

---

# Flex 3 Cookbook™

*Joshua Noble and Todd Anderson*

**O'REILLY®**

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo



*Adobe Developer Library*, a copublishing partnership between O'Reilly Media Inc., and Adobe Systems, Inc., is the authoritative resource for developers using Adobe technologies. These comprehensive resources offer learning solutions to help developers create cutting-edge interactive web applications that can reach virtually anyone on any platform.

With top-quality books and innovative online resources covering the latest tools for rich-Internet application development, the *Adobe Developer Library* delivers expert training straight from the source. Topics include ActionScript, Adobe Flex®, Adobe Flash®, and Adobe Acrobat®.

Get the latest news about books, online resources, and more at <http://adobedeveloperlibrary.com>.

This excerpt is protected by copyright law. It is your responsibility to obtain permissions necessary for any proposed use of this material. Please direct your inquiries to [permissions@oreilly.com](mailto:permissions@oreilly.com).

---

# Table of Contents

<b>Preface</b> .....	<b>xvii</b>
<b>1. Flex and ActionScript Basics</b> .....	<b>1</b>
1.1 Create a Flex Project in Flex Builder	2
1.2 Create a Flex Library Project in Flex Builder	7
1.3 Create an ActionScript Project	9
1.4 Set Compiler Options for the MXML Compiler in Flex Builder	11
1.5 Compile a Flex Project Outside of Flex Builder	14
1.6 Add an Event Listener in MXML	16
1.7 Set Properties of a Child Defined in MXML in ActionScript	18
1.8 Define Arrays and Objects	20
1.9 Set the Scope of Variables in ActionScript	21
1.10 Create a Component in ActionScript	24
1.11 Use Event Bubbling	26
1.12 Use a Code-Behind Model to Separate MXML and ActionScript	28
1.13 Make Properties of a Component Bindable	29
1.14 Use Custom Events and Dispatch Data with Events	30
1.15 Listen for a Keyboard Event	32
1.16 Define Optional Parameters for Methods	33
1.17 Determine the Type of an Object	34
1.18 Define and Implement an Interface	35
<b>2. Menus and Controls</b> .....	<b>39</b>
2.1 Listen to a Button Click	39
2.2 Create a Set of Buttons That Toggle	42
2.3 Use a ColorPicker to Set Canvas Color	45
2.4 Load a SWF by Using the SWFLoader	46
2.5 Set Tab Indexes for Components	46
2.6 Set a labelFunction for a Control	47
2.7 Provide Data for Menus	48
2.8 Dynamically Populate Menus	50
2.9 Create EventHandlers for Menu-Based Controls	52

2.10	Display an Alert in an Application	53
2.11	Use the Date from a Calendar Control	55
2.12	Display and Position Multiple Pop-ups	56
2.13	Create a Custom Border for a Pop-up Window	59
2.14	Handle focusIn and focusOut Events	60
<b>3.</b>	<b>Containers .....</b>	<b>63</b>
3.1	Position Children by Using Layout Management	63
3.2	Position and Size Containers via Percentage Positioning	65
3.3	Track Mouse Position Within Different Coordinate Systems	66
3.4	Dynamically Add and Remove Children from a Container	67
3.5	Use Constraint-Based Layout for Containers	69
3.6	Set Maximum and Minimum Sizes for Children Within Containers	70
3.7	Specify Constraint Rows and Columns for a Container	71
3.8	Create Layout Flows for Text Using Constraints	73
3.9	Control Scrolling and Overflow Within Containers	75
3.10	Control the Layout of Box Components	77
3.11	Use Containers for Initialization	78
3.12	Create a TitleWindow	79
3.13	Control a ViewStack via a LinkBar	80
3.14	Bind the Selected Index of a ViewStack to a Variable	81
3.15	Use Delayed Instantiation to Improve Startup Time	83
3.16	Create and Control Resizable Containers	84
3.17	Create, Enable, and Disable TabControls Within a TabNavigator	85
3.18	Create a TabNavigator with Closeable Tabs	87
3.19	Create and Control an Alert	88
3.20	Size and Position a Dialog Box Based on Its Calling Component	90
3.21	Manage Multiple Pop-up Dialog Boxes	91
3.22	Scroll to a Specific Child in a Container	93
3.23	Create a Template Using IDDeferredInstance	94
3.24	Manually Lay Out a Container	97
3.25	Measure and Alter Container Size	101
3.26	Control the Visibility and Layout of Children	102
3.27	Create a Tile Container with Simple Reorganization	104
3.28	Set a Background Image and Rounded Corners in an HBox	106
3.29	Control Positioning and Scrolling of Child Components	107
<b>4.</b>	<b>Text .....</b>	<b>111</b>
4.1	Correctly Set the Value of a Text Object	111
4.2	Bind a Value to TextInput	113
4.3	Create a Suggestive TextInput	114

