");
callback2.onResult = function(result)
{
    // do something else
}
```

Description

Method; invokes a web service operation. You invoke the method directly on the web service. For example, if your web service has the method `getCompanyInfo(tickerSymbol)`, you would make the following call:

```
myCallbackObject.myservice.getCompanyInfo(tickerSymbol);
```

All invocations are asynchronous, and return a callback object of type `PendingCall`.

WebService.onFault

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
MyWebServiceObject.onFault = function(fault)
{
    // Your code here.
}
```

Parameters

fault Decoded ActionScript object version of the error with properties. If the error information came from a server in the form of XML, then the `SOAPFault` object is the decoded ActionScript version of that XML.

The type of error object returned to `WebService.onFault` methods is a `SOAPFault` object. This object is not constructed directly by you, but returned as the result of a failure. This object is an ActionScript mapping of the `SOAPFault` XML type.

SOAPFault property	Description
<code>faultcode</code>	String; the short standard QName describing the error.
<code>faultstring</code>	String; the human-readable description of the error.
<code>detail</code>	String; the application-specific information associated with the error, such as a stack trace or other information returned by the web service engine.
<code>element</code>	XML; the XML object representing the XML version of the fault.
<code>faultactor</code>	String; the source of the fault. Optional if an intermediary is not involved.

Returns

Nothing.

Description

Callback function; called by Flash Player when the new `WebService()` constructor has failed and returned an error. This can happen when the WSDL file cannot be parsed or the file cannot be found. The `fault` parameter is an ActionScript `SOAPFault` object.

Example

The following example handles any error returned from the creation of the `WebService` object.

```
MyWebServiceObject.onFault = function(fault)
{
    // Captures the fault.
    DebugOutputField.text = fault.faultstring;

    // Adds code to handle any faults, for example, by telling the
    // user that the server isn't available or to contact technical
    // support.
}
```

WebService.onLoad

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myService.onLoad = function(wsd1Document)
{
    // Your code here.
}
```

Parameters

wsd1Document The WSDL XML document.

Returns

Nothing.

Description

Callback function; called by Flash Player when the WebService object has successfully loaded and parsed its WSDL file. If operations are invoked in an application before this callback function is called, they are queued internally and not actually transmitted until the WSDL has loaded.

Example

The following example specifies the WSDL URL, creates a new web service object, and receives the WSDL document after loading.

```
// Specify the WSDL URL.
var wsd1URI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";

// Creates a new web service object.
stockService = new WebService(wsd1URI);

// Receives the WSDL document after loading.
stockService.onLoad = function(wsd1Document);
{
    // Code to execute when the WSDL loading is complete and the
    // object has been created.
}
```


WebServiceConnector component (Flash Professional only)

The `WebServiceConnector` component lets you access remote methods exposed by a server using the industry-standard Simple Object Access Protocol (SOAP). A web service method can accept parameters and return a result. Using the Flash Professional 8 authoring tool and the `WebServiceConnector` component, you can inspect, access, and bind data between a remote web service and your Flash application.

A single instance of a `WebServiceConnector` component can be used to make multiple calls to the same operation. You need to use a different instance of `WebServiceConnector` for each different operation you want to call.

NOTE

The `WebServiceConnector` component appears on the Stage during application authoring but is not visible in the runtime application.

For introductory information on working with the results of this component, see “Working with schemas in the Schema tab (Flash Professional only)” in *Using Flash*.

Using the WebServiceConnector component (Flash Professional only)

You can use the `WebServiceConnector` component to connect to a web service and make the properties of the web service available for binding to properties of UI components in your application. To connect to a web service, you must first enter the URL for the WSDL file that represents the web service. You can enter this URL in the Component inspector or the Web Services panel. See “Connecting to web services with the `WebServiceConnector` component (Flash Professional only)” in *Using Flash*.

For more information on connecting to web services, see “Data binding (Flash Professional only)” in *Using Flash*.

WebServiceConnector parameters

You can set the following authoring parameters for each `WebServiceConnector` component instance by using the Parameters tab of the Component inspector:

multipleSimultaneousAllowed is a Boolean value that indicates whether multiple calls can take place at the same time; the default value is `false`. If this parameter is `false`, the `trigger()` method does not perform a call if a call is already in progress. A status event is emitted, with the code `CallAlreadyInProgress`. If this parameter is `true`, the call takes place.

operation is a string indicating the name of an operation that appears within the SOAP port in a WSDL file.

suppressInvalidCalls is a Boolean value that indicates whether to suppress a call if parameters are invalid; the default value is `false`. If this parameter is `true`, the `trigger()` method does not perform a call if the databound parameters fail the validation. A status event is emitted, with the code `InvalidParams`. If this parameter is `false`, the call takes place, using the invalid data as required.

WSDLURL (String type) is the URL of the WSDL file that defines the web service operation. When you set this URL during authoring, the WSDL file is immediately fetched and parsed. The resulting parameters and results information can be seen in the Schema tab of the Component inspector. The service description is also added to the Web Service panel. For example, see www.flash-mx.com/mm/tips/tips.cfc?wsdl.

Common workflow for the WebServiceConnector component

The following procedure shows the typical workflow for the `WebServiceConnector` component.

To use a `WebServiceConnector` component:

1. Use the Web Services panel to enter the URL for a web service WSDL file.
2. Add a call to a method of the web service by selecting the method, right-clicking (Windows) or Control-clicking (Macintosh), and selecting Add Method Call from the context menu.

This creates a `WebServiceConnector` component instance in your application. The schema for the component can then be found on the Schema tab of the Component inspector. You are free to edit this schema as needed—for example, to provide additional formatting or validation settings.

NOTE

The schema for the `params` and `results` component properties is updated each time you change the `WSDLURL` or `operation` parameter. This overwrites any settings that you have edited.

3. Use the Bindings tab in the Component inspector to bind the web service parameters and results that are now defined in your schema to UI components in your application.
4. Add a trigger to initiate the data binding operation in one of the following ways:
 - Attach the Trigger Data Source behavior to a button.
 - Add your own `ActionScript` to call the `trigger()` method on the `WebServiceConnector` component.
 - Create a binding between a web service parameter and a UI control, and set its `Kind` property to `AutoTrigger`. For more information, see “Schema kinds” in *Using Flash*.

For a step-by-step example that connects and displays a web service using the `WebServiceConnector` component, see “Web Service Tutorial: Macromedia Tips.”

WebServiceConnector class (Flash Professional only)

Inheritance `RPC > WebServiceConnector`

ActionScript Class Name `mx.data.components.WebServiceConnector`

This class allows you to connect to remote web services using `ActionScript` code instead of component instances on the Stage. To use the `WebServiceConnector` class, you need to add an instance of the `WebServiceConnector` component to your library. The component does not need to be placed directly on the Stage. You must import the `ActionScript` class `mx.data.components.WebServiceConnector` at the beginning of the script or use the fully qualified class name throughout your code.

Method summary for the `WebServiceConnector` class

The following table lists the method of the `WebServiceConnector` class.

Method	Description
<code>WebServiceConnector.trigger()</code>	Initiates a call to a web service.

Property summary for the `WebServiceConnector` class

The following table lists properties of the `WebServiceConnector` class.

Property	Description
<code>WebServiceConnector.multipleSimultaneousAllowed</code>	Indicates whether multiple calls can take place at the same time.
<code>WebServiceConnector.operation</code>	Indicates the name of an operation that appears within the SOAP port in a WSDL file.
<code>WebServiceConnector.params</code>	Specifies data that is sent to the server when the next <code>trigger()</code> operation is executed.
<code>WebServiceConnector.results</code>	Identifies data that was received from the server as a result of the <code>trigger()</code> operation.
<code>WebServiceConnector.suppressInvalidCalls</code>	Indicates whether to suppress a call if parameters are invalid.
<code>WebServiceConnector.WSDLURL</code>	Specifies the URL of the WSDL file that defines the web service operation.

Event summary for the `WebServiceConnector` class

The following table lists events of the `WebServiceConnector` class.

Event	Description
<code>WebServiceConnector.result</code>	Broadcast when a call to a web service completes successfully.
<code>WebServiceConnector.send</code>	Broadcast when the <code>trigger()</code> method is in process, after the parameter data has been gathered but before the data is validated and the call to the web service is initiated.
<code>WebServiceConnector.status</code>	Broadcast when a call to a web service is initiated, to inform the user of the status of the operation.

`WebServiceConnector.multipleSimultaneousAllowed`

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`componentInstance.multipleSimultaneousAllowed`

Description

Property; indicates whether multiple calls can (`true`) or cannot (`false`) take place at the same time. If this property is `true`, the call takes place. If this property is `false`, and another call is already in progress, the `WebServiceConnector.trigger()` method causes a `status` event to be emitted with the code `CallAlreadyInProgress`.

When multiple calls are simultaneously in progress, there is no guarantee that they will be completed in the order in which they were triggered. Also, the browser and/or operating system may place limits on the number of simultaneous network operations. The most likely limit you may encounter is the browser enforcing a maximum number of URLs that can be downloaded simultaneously. This is something that is often configurable in a browser. However, even in this case, the browser should queue streams, and this should not interfere with the expected behavior of the Flash application.

Example

The following example enables multiple simultaneous calls to `myXmlUrl` to take place:

```
myXmlUrl.multipleSimultaneousAllowed = true;
```

WebServiceConnector.operation

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.operation;
```

Description

Property; the name of an operation that appears within the SOAP port in a WSDL file.