4.4	Create an In-Place Editor	115
4.5	Determine All Fonts Installed on a User's Computer	116
4.6	Create a Custom TextInput	118
4.7	Set the Style Properties for Text Ranges	119
4.8	Display Images and SWFs in HTML	120
4.9	Highlight User-Input Text in a Search Field	121
4.10	Manipulate Characters as Individual Graphics	122
4.11	Specify Styles for HTML in a TextField	126
4.12	Use the RichTextEditor	127
4.13	Apply Embedded Fonts with HTML	128
4.14	Add a Drop Shadow to Text in a Text Component	129
4.15	Find the Last Displayed Character in a TextArea	131
<b>5.</b>	<b>Lists, Tiles, and Trees</b> .....	<b>133</b>
5.1	Create an Editable List	133
5.2	Set Icons for Items in a List	135
5.3	Add Effects to a List to Indicate Changes	136
5.4	Set a Basic Item Renderer for a TileList	138
5.5	Set XML Data for a Tree	140
5.6	Create an Item Renderer for a Tree	141
5.7	Use Complex Data Objects in a Tree Control	143
5.8	Allow Only Certain Items in a List to Be Selectable	148
5.9	Format and Validate Data Added in a List's Item Editor	151
5.10	Track All Selected Children in a TileList	153
5.11	Use and Display Null Items in an Item Renderer	156
5.12	Create a Right-Click Menu for a List	157
5.13	Customize the Appearance of a Selection in a List	159
<b>6.</b>	<b>DataGrid and Advanced DataGrid</b> .....	<b>161</b>
6.1	Create Custom Columns for a DataGrid	161
6.2	Specify Sort Functions for DataGrid Columns	164
6.3	Enable Multicolumn Sorting in a DataGrid	166
6.4	Filter Items in a DataGrid	168
6.5	Create Custom Headers for an AdvancedDataGrid	170
6.6	Handle Events from a DataGrid/AdvancedDataGrid	173
6.7	Select Items in an AdvancedDataGrid	176
6.8	Enable Drag-and-Drop in a DataGrid	179
6.9	Edit Items in a DataGrid	180
6.10	Search Within a DataGrid and Autoscroll to the Match	182
6.11	Generate a Summary for Flat Data by Using GroupingCollection	184
6.12	Create an Async Refresh for a GroupingCollection	187