Example

This example returns data from a remote web service and traces a tip and how long the service takes to return the data to the SWF file. Drag a `WebServiceConnector` component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.WebServiceConnector;

var startTime:Number;
var wsListener:Object = new Object();
wsListener.result = function(evt:Object) {
    var resultTimeMS:Number = getTimer()-startTime;
    trace("result loaded in "+resultTimeMS+" ms.");
    trace(evt.target.results);
};
wsListener.send = function(evt:Object) {
    startTime = getTimer();
};
```

```
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", wscListener);
wsConn.addEventListener("send", wscListener);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.multipleSimultaneousAllowed = false;
wsConn.trigger();
```

WebServiceConnector.params

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.params

Description

Property; specifies data that will be sent to the server when the next `trigger()` operation is executed. The data type is determined by the WSDL description of the web service.

When you call web service methods, the data type of the `params` property must be an ActionScript object or array as follows:

- If the web service is in document format, the data type of `params` is an XML document.
- If you use the Property inspector or Component inspector to set the WSDL URL and operation while authoring, you can provide `params` as an array of parameters in the same order as required by the web service method, such as `[1, "hello", 2432]`.

Example

The following example sets the `params` property for a web service component named `wsc`:

```
wsc.params = [param_txt.text];
```

WebServiceConnector.result

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.addEventListener("result", myListenerObject)

Description

Event; broadcast when a call to a web service completes successfully.

The parameter to the event handler is an object with the following fields:

- **type:** the string "result"
- **target:** a reference to the object that emitted the event (for example, a WebServiceConnector component)

You can retrieve the actual result value using the `results` property.

Example

The following example defines a function `res` for the `result` event and assigns the function to the `addEventListener` event handler:

```
var res = function (ev) {
    trace(ev.target.results);
};
wsc.addEventListener("result", res);
```

This example returns data from a remote web service and traces a tip. Drag a WebServiceConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.WebServiceConnector;
var wscListener:Object = new Object();
wscListener.result = function(evt:Object) {
    trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", wscListener);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.trigger();
```

WebServiceConnector.results

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.results

Description

Property; identifies data that was received from the server as a result of a `trigger()` operation. Each `WebServiceConnector` component defines how this data is fetched, and what the valid types are. This data appears when the RPC operation has successfully completed, as signaled by the `result` event. It is available until the component is unloaded, or until the next RPC operation.

The returned data can be large. You can manage this size in two ways:

- Select an appropriate movie clip, timeline, or screen as the parent for the `WebServiceConnector` component. The component's storage memory becomes available for garbage collection when the parent is destroyed.
- In ActionScript, you can assign `null` to this property at any time.

WebServiceConnector.send

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.addEventListener("send", myListenerObject)

Description

Event; broadcast during the processing of a `trigger()` operation, after the parameter data has been gathered but before the data is validated and the call to the web service is initiated. This is a good place to put code that modifies the parameter data before the call.

The parameter to the event handler is an object with the following fields:

- `type`: the string "send"
- `target`: a reference to the object that emitted the event (for example, a `WebServiceConnector` component)

You can retrieve or modify the actual parameter values by using the `params` property.

Example

The following example defines a function `sendFunction` for the `send` event and assigns the function to the `addEventListener` event handler:

```
var sendFunction = function (sendEnv) {
    sendEnv.target.params = [newParam_txt.text];
};
wsc.addEventListener("send", sendFunction);
```

WebServiceConnector.status

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.addEventListener("status", myListenerObject)
```

Description

Event; broadcast when a call to a web service is initiated, to inform the user of the status of the operation.

The parameter to the event handler is an object with the following fields:

- `type`: the string "status"
- `target`: a reference to the object that emitted the event (for example, a `WebServiceConnector` component)
- `code`: a string giving the name of the condition that occurred
- `data`: an object whose contents depend on the code

The following are the codes and associated data available for the `status` event:

Code	Data	Description
StatusChange	{callsInProgress:nnn}	This event is emitted whenever a web service call starts or finishes. The item <code>nnn</code> gives the number of calls currently in progress.
CallAlreadyInProgress	No data	This event is emitted if <code>trigger()</code> is called, <code>multipleSimultaneousAllowed</code> is <code>false</code> , and a call is already in progress. After this event occurs, the attempted call is considered complete, and there is no <code>result</code> or <code>send</code> event.
InvalidParams	No data	This event is emitted if the <code>trigger()</code> method found that the <code>params</code> property did not contain valid data. If the <code>suppressInvalidCalls</code> property is <code>true</code> , the attempted call is considered complete, and there is no <code>result</code> or <code>send</code> event.
WebServiceFault	{faultcode: code, faultstring: string, detail: detail}	This event is emitted if other problems occur during the processing of the call. The data is a <code>SOAPFault</code> object. After this event occurs, the attempted call is considered complete, and there is no "result" or "send" event. See the following table for a list of the faults that can occur.

Here are the possible web service faults:

faultcode	faultstring	detail
Timeout	Timeout while calling method xxx	
MustUnderstand	No callback for header xxx	
Server.Connection	Unable to connect to endpoint: xxx	
VersionMismatch	Request implements version: xxx Response implements version yyy	

faultcode	faultstring	detail
Client.Disconnected	Could not load WSDL	Unable to load WSDL, if currently online, please verify the URI and/or format of the WSDL xxx
Server	Faulty WSDL format	Definitions must be the first element in a WSDL document
Server.NoServicesInWSDL	Could not load WSDL	No elements found in WSDL at xxx
WSDL.UnrecognizedNamespace	The WSDL parser had no registered document for the namespace xxxx	
WSDL.UnrecognizedBindingName	The WSDL parser couldn't find a binding named xxx in namespace yyy	
WSDL.UnrecognizedPortTypeName	The WSDL parser couldn't find a portType named xxx in namespace yyy	
WSDL.UnrecognizedMessageName	The WSDL parser couldn't find a message named xxx in namespace yyy	
WSDL.BadElement	Element xxx not resolvable	
WSDL.BadType	Type xxx not resolvable	
Client.NoSuchMethod	Couldn't find method 'xxx' in service	
yyy	yyy - errors reported from server, this depends on which server you talk to	
No.WSDLURL.Defined	The WebServiceConnector component had no WSDL URL defined	

faultcode	faultstring	detail
Unknown.Call.Failure	WebService invocation failed for unknown reasons	
Client.Disconnected	Could not load imported schema	Unable to load schema; if currently online, please verify the URI and/or format of the schema at (XXXX)

Example

The following example defines a function `fault` for the `status` event and assigns the function to the `addEventListener` event handler. The example intentionally misspells the URI for the service to return a web service fault (the url should be "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl") and a message asking the user to verify the URI. With a `WebServiceConnector` component in the library, add the following to the first frame of the timeline:

```
import mx.data.components.WebServiceConnector;

var fault = function (stat) {
    if (stat.code == "WebServiceFault"){
        trace(stat.data.faultcode);
        trace(stat.data.faultstring);
        trace(stat.data.detail);
    }
};

var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("status", fault);
wsConn.WSDLURL = "http://www.flasht-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.trigger();
```

WebServiceConnector.suppressInvalidCalls

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.suppressInvalidCalls

Description

Property; indicates whether to suppress a call if parameters are invalid. If this property is true, the `trigger()` method does not perform a call if the bound parameters fail the validation. A status event is emitted, with the code `InvalidParams`. If this property is false, the call takes place, using the invalid data as required.

Example

This example displays an error because the required parameters are not being passed. Drag a `WebServiceConnector` component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
    trace(evt.target.results);
};
var stat:Function = function (error:Object) {
    switch (error.code) {
        case 'InvalidParams' :
            trace("Unable to connect to remote Web Service: "+error.code);
            break;
        case 'StatusChange' :
            break;
        default :
            trace("Error: "+error.code);
            break;
    }
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.addEventListener("status", stat);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
// wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```

To display a tip instead of an error, uncomment the line `wsConn.params = ["Flash"];`.

WebServiceConnector.trigger()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.trigger();
```

Description

Method; initiates a call to a web service. Each web service defines exactly what this involves. If the operation is successful, the results of the operation appear in the `results` property for the web service.

The `trigger()` method performs the following steps:

1. If any data is bound to the `params` property, the method executes all the bindings to ensure that up-to-date data is available. This also causes data validation to occur.
2. If the data is not valid and `suppressInvalidCalls` is set to `true`, the operation is discontinued.
3. If the operation continues, the `send` event is emitted.
4. The actual remote call is initiated using the connection method indicated (for example, HTTP).

Example

This example returns data from a remote web service and traces a tip. Drag a `WebServiceConnector` component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
    trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```

WebServiceConnector.WSDLURL

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.WSDLURL

Description

Property; the URL of the WSDL file that defines the web service operation. When you set this URL while authoring, the WSDL file is immediately fetched and parsed. The resulting parameters and results appear in the Schema tab of the Component inspector. The service description also appears in the Web Service panel.

Example

This example returns data from a remote web service and traces a tip. Drag a WebServiceConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
    trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```

A Window component displays the contents of a movie clip inside a window with a title bar, a border, and an optional close button.

A Window component can be modal or nonmodal. A modal window prevents mouse and keyboard input from going to other components outside the window. The Window component also supports dragging; a user can click the title bar and drag the window and its contents to another location. Dragging the borders doesn't resize the window.

A live preview of each Window instance reflects changes made to all parameters except `contentPath` in the Property inspector or Component inspector during authoring.

When you add the Window component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.WindowAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component. For more information, see Chapter 19, “Creating Accessible Content,” in *Using Flash*.

Using the Window component

You can use a window in an application whenever you need to present a user with information or a choice that takes precedence over anything else in the application. For example, you might need a user to fill out a login window, or a window that changes and confirms a new password.

There are several ways to add a window to an application. You can drag a window from the Components panel to the Stage. You can also call `createClassObject()` (see [UIObject.createClassObject\(\)](#)) to add a window to an application. The third way of adding a window to an application is to use the [PopUpManager class](#). Use the PopUp Manager to create modal windows that overlap other objects on the Stage. For more information, see “Window class” on page 1472.

If you use the Popup Manager to add a Window component to a document, the Window instance will have its own Focus Manager, distinct from the rest of the document. If you don't use the Popup Manager, the window's contents participate in focus ordering with the other components in the document. For more information about controlling focus, see [“FocusManager class” on page 721](#) or [“Creating custom focus navigation” in *Using Components*](#).