<b>7.</b>	<b>Renderers and Editors</b>	<b>193</b>
7.1	Create Your Own Renderers	193
7.2	Use the ClassFactory to Generate Renderers	196
7.3	Access the Component That Owns a Renderer	200
7.4	Create a Single Component to Act as Renderer and Editor	203
7.5	Create an Item Editor to Handle Data with Multiple Fields	205
7.6	Display SWF Objects as Items in a Menu by Using an Item Renderer	207
7.7	Select a DataGrid Column with a CheckBox Header Renderer	209
7.8	Create a Self-Contained CheckBox itemRenderer for Use in a DataGrid	212
7.9	Efficiently Set Images in a Renderer	214
7.10	Use Runtime Styling with itemRenderers and itemEditors	217
7.11	Use States and Transitions with an itemEditor	219
7.12	Create a CheckBox Tree Control	221
7.13	Resize Renderers Within a List	226
<b>8.</b>	<b>Images, Bitmaps, Videos, Sounds</b>	<b>229</b>
8.1	Load and Display an Image	230
8.2	Create a Video Display	231
8.3	Play and Pause an MP3 File	232
8.4	Create a Seek Bar for a Sound File	234
8.5	Blend Two Images	235
8.6	Apply a Convolution Filter to an Image	238
8.7	Send Video to an FMS Instance via a Camera	240
8.8	Access a User's Microphone and Create a Sound Display	242
8.9	Smooth Video Displayed in a Flex Application	244
8.10	Check Pixel-Level Collisions	245
8.11	Read and Save a User's Webcam Image	248
8.12	Use Blend Modes with Multiple Images	250
8.13	Handle Cue Points in FLV Data	251
8.14	Create a Video Scrubber	253
8.15	Read ID3 Data from an MP3 File	255
8.16	Display a Custom Loader while Loading Images	257
8.17	Enable Image Upload in Flex	258
8.18	Compare Two Bitmap Images	260
<b>9.</b>	<b>Skinning and Styling</b>	<b>263</b>
9.1	Use CSS to Style Components	264
9.2	Override the Default Application Style	266
9.3	Embed Styles by Using CSS	268
9.4	Override Base Style Properties	269
9.5	Customize Styles at Runtime	270

9.6	Load CSS at Runtime	272
9.7	Declare Styles at Runtime	274
9.8	Create Custom Style Properties for Components	276
9.9	Use Multiple Themes in the Same Application	279
9.10	Compile a Theme SWC	280
9.11	Use Embedded Fonts	283
9.12	Embed Fonts from a SWF File	285
9.13	Skin with Embedded Images	289
9.14	Apply Skins from a SWF File	291
9.15	Programmatically Skin a Component	295
9.16	Programmatically Skin a Stateful Control	299
9.17	Create Animated Skins from a SWF File	302
9.18	Customize the Preloader	306
<b>10.</b>	<b>Dragging and Dropping</b>	<b>313</b>
10.1	Use the DragManager Class	313
10.2	Specify a Drag Proxy	317
10.3	Drag and Drop Within a List	319
10.4	Drag and Drop Between Lists	322
10.5	Enable and Disable Drag Operations	323
10.6	Customize the DragImage of a List-Based Control	326
10.7	Customize the Drop Indicator of a List-Based Control	329
<b>11.</b>	<b>States</b>	<b>333</b>
11.1	Set Styles and Properties in a State	334
11.2	Create Transitions to Enter and Leave States	335
11.3	Use the addChildAction and removeChildAction	337
11.4	Filter Transitions to Affect Only Certain Types of Children	339
11.5	Apply Parts of a Transition to Certain Children	341
11.6	Base a State on Another State	344
11.7	Integrate View States with HistoryManagement	345
11.8	Use Deferred Instance Factories with States	347
11.9	Use Data Binding with Objects Added in a State	349
11.10	Add and Remove Event Listeners in State Changes	351
11.11	Add View States to a Flash Component	352
11.12	Work with State Change Events	355
11.13	Dynamically Generate and Use New States and Transitions	357
11.14	Create Custom Actions to Use in a State	358
<b>12.</b>	<b>Effects</b>	<b>361</b>
12.1	Call an Effect in MXML and in ActionScript	362
12.2	Build a Custom Effect	363
12.3	Create Parallel Series or Sequences of Effects	365