Components such as Loader, ScrollPane, and Window have events to determine when content finishes loading. To set properties on the content of a Loader, ScrollPane, or Window, add the property statement within a “complete” event handler, as shown in the following example:

```
loadtest = new Object();
loadtest.complete = function(eventObject){
    content_mc._width= 100;
}
my_window.addEventListener("complete", loadtest)
```

For more information, see [“Window.complete” on page 1479](#).

Window parameters

You can set the following authoring parameters for each Window component instance in the Property inspector or in the Component inspector (Window > Component Inspector menu option):

closeButton indicates whether a close button is displayed (`true`) or not (`false`). Clicking the close button broadcasts a `click` event, but doesn't close the window. You must write a handler that calls `Window.deletePopUp()` to explicitly close the window. For more information about the `click` event, see [Window.click](#).

NOTE

If a window was created by means other than PopUp Manager, you can't close it.

contentPath specifies the contents of the window. This can be the linkage identifier of the movie clip or the symbol name of a screen, form, or slide that contains the contents of the window. This can also be an absolute or relative URL for a SWF or JPEG file to load into the window. The default value is `" "`. Loaded content is clipped to fit the window.

title indicates the title of the window.

NOTE

The `minHeight` and `minWidth` properties are used by internal sizing routines. They are defined in `UIObject`, and are overridden by different components as needed. These properties can be used if you make a custom layout manager for your application. Otherwise, setting these properties in the Component inspector has no visible effect.

You can set the following additional parameters for each Window component instance in the Component inspector (Window > Component Inspector):

enabled is a Boolean value that indicates whether the component can receive focus and input. The default value is `true`.

visible is a Boolean value that indicates whether the object is visible (`true`) or not (`false`). The default value is `true`.

NOTE

For more information about the following five skin parameters, see [“Using skins with the Window component” on page 1470](#).

skinCloseDisabled determines the close button in its disabled state. The default value is `CloseButtonDisabled`.

skinCloseDown determines the close button in its down state. The default value is `CloseButtonDown`.

skinCloseOver determines the close button in its over state. The default value is `CloseButtonOver`.

skinCloseUp determines the close button in its up (default) state. The default value is `CloseButtonUp`.

skinTitleBackground determines the title bar appearance. The default value is `TitleBackground`.

titleStyleDeclaration assigns the name of the style declaration for the title text. The default value is undefined, which causes the title bar to have white, bold text. See “Setting custom styles for groups of components” in *Using Components*.

You can write ActionScript to control these and additional options for the Window component using its properties, methods, and events. For more information, see [“Window class” on page 1472](#).

Creating an application with the Window component

The following procedure explains how to add a Window component to an application. In this example, when the user clicks a button the window displays an image.

To create an application with the Window component:

1. Drag a Window component from the Components panel to the current document’s library. This adds the component to the library but not to the Stage.
2. Drag a button component from the Components panel to the Stage; in the Property inspector, give it the instance name **my_button**.

3. Open the Actions panel, and enter the following click handler in Frame 1:

```
/**
 * Requires:
 *   - Button component on Stage (instance name: my_button)
 *   - Window component in library
 */
import mx.containers.Window;

var my_button:mx.controls.Button;

System.security.allowDomain("http://www.helpexamples.com");

// Create listener object.
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
    // Instantiate Window.
    var my_win:MovieClip =
        mx.managers.PopUpManager.createPopUp(evt_obj.target, Window, true,
        {title:"Sample Image", contentPath:"http://www.helpexamples.com/
        flash/images/image1.jpg"});
    my_win.setSize(320, 240);
};
// Add listener.
my_button.addEventListener("click", buttonListener);
```

This example creates a `click()` function that the `buttonListener` event listener calls when the user clicks the button `my_button`. The click event handler, `buttonListener.click()`, calls `PopUpManager.createPopUp()` to instantiate a window that displays an image. To close the window when the OK or Cancel button is clicked, you would need to write another handler.

Customizing the Window component

You can transform a Window component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use `UIObject.setSize()`.

Resizing the window does not change the size of the close button or title caption. The title caption is aligned to the left and the close bar to the right.

Using styles with the Window component

A Window component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color. The default value is white for the Halo theme and OxEFEBEF (light gray) for the Sample theme.
<code>borderStyle</code>	Both	The Window component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” on page 1063 . The Window component has a component-specific border style of “default” with the Halo theme and “outset” with the Sample theme.
<code>color</code>	Both	The text color. The default value is OxOB333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is Ox848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font is not used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text is displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".

Style	Theme	Description
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.

Text styles can be set on the Window component itself, or they can be set on the `_global.styles.windowStyles` class style declaration (text styles, only, not other styles like `themeColor` or `backgroundColor`, which come from the `_global.styles.Window` class style declaration). This has the advantage of not causing style settings to propagate to child components through style inheritance.

The following example demonstrates how to italicize the title of a Window component without having this setting propagate to child components.

```
import mx.containers.Window;
_global.styles.windowStyles.setStyle("fontStyle", "italic");
createClassObject(Window, "window", 1, {title: "A Window"});
```

Notice that this example sets the property before instantiating the window through `createClassObject()`. For the styles to take effect, they must be set before the window is created.

Using skins with the Window component

The Window component uses skins for its title background and close button, and a `RectBorder` instance for the border. The Window skins are found in the Flash UI Components 2/Themes/ MMDDefault/Window Assets folder in each of the theme files. For more information about skinning, see “About skinning components” in *Using Components*. For more information about the `RectBorder` class and using it to customize borders, see “[RectBorder class](#)” on page 1063.

The title background skin is always displayed. The height of the background is determined by the skin graphics. The width of the skin is set by the Window component according to the Window instance’s size. The close skins are displayed when the `closeButton` property is set to `true` and when a change state results from user interaction.

A Window component uses the following skin properties:

Property	Description
<code>skinTitleBackground</code>	The title bar. The default value is <code>TitleBackground</code> .
<code>skinCloseUp</code>	The close button. The default value is <code>CloseButtonUp</code> .

Property	Description
<code>skinCloseDown</code>	The close button in its down state. The default value is <code>CloseButtonDown</code> .
<code>skinCloseDisabled</code>	The close button in its disabled state. The default value is <code>CloseButtonDisabled</code> .
<code>skinCloseOver</code>	The close button in its over state. The default value is <code>CloseButtonOver</code> .

The following example demonstrates how to create a new movie clip symbol to use as the title background.

To set the title of a Window component to a custom movie clip symbol:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `TitleBackground`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.
6. The identifier is automatically filled out with `TitleBackground`.
7. Set the AS 2.0 class to `mx.skins.SkinElement`.

`SkinElement` is a simple class that can be used for all skin elements that don't provide their own ActionScript implementation. It provides movement and sizing functionality required by the version 2 of the Macromedia Component Architecture component framework.

8. Make sure that Export in First Frame is already selected, and click OK.
9. Open the new symbol for editing.
10. Use the drawing tools to create a box with a red fill and black line.
11. Set the border style to hairline.
12. Set the box, including the border, so that it is positioned at (0,0) and has a width of 100 and height of 22.

The Window component sets the proper width of the skin as needed but it uses the existing height as the height of the title.

13. Click the Back button to return to the main timeline.
14. Drag the Window component to the Stage.
15. Select Control > Test Movie.

Window class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > ScrollView > Window

ActionScript Class Name mx.containers.Window

The properties of the Window class let you do the following at runtime: set the title caption, add a close button, and set the display content. Setting a property of the Window class with ActionScript overrides the parameter of the same name set in the Property inspector or Component Inspector panel.

The best way to instantiate a window is to call `PopUpManager.createPopUp()`. This method creates a window that can be modal (overlapping and disabling existing objects in an application) or nonmodal. For example, the following code creates a modal Window instance (the last parameter indicates modality):

```
var newWindow = PopUpManager.createPopUp(this, Window, true);
```

Flash simulates modality by creating a large transparent window underneath the Window component. Because of the way transparent windows are rendered, you may notice a slight dimming of the objects under the transparent window. You can set the effective transparency by changing the `_global.style.modalTransparency` value from 0 (fully transparent) to 100 (opaque). If you make the window partially transparent, you can also set the color of the window by changing the Modal skin in the default theme.

If you use `PopUpManager.createPopUp()` to create a modal window, you must call `Window.deletePopUp()` to remove it so that the transparent window is also removed. For example, if you use the close button in the window, you would write the following code:

```
var win = PopUpManager.createPopUp(_root, Window, true,
    {closeButton:true});
function click(evt){
    evt.target.deletePopUp();
}
win.addEventListener("click", this);
```

Code does not stop executing when a modal window is created. In other environments (for example, Microsoft Windows), if you create a modal window, the lines of code that follow the creation of the window do not run until the window is closed. In Flash, the lines of code are run after the window is created and before it is closed.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.Window.version);
```

NOTE

The code `trace(myWindowInstance.version);` returns `undefined`.

Method summary for the Window class

The following table lists the method of the Window class.

Method	Description
Window.deletePopup()	Removes a window instance created by PopupManager.createPopup() .

Methods inherited from the UIObject class

The following table lists the methods the Window class inherits from the UIObject class.

When calling these methods from the Window object, use the form

WindowInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it is redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `Window` class inherits from the `UIComponent` class. When calling these methods from the `Window` object, use the form `WindowInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `Window` class

The following table lists properties of the `Window` class.

Property	Description
<code>Window.closeButton</code>	Indicates whether a close button is (<code>true</code>) or is not (<code>false</code>) included on the title bar.
<code>Window.content</code>	A reference to the content (root movie clip) of the window (read-only).
<code>Window.contentPath</code>	Sets the name of the content to display in the window.
<code>Window.title</code>	The text that appears in the title bar.
<code>Window.titleStyleDeclaration</code>	The style declaration that formats the text in the title bar.

Properties inherited from the `UIObject` class

The following table lists the properties the `Window` class inherits from the `UIObject` class. When accessing these properties from the `Window` object, use the form `WindowInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	Read-only; the position of the bottom edge of the object, relative to the bottom edge of its parent.
<code>UIObject.height</code>	Read-only; the height of the object, in pixels.
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.right</code>	Read-only. The position of the right edge of the object, relative to the right edge of its parent.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.

Property	Description
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	Read-only; the width of the object, in pixels.
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Window` class inherits from the `UIComponent` class. When accessing these properties from the `Window` object, use the form `WindowInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `Window` class

The following table lists the events of the `Window` class.

Event	Description
<code>Window.click</code>	Broadcast when the close button is clicked (released).
<code>Window.complete</code>	Broadcast when a window is created.
<code>Window.mouseDownOutside</code>	Broadcast when the mouse is clicked (released) outside the modal window.

Events inherited from the UIObject class

The following table lists the events the Window class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Window class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Window.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObject:Object) {
    // ...
};
windowInstance.addEventListener("click", listenerObject);
```

Usage 2:

```
on (click) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the close button.

The first usage example uses a dispatcher/listener event model. A component instance (*windowInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a Window instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the Window instance `myWindow`, sends `“_level0.myWindow”` to the Output panel:

```
on(click){
    trace(this);
}
```

Example

The following example creates a modal window with a close button. It defines a click handler that calls the `click()` method to delete the window when the user clicks the button. You drag a Window component from the Components panel to the current document's library; then add the following code to Frame 1:

```
/**
 * Requires:
 * - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, contentPath:"http://www.flash-mx.com/images/
    image1.jpg"});
var winListener:Object = new Object();
winListener.click = function() {
    my_win.deletePopUp();
};
my_win.addEventListener("click", winListener);
```

See also

[EventDispatcher.addEventListener\(\)](#), [Window.closeButton](#)

Window.closeButton

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.closeButton

Description

Property; a Boolean value that indicates whether the title bar should have a close button (true) or not (false). This property must be set in the *initObject* parameter of the [PopUpManager.createPopUp\(\)](#) method. The default value is false.

Clicking the close button broadcasts a `click` event, but doesn't close the window. You must write a handler that calls `Window.deletePopUp()` to explicitly close the window. For more information about the `click` event, see [Window.click](#).

Example

The following example creates a pop-up window and sets the `closeButton` property to add a close button to it. You drag a `Window` component from the Components panel to the current document's library, and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, contentPath:"http://www.flash-mx.com/images/
    image1.jpg"});
```

See also

[PopUpManager.createPopUp\(\)](#), [Window.click](#)

Window.complete

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
    ...
}
windowInstance.addEventListener("complete", listenerObject)
```

Description

Event; broadcast to all registered listeners when a window is created. Use this event to size a window to fit its contents.

A component instance (*windowInstance*) dispatches an event (in this case, *complete*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the [EventDispatcher.addEventListener\(\)](#) method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

Example

The following example creates a window and then defines a *complete* handler that resizes the window to fit its contents. You drag a **Window** component from the Components panel to the current document’s library, and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, contentPath:"http://www.flash-mx.com/images/
    image1.jpg"});
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
    my_win.deletePopUp();
};
winListener.complete = function(evt_obj:Object) {
    my_win.setSize(my_win.content._width, my_win.content._height + 25);
}
my_win.addEventListener("click", winListener);
my_win.addEventListener("complete", winListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

Window.content

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.content

Description

Read-only property; a reference to the content (root movie clip) of the window. This property returns a MovieClip object. When you attach a symbol from the library, the default value is an instance of the attached symbol. When you load content from a URL, the default value is `undefined` until the load operation has started.

Example

The following example creates a window and then defines a complete handler that resizes the window to fit its contents. It uses the `content` property to reference the width of the window's movie clip content. You drag a Window component from the Components panel to the current document's library, and then add the following code to Frame 1:

```
/**
 *
 * Requires:
 *   - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, contentPath:"http://www.flash-mx.com/images/
    image1.jpg"});
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
    my_win.deletePopUp();
};
winListener.complete = function(evt_obj:Object) {
    my_win.setSize(my_win.content._width, my_win.content._height + 25);
}
my_win.addEventListener("click", winListener);
my_win.addEventListener("complete", winListener);
```

Window.contentPath

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.contentPath

Description

Property; sets the name of the content to display in the window. This value can be the linkage identifier of a movie clip in the library, or the absolute or relative URL of a SWF or JPEG file to load. The default value is "" (an empty string).

Example

The following example creates a window and uses the `contentPath` property to specify the location of the image to display in the window. You drag a `Window` component from the Components panel to the current document's library, and then add the following code to Frame 1:

```
/**
 * Requires:
 * - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

// Create window.
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true, {
    contentPath:"http://www.flash-mx.com/images/image2.jpg"});
```

Window.deletePopUp()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.deletePopUp()

Parameters

None.

Returns

Nothing.

Description

Method; deletes the window instance and removes the modal state. This method can be called only on Window instances that were created by `PopUpManager.createPopUp()`.

Example

The following example creates a modal window and then defines a click handler that calls the `deletePopUp()` function to delete the window. You drag a Window component from the Components panel to the current document's library, and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Window component in library
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, contentPath:"http://www.flash-mx.com/images/
    image1.jpg"});
var winListener:Object = new Object();
winListener.click = function() {
    my_win.deletePopUp();
};
my_win.addEventListener("click", winListener);
```

Window.mouseDownOutside

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
var listenerObject:Object = new Object();
listenerObject.mouseDownOutside = function(eventObject:Object) {
    // ...
};
windowInstance.addEventListener("mouseDownOutside", listenerObject);
```

Usage 2:

```
on (mouseDownOutside) {
    // ...
}
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) outside the modal window. This event is rarely used, but you can use it to dismiss a window if the user tries to interact with something outside it.

The first usage example uses a dispatcher/listener event model. A component instance (*windowInstance*) dispatches an event (in this case, *mouseDownOutside*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see [“EventDispatcher class” on page 499](#).

The second usage example uses an `on()` handler and must be attached directly to a `Window` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `Window` instance `myWindowComponent`, sends “`_level0.myWindowComponent`” to the Output panel:

```
on (mouseDownOutside) {  
    trace(this);  
}
```

Example

The following example creates a window instance and defines a `mouseDownOutside` handler that displays a message if the user clicks outside the window. You drag a `Window` component from the Components panel to the current document’s library, and then add the following code to Frame 1:

```
/**  
    Requires:  
    - Window component in library  
*/  
  
import mx.managers.PopUpManager;  
import mx.containers.Window;  
  
System.security.allowDomain("http://www.flash-mx.com");  
  
// Create window.  
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,  
    undefined, true);  
  
// Create a listener object.  
var winListener:Object = new Object();  
winListener.mouseDownOutside = function(evt_obj:Object)  
{  
    trace("mouseDownOutside event triggered.");  
}  
// Add listener.  
my_win.addEventListener("mouseDownOutside", winListener);
```

See also

[EventDispatcher.addEventListener\(\)](#)

Window.title

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.title

Description

Property; a string indicating the text of the title bar. The default value is "" (an empty string).

Example

The following example creates a pop-up window and uses the `title` property to set the title to “Hello World”. You drag a `Window` component from the Components panel to the current document’s library, and then add the following code to Frame 1:

```
/**
 * Requires:
 *   - Window component in library
 */

import mx.managers.PopUpManager
import mx.containers.Window

// Create window.
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true);

// Set window attributes.
my_win.title = "Hello World!";
my_win.setSize(200, 100);
my_win.move(20, 20);
```

Window.titleStyleDeclaration

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

windowInstance.titleStyleDeclaration

Description

Property; a string indicating the style declaration that formats the title bar of a window. The default value is `undefined`, which indicates bold, white text.

Example

The following example creates a CSS style declaration to make text 14 points in size, italicized and underlined. It uses the `titleStyleDeclaration` property to apply that style to the title of the pop-up window that it creates. You drag a `Window` component from the Components panel to the current document's library, and then add the following code to Frame 1.

```
/**
 * Requires:
 *   - Window component in library
 */

import mx.styles.CSSStyleDeclaration
import mx.managers.PopUpManager
import mx.containers.Window

// Create a new CSSStyleDeclaration named TitleStyles
// and list it with the global styles list.
_global.styles.TitleStyles = new CSSStyleDeclaration();

// Customize styles.
_global.styles.TitleStyles.fontStyle = "italic";
_global.styles.TitleStyles.textDecoration = "underline";
_global.styles.TitleStyles.color = 0xff0000;
_global.styles.TitleStyles.fontSize = 14;

// Create window.
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
    {closeButton:true, titleStyleDeclaration:"TitleStyles"});
```

```
// Set window attributes.
my_win.title = "Testing Styles";
my_win.setSize(200, 100);
my_win.move(20, 20);

// Create listener object.
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
    trace("closing window");
    evt_obj.target.deletePopUp();
};
// Add listener.
my_win.addEventListener("click", winListener);
```

For more information about styles, see “Using styles to customize component color and text” in *Using Components*.

XMLConnector component (Flash Professional only)

The XMLConnector component lets you read or write XML documents using HTTP GET and POST operations. It acts as a connector between other components and external XML data sources. The XMLConnector component communicates with other components in your application using either ActionScript code or data binding features in the Flash authoring environment. The XMLConnector component has properties, methods, and events, but it has no visual appearance at runtime.

Using the XMLConnector component (Flash Professional only)

The XMLConnector component provides your application with access to any external data source that returns or receives XML through HTTP. The easiest way to connect with an external XML data source and use the parameters and results of that data source for your application is to specify a *schema*, the structure of the XML document that identifies the data elements in the document to which you can bind.

For more information on working with the XMLConnector component, see “Connecting to XML data with the XMLConnector component (Flash Professional only)” in *Using Flash*.

XMLConnector parameters

You can set the following authoring parameters for each XMLConnector component instance in the Parameters tab of the Component inspector:

URL is a string that points to an external XML data source.

direction is a string that defines what HTTP operation to perform when the `XMLConnector.trigger()` method is called. This parameter can have the value "send", "receive", or "send/receive".

A value of "send" means that the XML data is sent (via HTTP POST) to the URL, but Flash ignores any data that comes back. The `XMLConnector.results` property is never set to anything, and no `result` event occurs.

A value of "receive" means that no data is sent out to the XML URL. Flash accesses the URL via HTTP GET, and expects valid XML data to come back.

A value of "send/receive" means that Flash sends the XML data via HTTP POST, and expects valid XML data to come back.

If the `direction` parameter is `null`, or unrecognized, the default value is "send/receive".

`ignoreWhite` is a Boolean value; the default setting is `false`. When this parameter is set to `true`, the text nodes that contain only white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected.

`multipleSimultaneousAllowed` is a Boolean value; when set to `true`, it allows a `trigger()` operation to initiate when another `trigger()` operation is already in progress. Multiple simultaneous `trigger()` operations may not be completed in the same order they were called. Also, Flash Player may place limits on the number of simultaneous network operations. This limit varies by version and platform. When the parameter is set to `false`, a `trigger()` operation cannot initiate if another one is in progress.

`suppressInvalidCall` is a Boolean value; when set to `true`, it suppresses the `trigger()` operation if the data parameters are invalid. When `suppressInvalidCall` is set to `false`, the `trigger()` operation executes and uses invalid data if necessary.

Common workflow for the XMLConnector component

The following procedure outlines the typical workflow for the XMLConnector component.

To use an XMLConnector component:

1. Add an instance of the XMLConnector component to your application and give it an instance name.
2. Use the Parameters tab of the Component inspector to enter the URL for the external XML data source that you want to access.
3. Use the Schema tab of the Component inspector to specify a schema for the XML document.

NOTE

You can use the Import Sample Schema button to automate this process.

4. Use the Bindings tab of the Component inspector to bind data elements (`params` and `results`) from the XML document to properties of the visual components in your application.

For example, you can connect to an XML document that provides weather data and bind the Location and Temperature data elements to label components in your application. The name and temperature of a specified city appears in the application at runtime.

5. Add a trigger to initiate the data binding operation by using one of the following methods:
 - Attach the Trigger Data Source behavior to a button.
 - Add your own ActionScript to call the `trigger()` method on the XMLConnector component.
 - Create a binding between an XML parameter and a UI control and set its Kind property to AutoTrigger. For more information, see “Schema kinds” in *Using Flash*.

For a step-by-step example that connects and displays XML using the XMLConnector component, see “XML Tutorial: Timesheet” in the Data Integration tutorials at www.macromedia.com/go/data_integration.

XMLConnector class (Flash Professional only)

Inheritance RPCCall > XMLConnector

ActionScript Class Name mx.data.components.XMLConnector

The XMLConnector class lets you send or receive XML files using HTTP. You can use ActionScript to bind other components to a data source that returns XML data, allowing communication between the components.

Method summary for the XMLConnector class

The following table lists the method of the XMLConnector class.

Method	Description
<code>XMLConnector.trigger()</code>	Initiates a remote procedure call.

Property summary for the XMLConnector class

The following table lists properties of the XMLConnector class.

Property	Description
<code>XMLConnector.direction</code>	Indicates whether data is being sent, received, or both.
<code>XMLConnector.ignoreWhite</code>	Indicates whether text nodes containing only white space are discarded during the parsing process.
<code>XMLConnector.multipleSimultaneousAllowed</code>	Indicates whether multiple calls can take place at the same time.
<code>XMLConnector.params</code>	Specifies data that is sent to the server when the next <code>trigger()</code> operation is executed.
<code>XMLConnector.results</code>	Identifies data that was received from the server as a result of the <code>trigger()</code> operation.
<code>XMLConnector.suppressInvalidCalls</code>	Indicates whether to suppress a call if parameters are invalid.
<code>XMLConnector.URL</code>	The URL used by the component in HTTP operations.

Event summary for the XMLConnector class

The following table lists events of the XMLConnector class.

Event	Description
<code>XMLConnector.result</code>	Broadcast when a remote procedure call completes successfully.
<code>XMLConnector.send</code>	Broadcast when the <code>trigger()</code> method is in process, after the parameter data has been gathered but before the data is validated and the remote procedure call is initiated.
<code>XMLConnector.status</code>	Broadcast when a remote procedure call is initiated, to inform the user of the status of the operation.

XMLConnector.direction

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.direction

Description

Property; indicates whether data is being sent, received, or both. The values are the following:

- `send` XML data for the `params` property is sent by HTTP `POST` method to the URL for the XML document. Any data that is returned is ignored. The `results` property is not set to anything, and there is no `result` event.

NOTE

The `params` and `results` properties and the `result` event are inherited from the RPC component API.

- `receive` No `params` data is sent to the URL. The URL for the XML document is accessed through HTTP `GET`, and valid XML data is expected from the URL.
- `send/receive` The `params` data is sent to the URL, and valid XML data is expected from the URL.

Example

The following example sets the `direction` to `receive` for the document `mysettings.xml`:

```
myXMLConnector.direction = "receive";  
myXMLConnector.URL = "mysettings.xml";  
myXMLConnector.trigger();
```

XMLConnector.ignoreWhite

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.ignoreWhite

Description

Property; a Boolean value. When this parameter is set to `true`, the text nodes that contain only white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected. The default setting is `false`.

Example

The following code sets the `ignoreWhite` property to `true`:

```
myXMLConnector.ignoreWhite = true;
```

XMLConnector.multipleSimultaneousAllowed

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.multipleSimultaneousAllowed

Description

Property; indicates whether multiple calls can take place at the same time. If this property is `false`, the `XMLConnector.trigger()` method performs a call if another call is already in progress. A status event is emitted, with the code `CallAlreadyInProgress`. If this property is `true`, the call takes place.

When multiple calls are simultaneously in progress, there is no guarantee that they will be completed in the order in which they were triggered. Also, the browser and/or operating system may place limits on the number of simultaneous network operations. The most likely limit you may encounter is the browser enforcing a maximum number of URLs that can be downloaded simultaneously. This is something that is often configurable in a browser.

However, even in this case, the browser should queue streams, and this should not interfere with the expected behavior of the Flash application.

Example

This example retrieves a remote XML file using the XMLConnector component by setting the `direction` property to `receive`. Drag an XMLConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
    trace("results:");
    trace(evt.target.results);
    trace("");
};
xmlListener.status = function(evt:Object) {
    trace("status: "+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

This example specifies the URL of the XML file, and sets `multipleSimultaneousAllowed` to `false`. It triggers the XMLConnector instance three times, which causes the event listener's `status` method to display the error code `CallAtReadyInProgress` two times in the Output panel. The first attempt is successfully sent from Flash to the server. When the first trigger successfully receives a result, the `result` event is broadcast and the XML packet you receive is displayed in the Output panel.

XMLConnector.params

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.params

Description

Property; specifies data that will be sent to the server when the next `trigger()` operation is executed. Each RPC component defines how this data is used, and what the valid types are.

Example

The following example defines `name` and `city` parameters for `myXMLConnector`:

```
myXMLConnector.params = new XML("<mydoc><name>Bob</name><city>Oakland</city></mydoc>");
```

XMLConnector.result

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.addEventListener("result", myListenerObject)
```

Description

Event; broadcast when a remote procedure call completes successfully.

The parameter to the event handler is an object with the following fields:

- `type`: the string "result"
- `target`: a reference to the object that emitted the event (for example, a `WebServiceConnector` component)

You can retrieve the actual result value using the `results` property.

Example

The following example defines a function `res` for the `result` event and assigns the function to the `addEventListener` event handler:

```
var res = function (ev) {  
    trace(ev.target.results);  
};  
xcon.addEventListener("result", res);
```

XMLConnector.results

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.results

Description

Property; identifies data that was received from the server as a result of a `trigger()` operation. Each RPC component defines how this data is fetched, and what the valid types are. This data appears when the RPC operation has successfully completed, as signaled by the `result` event. It is available until the component is unloaded, or until the next RPC operation.

It is possible for the returned data to be very large. You can manage this in two ways:

- Select an appropriate movie clip, timeline, or screen as the parent for the RPC component. The component's memory becomes available for garbage collection when the parent is destroyed.
- In ActionScript, you can assign `null` to this property at any time.

Example

The following example traces the `results` property for `myXMLConnector`:

```
trace(myXMLConnector.results);
```

XMLConnector.send

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.addEventListener("send", myListenerObject)

Description

Event; broadcast when the `trigger()` operation is in process, after the parameter data has been gathered but before the data is validated and the remote procedure call is initiated. This is a good place to put code that modifies the parameter data before the call.

The parameter to the event handler is an object with the following fields:

- `type`: the string "send"
- `target`: a reference to the object that emitted the event (for example, a `WebServiceConnector` component)

You can retrieve or modify the actual parameter values by using the `params` property.

Example

The following example defines a function `sendFunction` for the `send` event and assigns the function to the `addEventListener` event handler:

```
var sendFunction = function (sendEnv) {
    sendEnv.target.params = [newParam_txt.text];
};
xcon.addEventListener("send", sendFunction);
```

XMLConnector.status

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
componentInstance.addEventListener("status", myListenerObject)
```

Description

Event; broadcast when a remote procedure call is initiated, to inform the user of the status of the operation.

The parameter to the event handler is an object with the following fields:

- `type`: the string "status"
- `target`: a reference to the object that emitted the event (for example, a `WebServiceConnector` component)
- `code`: a string giving the name of the specific condition that occurred
- `data`: an object whose contents depend on the code

The code field for the status event is set to Fault if problems occur with the call, as follows:

Code	Data	Description
Fault	{faultcode: code, faultstring: string, detail: detail, element: element, faultactor: actor}	This event is emitted if other problems occur during the processing of the call. The data is a SOAPFault object. After this event occurs, the attempted call is considered complete, and there is no result or send event.

The following are the faults that can occur with the status event:

FaultCode	FaultString	Notes
XMLConnector.Not.XML	params is not an XML object	The params value must be an ActionScript XML object.
XMLConnector.Parse.Error	params had XML parsing error NN.	The status property of the params XML object had a nonzero value NN. To see the possible errors NN, see XML.status in <i>ActionScript 2.0 Language Reference</i> .
XMLConnector.No.Data.Received	no data was received from the server	Due to various browser limitations, this message can mean either (a) the server URL was invalid, did not respond, or returned an HTTP error code; or (b) the server request succeeded but the response was 0 bytes of data. To work around this restriction, design your application so that the server never returns 0 bytes of data. If you receive the fault code XMLConnector.No.Data.Received, you will know that there was a server error, and can inform the user accordingly.

FaultCode	FaultString	Notes
<code>XMLConnector.Results.Parse.Error</code>	received data had an XML parsing error NN	The received XML was not valid, as determined by the Flash Player built-in XML parser. To see the possible errors NN, see <i>XML.status</i> in <i>ActionScript 2.0 Language Reference</i> .
<code>XMLConnector.Params.Missing</code>	Direction is 'send' or 'send/receive', but params are null.	

Example

The following example defines a function `statusFunction` for the `status` event and assigns the function to the `addEventListener` event handler:

```
var statusFunction = function (stat) {
    trace(stat.code);
    trace(stat.data.faultcode);
    trace(stat.data.faultstring);
};
xcon.addEventListener("status", statusFunction);
```

XMLConnector.suppressInvalidCalls

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

`componentInstance.suppressInvalidCalls`

Description

Property; indicates whether to suppress a call if parameters are invalid. If this property is `true`, the `trigger()` method does not perform a call if the bound parameters fail the validation. A `status` event is emitted, with the code `InvalidParams`. If this property is `false`, the call takes place, using the invalid data as required.

Example

This example displays an error because the required parameters are not being passed. Drag an XMLConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
    trace("results:");
    trace(evt.target.results);
    trace("");
};
xmlListener.status = function(evt:Object) {
    switch (evt.code) {
        case 'Fault' :
            trace("ERROR! ["+evt.data.faultcode+"]");
            trace("\t"+evt.data.faultstring);
            break;
        case 'InvalidParams' :
            trace("ERROR! ["+evt.code+"]");
            break;
    }
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "send/receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/login_xml.cfm";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = false;
// myXMLConnector.params = new XML("<login username='Mort'
    password='Guacamole' />");
myXMLConnector.trigger();
```

Remove the comments from the second to last line of code for the snippet to work correctly.

XMLConnector.trigger()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.trigger()

Description

Method; initiates a remote procedure call by the XMLConnector component. This can be either getting or posting to the specified XML file. If the operation is successful, the results of the operation appear in the RPC component's `results` property.

The `trigger()` method performs the following steps:

1. If any data is bound to the `params` property, the method executes all the bindings to ensure that up-to-date data is available. This also causes data validation to occur.
2. If the data is not valid and `suppressInvalidCalls` is set to `true`, the operation is discontinued.
3. If the operation continues, the `send` event is emitted.
4. The actual remote call is initiated using the connection method indicated (for example, HTTP).

Example

This example retrieves a remote XML file using the XMLConnector by setting the `direction` property to `receive`. Drag an XMLConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
    trace("results:");
    trace(evt.target.results);
    trace("");
};
xmlListener.status = function(evt:Object) {
    trace("status::"+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

This code specifies the URL of the XML file and sets `multipleSimultaneousAllowed` to `false`. It triggers the XMLConnector instance three times, which causes the event listener's `status` method to display the error code `CallAtReadyInProgress` two times in the Output panel. The first attempt is successfully sent from Flash to the server. When the first trigger successfully receives a result, the `result` event is broadcast and the XML packet you receive is displayed in the Output panel.

XMLConnector.URL

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

componentInstance.URL

Description

Property; the URL that this component uses when carrying out HTTP operations. This URL may be an absolute or relative URL. The URL is subject to all the standard Flash Player security protections (for more information about Flash Player security protections, see “Understanding Security” in *Learning ActionScript 2.0 in Flash*).

Example

This example retrieves a remote XML file using the XMLConnector component by setting the `direction` property to `receive`. Drag an XMLConnector component into your library, and enter the following code on Frame 1 of the timeline:

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
    trace("results:");
    trace(evt.target.results);
    trace("");
};
xmlListener.status = function(evt:Object) {
    trace("status: "+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

This code specifies the URL of the XML file and sets `multipleSimultaneousAllowed` to `false`. It triggers the XMLConnector instance three times, which causes the event listener's `status()` method to display the error code `CallAlreadyInProgress` two times in the Output panel. The first attempt is successfully sent from Flash to the server. When the first trigger successfully receives a result, the `result` event is broadcast and the XML packet you receive is displayed in the Output panel.

ActionScript Class Name `mx.xpath.XPathAPI`

The XPathAPI class allows you to do simple XPath searches within Macromedia Flash. This can be very useful for searching XML packets based on node names and attribute values. In other words, you can quickly find nodes and attributes in an XML document using the XPathAPI methods.

In order to use XPath searches within Flash, you first need to include the XPathAPI class into your Flash library by adding the DataBindingClass (if it hasn't been added already). If you've already set up bindings, this class may have been included automatically; otherwise, you need to select the class from the common libraries (Window > Common Libraries > Classes). From the Classes.fla library panel, you can simply drag a copy of the DataBindingClasses component into your current Flash document's library. Now, you can import the class by typing `import mx.xpath.XPathAPI` or by using the classes fully qualified name when accessing its methods by prefixing the class methods with `mx.xpath.XPathAPI.method_name`.

For more information about this class, see the Flash Documentation Resource Center at www.macromedia.com/go/xpathapi.

XUpdateResolver component (Flash Professional only)

Resolver components are used with the DataSet component (part of the data management functionality in the Flash data architecture) to save changes to an external data source. Resolvers take a `DataSet.deltaPacket` object and convert it to an update packet in a format appropriate to the type of resolver. The update packet can then be transmitted to the external data source by one of the connector components. Resolver components have no visual appearance at runtime.

For general information on how to manage data in Flash using the DataSet component, see “Data management (Flash Professional only)” in *Using Flash*.

XUpdate is a standard for describing changes that are made to an XML document and is supported by a variety of XML databases, such as Xindice and XHive. The XUpdateResolver component translates the changes made to a DataSet component into XUpdate statements. The updates from the XUpdateResolver component are sent in the form of an XUpdate data packet, which is communicated to the database or server through a connection object. The XUpdateResolver component gets a delta packet from a DataSet component, sends its own update packet to a connector, receives server errors back from the connection, and communicates them back to the DataSet component—all using bindable properties.

For information about the working draft of the XUpdate language specification, see <http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html>. For information about the Flash data architecture, see “Data resolution (Flash Professional only)” in *Using Flash*; for information about resolving XML data, see “Resolving XML data with the XUpdateResolver component (Flash Professional only)” in *Using Flash*.

NOTE

You can also use the XUpdateResolver component to send data updates to any external data source that can parse the XUpdate language—for example, an ASP page, a Java servlet, or a ColdFusion component.

Using the XUpdateResolver component (Flash Professional only)

The XUpdateResolver component is used only when your Flash application contains a DataSet component and must send an update back to an external data source.

The XUpdateResolver component communicates with the DataSet component by using the DataSetDeltaToXUpdateDelta encoder. This encoder creates XPath statements that uniquely identify nodes within an XML file according to the information contained in the DataSet component's delta packet. This information is used by the XUpdateResolver component to generate XUpdate statements. For more information on the DataSetDeltaToXUpdateDelta encoder, see "Schema encoders" in *Using Flash*.

For more information on working with the XUpdateResolver component, see "Data resolution (Flash Professional only)" in *Using Flash*.

XUpdateResolver component parameter

The XUpdateResolver component has one authoring parameter, the Boolean `includeDeltaPacketInfo` parameter. When this parameter is set to `true`, the update packet includes additional information that can be used by an external data source to generate results that can be sent back to your application. This information includes a unique transaction and operation ID that is used internally by the data set.

NOTE

The additional information that is included in the update packet invalidates the XUpdate. You would choose to add this information only if you were going to store it in a server object and use it to generate a result packet. In this scenario, your server object would pull the information out of the update packet for its own needs and then pass on the (now valid) XUpdate to the database.

The following is an example of an XML update packet when the `includeDeltaPacketInfo` parameter is set to `false`:

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:remove select="/datapacket/row[@id='100']"/>
</xupdate:modifications>
```


The following is an example of an XML update packet when the `includeDeltaPacketInfo` parameter is set to `true`:

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate"
  transId="46386292065:Wed Jun 25 15:52:34 GMT-0700 2003">
  <xupdate:remove select="/datapacket/row[@id='100']" opId="0123456789"/>
</xupdate:modifications>
```

Common workflow for the XUpdateResolver component

The following procedure outlines the typical workflow for the XUpdateResolver component.

To use an XUpdateResolver component:

1. Add two instances of the XMLConnector component and one instance each of the DataSet component and the XUpdateResolver component to your application, and give them instance names.
2. Select the first XMLConnector component, and use the Parameters tab of the Component inspector to enter the URL for the external XML data source that you want to access.
3. With the XMLConnector component still selected, click the Schema tab of the Component inspector and import a sample XML file to generate your schema.

NOTE

You may need to create a virtual schema for your XML file if you want to access a subelement of the array that you are binding to the data set. For more information, see “Virtual schemas” in *Using Flash*.

4. Use the Bindings tab of the Component inspector to bind an array in the XMLConnector component to the `dataProvider` property of the DataSet component.
5. Select the DataSet component and use the Schema tab of the Component inspector to create the DataSet fields that will be bound to the fields of the object within the array.
6. Use the Bindings tab of the Component inspector to bind data elements (DataSet fields) to the visual components in your application.
7. Select the Schema tab of the XUpdateResolver component. With the `deltaPacket` component property selected, use the Schema Attributes pane to set the `encoder` property to the `DataSetDeltaToXUpdateDelta` encoder.

8. Select Encoder Options and enter the `rowNodeKey` value that uniquely identifies the row node within the XML file.

NOTE

The `rowNodeKey` value combines an XPath statement with a field parameter to define how unique XPath statements should be generated for the update data contained within the delta packet. See information on the `DataSetDeltaToXUpdateDelta` encoder in “Schema encoders” in *Using Flash*.

9. Click the Bindings tab and create a binding between the `XUpdateResolver` component’s `deltaPacket` property and the `DataSet` component’s `deltaPacket` property.
10. Create another binding from the `xupdatePacket` property to the second `XMLConnector` component to send the data back to the external data source.

NOTE

The `xupdatePacket` property contains the formatted delta packet (`XUpdate` statements) that will be sent to the server.

11. Add a trigger to initiate the data binding operation: use the Trigger Data Source behavior attached to a button, or add `ActionScript`.

In addition to these steps, you can also create bindings to apply the result packet sent back from the server to the data set by the `XUpdateResolver` component.

For a step-by-step example that resolves data to an external data source using `XUpdate`, see “Update the timesheet” in the Data Integration tutorials at www.macromedia.com/go/data_integration.

XUpdateResolver class (Flash Professional only)

Inheritance MovieClip > XUpdateResolver

ActionScript Class Name mx.data.components.XUpdateResolver

The properties and events of the XUpdateResolver class allow you to work with the DataSet component to save changes to external data sources.

Property summary for the XUpdateResolver class

The following table lists properties of the XUpdateResolver class.

Property	Description
XUpdateResolver.deltaPacket	Contains a description of the changes to the DataSet component. The DataSet component's <code>deltaPacket</code> property should be bound to this property so that when <code>DataSet.applyUpdates()</code> is called, the binding copies it across and the resolver creates the XUpdate packet.
XUpdateResolver.includeDeltaPacketInfo	Includes additional information from the delta packet in attributes on the XUpdate nodes.
XUpdateResolver.updateResults	Describes results of an update.
XUpdateResolver.xupdatePacket	Contains the XUpdate translation of the changes to the DataSet component.

Event summary for the XUpdateResolver class

The following table lists events of the XUpdateResolver class.

Event	Description
XUpdateResolver.beforeApplyUpdates	Called by the resolver component to make custom modifications immediately after the XML packet has been created and immediately before that packet is sent.
XUpdateResolver.reconcileResults	Called by the resolver component to compare two packets.

XUpdateResolver.beforeApplyUpdates

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.beforeApplyUpdates(eventObject)
```

Parameters

eventObject Resolver event object; describes the customizations to the XML packet before the update is sent through the connector to the database. This event object should contain the following properties:

Property	Description
target	Object; the resolver generating this event.
type	String; the name of the event.
updatePacket	XML object; the XML object that is about to be applied.

Returns

None.

Description

Event; called by the resolver component to make custom modifications immediately after the XML packet has been created for a new delta packet, and immediately before that packet is sent out using data binding. You can use this event handler to make custom modifications to the XML before sending the updated data to a connector.

Example

The following example adds the user authentication data to the XML packet:

```
on (beforeApplyUpdates) {  
    // Add user authentication data.  
    var userInfo = new XML(""+getUserId()+" "+getPassword()+"");  
    xupdatePacket.firstChild.appendChild(userInfo);  
}
```

XUpdateResolver.deltaPacket

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.deltaPacket

Description

Property; contains a description of the changes to the DataSet component. This property is of type `deltaPacket` and receives a delta packet to be translated into an XUpdate packet, and outputs a delta packet from any server results placed in the `updateResults` property. This property provides a way for you to make custom modifications to the XML before sending the updated data to a connector.

Messages in the `updateResults` property are treated as errors. This means that a delta with messages is added to the delta packet again so it can be re-sent the next time the delta packet is sent to the server. You must write code that handles deltas that have messages so that the messages are presented to the user and the deltas can be modified before being added to the next delta packet.

The DataSet component's `deltaPacket` property should be bound to this property so that when `DataSet.applyUpdates()` is called, the binding copies it across and the resolver creates the XUpdate packet.

XUpdateResolver.includeDeltaPacketInfo

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.includeDeltaPacketInfo
```

Description

Property; a Boolean property that, if *true*, includes additional information from the delta packet in attributes on the XUpdate nodes. This information consists of the transaction ID and operation ID.

For an example of the resulting XML, see [“XUpdateResolver component parameter” on page 1508](#).

XUpdateResolver.reconcileResults

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.reconcileResults(eventObject)
```

Parameters

eventObject ResolverEvent object; describes the event object used to compare two update packets. This event object should contain the following properties:

Property	Description
<i>target</i>	Object; the resolver generating this event.
<i>type</i>	String; the name of the event.

Returns

None.

Description

Event; called by the resolver component to compare two packets. Use this callback to insert any code after the results have been received from the server and immediately before the transmission, through data binding, of the delta packet that contains operation results. This is a good place to put code that handles messages from the server.

Example

The following example reconciles two update packets and clears the updates on success:

```
on (reconcileResults) {
    // Examine results.
    if(examine(updateResults))
        myDataSet.purgeUpdates();
    else
        displayErrors(results);
}
```

XUpdateResolver.updateResults

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
resolveData.updateResults
```

Description

Property; property of type `deltaPacket` that contains the results of an update returned from the server using a connector. Use this property to transmit errors and updated data from the server to a `DataSet` component—for example, when the server assigns new IDs for an auto-assigned field. Bind this property to a connector's `results` property so that it can receive the results of an update and transmit the results back to the `DataSet` component.

Messages in the `updateResults` property are treated as errors. This means that a delta with messages is added to the delta packet again so it can be re-sent the next time the delta packet is sent to the server. You must write code that handles deltas that have messages so that the messages are presented to the user and the deltas can be modified before being added to the next delta packet.

XUpdateResolver.xupdatePacket

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

resolveData.xupdatePacket

Description

Property; property of type `xm1` that contains the XUpdate translation of the changes to the DataSet component. Bind this property to the connector component's property that transmits the translated update packet of changes back to the DataSet component.

Index

A

- Accordion component
 - applying easing methods to 1315
 - creating applications 37
 - customizing 40
 - events 50
 - inheritance 47
 - methods 47
 - package 47
 - parameters 36
 - properties 49
 - using skins 42
 - using styles 41
- activators, menu 945
- Alert component
 - creating applications 66
 - customizing 67
 - events 74
 - inheritance 71
 - methods 71
 - package 71
 - parameters 66
 - properties 73
 - using skins 69
 - using styles 67
- authentication and WebService class 1441

B

- behaviors and video playback 841
- Binding class
 - about 208
 - methods 209
- borders. *See* RectBorder class

- Button component
 - about 89
 - creating applications 91
 - customizing 94
 - events 105
 - inheritance 101
 - methods 102, 1125
 - package 101
 - parameters 90
 - properties 103
 - using skins 95
 - using styles 94

C

- CellRenderer API
 - about 109
 - example 112
 - methods 117, 118
 - properties 118
 - using 111
- CheckBox component
 - about 129
 - creating applications 130
 - events 139
 - inheritance 135
 - methods 136
 - package 135
 - parameters 130
 - properties 137
 - using skins 134
 - using styles 132
- classes
 - Accordion 47
 - Alert 71
 - Binding 208
 - Button 101

- CheckBox 135
- ComboBox 165
- ComponentMixins 226
- CustomFormatter 212
- CustomValidator 216
- data binding 207
- DataGrid 262
- DataGridColumn 300
- DataHolder 315
- DataSet 335
- DataType 233
- DateChooser 417
- DateField 439
- Delegate 461
- DeltaItem 463
- DepthManager 487
- EndPoint 220
- EventDispatcher 499
- FLVPlayback 539
- FocusManager 721
- Form 735
- Label 755
- List 770
- Loader 817
- Log 1414
- Media 847
- Menu 883
- MenuBar 951
- MenuDataProvider 933
- NumericStepper 975
- PendingCall 1423
- PopUpManager 987
- ProgressBar 999
- RadioButton 1029
- RDBMSResolver 1051
- RectBorder 1063
- Screen 1071
- ScrollPane 1098
- SimpleButton 1125
- Slide 1138
- SOAPCall 1434
- StyleManager 1171
- SystemManager 1175
- TextArea 1182
- TextInput 1214
- Tree 1278
- TypedValue 245
- UIEventDispatcher 1351
- UIScrollBar 1395
- web service 1413
- WebService 1437
- WebServiceConnector 1451
- Window 1472
- XMLConnector 1491
- XUpdateResolver 1507
- Collection interface
 - about 147
 - methods 148
- columns, DataGridColumn class 300
- ComboBox component
 - about 157
 - applying easing methods to 1316
 - creating applications 160
 - events 169
 - inheritance 165
 - methods 166
 - package 165
 - parameters 159
 - properties 168
 - using skins 164
 - using styles 162
- component categories
 - data 31
 - managers 32
 - media 32
 - other 33
 - screens 32
 - UI components 30
- Component inspector, media components 840
- ComponentMixins class
 - about 226
 - methods 227
- components, applying easing methods to 1315
- cue points, using 513
- CustomFormatter class
 - about 212
 - methods 214
 - sample 212
- CustomValidator class
 - about 216
 - methods 216

D

- data binding classes
 - about 207
 - Binding class 208
 - ComponentMixins class 226
 - CustomValidator class 216
 - DataType class 233
 - EndPoint class 220
 - package 208
 - TypedValue class 245
 - using at runtime 207
- data components 31
- data models
 - DataGrid component 252
 - Menu component 886
- data sets. *See* DataSet component
- data types, supported by web services classes 1438
- DataGrid component
 - about 249
 - animating 1317
 - creating applications 254
 - customizing 258
 - data model 251, 252
 - DataGridColumn class 302
 - design 252
 - events 267
 - events, inherited 267, 268
 - inheritance 262
 - interacting with 250
 - methods 262
 - methods, inherited 263
 - package 262
 - parameters 253
 - performance strategies 256
 - properties 264, 265
 - properties inherited 265
 - properties, inherited 266
 - using 251
 - using skins 261
 - using styles 258, 259
 - view, data 251, 252
- DataGridColumn class
 - about 300
 - properties 301
- DataHolder component
 - about 313
 - creating applications 314
 - inheritance 315
 - package 315
 - properties 315
- DataProvider API
 - about 317
 - events 318
 - methods 318
 - package 317
 - properties 318
- DataSet component
 - about 331
 - common workflow 333
 - creating applications 333
 - events 338
 - inheritance 335
 - methods 336
 - package 335
 - parameters 332
 - properties 337
- DataType class
 - about 233
 - methods 234
 - properties 234
- date field component
 - customizing 435
- DateChooser component
 - about 411
 - class 417
 - creating applications 412
 - customizing 413
 - events 420
 - inheritance 417
 - methods 418
 - package 417
 - parameters 411
 - properties 419
 - using skins 415
 - using styles 413
- DateField component
 - about 433
 - creating applications 434
 - events 442
 - inheritance 439
 - methods 439
 - package 439
 - parameters 434
 - properties 441
 - using skins 438
 - using styles 436
- Delegate class
 - about 461
 - methods 461

- Delta interface
 - about 469
 - methods 469
- DeltaItem class
 - about 463
 - properties 463
- DeltaPacket interface
 - about 479
 - methods 480
- DepthManager class 487
 - methods 488
- detail property
 - PendingCall.onFault 1431
 - WebService.onFault 1446

E

- easing classes and methods, Tween class 1314
- element
 - PendingCall.onFault 1431
 - WebService.onFault 1446
- EndPoint class
 - about 220
 - methods 221
- event object 499
- EventDispatcher class
 - about 499
 - methods 500
 - package 500
- events
 - event object 499

F

- faultactor property
 - PendingCall.onFault 1431
 - WebService.onFault 1446
- faultcode property
 - PendingCall.onFault 1431
 - WebService.onFault 1446
- faultstring property
 - PendingCall.onFault 1431
 - WebService.onFault 1446
- FLVPlayback component 505
 - class 539
 - component parameters 510

- creating a new skin 532
- creating applications 507
- customizing 524
- events 546
- methods 539
- playing multiple FLVs 521
- properties 541
- using 507
- using a SMIL file 712
- using cue points 513
- VideoError class 698
- VideoPlayer class 706
- FLVs, playing 505
- FocusManager class
 - about 721
 - creating applications 724
 - customizing 725
 - events 728
 - inheritance 725
 - methods 726
 - package 725
 - properties 727
- Form class
 - about 735
 - events 741
 - inheritance 736
 - methods 737
 - package 736
 - parameters 736
 - properties 738

I

- interfaces
 - Collection 147
 - Delta 469
 - DeltaPacket 479
 - Iterator 749
 - TransferObject 1233
 - TreeDataProvider 1257
- Iterator interface
 - about 749
 - methods 749
 - package 749

L

Label component

- about 751
- creating applications 753
- customizing 753
- events 757
- inheritance 755
- methods 755
- package 755
- parameters 752
- properties 756
- using styles 753

List component

- about 761
- creating applications 764
- customizing 766
- design 109
- events 775
- inheritance 770
- methods 771
- package 770
- parameters 764
- properties 773
- scrolling behavior 110
- using skins 770
- using styles 766

Loader component

- about 813
- creating applications 815
- customizing 816
- events 820
- inheritance 817
- methods 817
- package 817
- parameters 814
- properties 818
- using skins 816
- using styles 816

loading external content 1072

Log class 1414

M

manager components 32

Media components

- about 32, 831
- behaviors 841
- Component inspector 840
- creating applications 846

customizing 847

design 833

events 850

inheritance 847

MediaController component 831

MediaDisplay component 831

MediaPlayback component 831

methods 848

packages 847

parameters 843

properties 849

using skins 847

using styles 847

MediaController component

about 836

parameters 844

MediaDisplay component

about 836

parameters 843

MediaPlayback component

about 836

parameters 845

Menu component

about 883

about XML attributes 887

adding hierarchical menus 886

creating applications 892

customizing 897

data model 886

events 904

exposing items to ActionScript 890

initialization object properties 891

menu item types 888

methods 902

parameters 892

properties 903

using skins 900

using styles 897

view 886

MenuBar component

about 945

class 951

creating applications 947

customizing 948

events 954

methods 951

parameters 946

properties 953

using skins 950

using styles 949

MenuDataProvider class
 about 933
 events 934
 methods 934
multipleSimultaneousAllowed parameter 1450

N

NumericStepper component
 about 969
 creating applications 971
 customizing 972
 events 978
 methods 976
 parameters 970
 properties 977
 using skins 974
 using styles 973

O

onFault callback function 1446
operation parameter 1450

P

PendingCall class
 about 1423
 callbacks 1425
 methods 1424
 properties 1424
PopUpManager class 987
ProgressBar component
 about 991
 creating applications 993
 customizing 996
 events 1002
 methods 999
 parameters 992
 properties 1000
 using skins 998
 using styles 996

R

RadioButton component
 about 1023
 creating applications 1024
 customizing 1025

events 1033
methods 1029
parameters 1024
properties 1030
using skins 1027
using styles 1026
RDBMSResolver component
 about 1047
 common workflow 1050
 events 1052
 methods 1051
 parameters 1048
 properties 1051
RectBorder class
 about 1063
 using styles 1064

S

schema types, XML 1438
Screen class
 about 1071
 events 1078
 loading external content 1072
 methods 1074
 properties 1076
 referencing screens 1073
screen components 32
ScrollPane component
 about 1093
 creating applications 1095
 customizing 1096
 events 1102
 methods 1098
 parameters 1094
 properties 1100
 using skin 1097
 using styles 1097
security, and WebService class 1441
separator menu items 888
SimpleButton class 1125
 about 1125
 events 1128
 methods 1125
 properties 1126
skin, customizing FLVPlatyback 524
Slide class 1135
 events 1143
 example 1137

- inheritance 1138
- methods 1138
- package 1138
- parameters 1136
- properties 1140
- SOAPCall class
 - about 1434
 - properties 1435
- SOAPFault object 1446
- StyleManager class
 - about 1171
 - methods 1171
- styles
 - RectBorder class 1064
 - See also* individual component names
- suppressInvalidCalls parameter 1450
- SystemManager class
 - about 1175
 - properties 1175

T

- tab order, for components 721
- tables. *See* DataGrid component
- TextArea component
 - about 1177
 - creating applications 1179
 - customizing 1180
 - events 1186
 - inheritance 1182
 - methods 1183
 - package 1182
 - parameters 1178
 - properties 1184
 - using skins 1182
 - using styles 1180
- TextArea.styleSheet 1202
- TextInput component 1209
 - about 1209
 - class 1214
 - creating applications 1211
 - customizing 1212
 - events 1218
 - methods 1215
 - parameters 1210
 - properties 1216
 - using 1210
 - using styles 1212

- TransferObject interface
 - about 1233
 - methods 1233
- TransitionManager class
 - Blinds transition 1250
 - events 1239
 - Fade transition 1251
 - Fly transition 1251
 - Iris transition 1252
 - methods 1239
 - parameters 1238
 - Photo transition 1253
 - PixelDissolve transition 1253
 - properties 1239
 - Rotate transition 1254
 - Squeeze transition 1255
 - transition-based classes 1249
 - Wipe transition 1255
 - Zoom transition 1256
- Tree component
 - creating applications 1268
 - customizing 1273
 - events 1283
 - inheritance 1278
 - methods 1280
 - package 1278
 - parameters 1268
 - properties 1281
 - using skins 1278
 - using styles 1274
 - XML formatting 1266
- TreeDataProvider interface
 - about 1257
 - methods 1257
 - properties 1258
- Tween class
 - Accordion component 1315
 - applying easing methods to components 1315
 - ComboBox component 1316
 - DataGrid component 1317
 - easing classes and methods 1314
 - events 1312
 - methods 1311
 - parameters 1313
 - properties 1312
- TypedValue class
 - about 245
 - properties 245
- types. *See* data types

U

- UI components 30
- UIComponent class
 - about 1339
 - events 1342
 - inheritance 1339
 - methods 1340
 - package 1339
 - properties 1341
- UIEventDispatcher class
 - about 1351
 - events 1352
 - methods 1351
- UIObject class 1359
 - about 1311, 1359
 - events 1312, 1361
 - inheritance 1359
 - methods 1311, 1360
 - package 1359
 - properties 1312, 1360
- UIScrollBar component
 - about 1389
 - creating applications 1390
 - customizing 1393
 - events 1399
 - inheritance 1395
 - methods 1396
 - package 1395
 - parameters 1390
 - properties 1397
 - using skins 1394
 - using styles 1393
- user interface components 30

V

- video playback 841
- VideoError class
 - defined 698
 - properties 698
- VideoPlayer class 706
 - events 711
 - methods 707
 - properties 707
- view, Menu component 886

W

- web service classes
 - about 1413
 - Log class 1414
 - PendingCall class 1423
 - SOAPCall class 1434
 - using at runtime 1414
 - WebService class 1437
- WebService class
 - about 1437
 - callbacks 1438
 - methods 1438
 - security 1441
 - supported types 1438
- WebServiceConnector component
 - about 1449
 - common workflow 1450
 - events 1453
 - methods 1452
 - parameters 1450
 - properties 1452
- Window component
 - about 1465
 - creating applications 1467
 - customizing 1468
 - events 1475
 - inheritance 1472
 - methods 1473
 - package 1472
 - parameters 1466
 - properties 1474
 - using skins 1470
 - using styles 1469
- WSDLURL parameter 1450

X

- XML
 - attributes of menu item 887
 - formatting for the Tree component 1266
 - schema types 1438
- XMLConnector component
 - about 1489
 - common workflow 1490
 - events 1492
 - methods 1491
 - parameters 1489
 - properties 1492
 - schemas and 1489

XUpdateResolver component
 about 1507
 common workflow 1509
 events 1511
 parameters 1508
 properties 1511