12.4	Pause, Reverse, and Restart an Effect	366
12.5	Create Custom Effect Triggers	367
12.6	Create Tween Effects	368
12.7	Use the DisplacementMapFilter Filter in a Flex Effect	371
12.8	Create an AnimateColor Effect	375
12.9	Use the Convolution Filter to Create a Tween	376
<b>13.</b>	<b>Collections</b>	<b>381</b>
13.1	Add, Sort, and Retrieve Data from an ArrayCollection	381
13.2	Filter an ArrayCollection	384
13.3	Determine When an Item Is Modified in an ArrayCollection	385
13.4	Create a GroupingCollection	386
13.5	Create a Hierarchical Data Provider for a Control	387
13.6	Navigate a Collection Object and Save Your Position	392
13.7	Create a HierarchicalViewCollection Object	394
13.8	Filter and Sort an XMLListCollection	396
13.9	Sort on Multiple Fields in a Collection	398
13.10	Sort on Dates in a Collection	399
13.11	Create a Deep Copy of an ArrayCollection	400
13.12	Use Data Objects with Unique IDs	402
<b>14.</b>	<b>Data Binding</b>	<b>405</b>
14.1	Bind to a Property	406
14.2	Bind to a Function	408
14.3	Create a Bidirectional Binding	411
14.4	Bind to Properties by Using ActionScript	411
14.5	Use Bindable Property Chains	415
14.6	Bind to Properties on XML by Using E4X	417
14.7	Create Customized Bindable Properties	419
14.8	Bind to a Generic Object	423
14.9	Bind to Properties on a Dynamic Class	425
<b>15.</b>	<b>Validation, Formatting, and Regular Expressions</b>	<b>431</b>
15.1	Use Validators and Formatters with TextInput and TextArea Controls	432
15.2	Create a Custom Formatter	434
15.3	Create a More-International Zip Code Validator by Using Regular Expressions	436
15.4	Create a Validator to Validate UPCs	438
15.5	Validate Combo Boxes and Groups of Radio Buttons	440
15.6	Show Validation Errors by Using ToolTips in a Form	443
15.7	Use Regular Expressions for Locating Email Addresses	446
15.8	Use Regular Expressions for Matching Credit Card Numbers	446

15.9	Use Regular Expressions for Validating ISBNs	447
15.10	Create Regular Expressions by Using Explicit Character Classes	447
15.11	Use Character Types in Regular Expressions	448
15.12	Match Valid IP Addresses by Using Subexpressions	450
15.13	Use Regular Expressions for Different Types of Matches	451
15.14	Match Ends or Beginnings of Lines with Regular Expressions	453
15.15	Use Back-References	453
15.16	Use a Look-Ahead or Look-Behind	455
<b>16.</b>	<b>Working with Services and Server-Side Communication</b>	<b>457</b>
16.1	Configure an HTTPService	458
16.2	Use RESTful Communication Between Flex Applications	460
16.3	Configure and Connect to a RemoteObject	461
16.4	Use Flex Remoting with AMFPHP 1.9	464
16.5	Use the IExternalizable Interface for Custom Serialization	468
16.6	Track Results from Multiple Simultaneous Service Calls	470
16.7	Use Publish/Subscribe Messaging	471
16.8	Register a Server-Side Data Type Within a Flex Application	472
16.9	Communicate with a WebService	474
16.10	Add a SOAP Header to a Request to a WebService	476
16.11	Parse a SOAP Response from a WebService	477
16.12	Communicate Securely with AMF by Using SecureAMFChannel	479
16.13	Send and Receive Binary Data via a Binary Socket	480
16.14	Communicate Using an XMLSocket	482
<b>17.</b>	<b>Browser Communication</b>	<b>483</b>
17.1	Link to an External URL	483
17.2	Work with FlashVars	484
17.3	Invoke JavaScript Functions from Flex	486
17.4	Invoke ActionScript Functions from JavaScript	487
17.5	Change the HTML Page Title via BrowserManager	489
17.6	Parse the URL via BrowserManager	490
17.7	Deep-Link to Data via BrowserManager	492
17.8	Deep-Link Containers via BrowserManager	493
17.9	Implement Custom History Management	496
<b>18.</b>	<b>Modules and Runtime Shared Libraries</b>	<b>499</b>
18.1	Create a Runtime Shared Library	500
18.2	Use Cross-Domain Runtime Shared Libraries	503
18.3	Use the Flex Framework as a Runtime Shared Library	505
18.4	Optimize a Runtime Shared Library	507
18.5	Create an MXML-Based Module	508

18.6	Create an ActionScript-Based Module	510
18.7	Load a Module by Using ModuleLoader	512
18.8	Use ModuleManager to Load Modules	514
18.9	Load Modules from Different Servers	516
18.10	Communicate with a Module	518
18.11	Pass Data to Modules by Using Query Strings	523
18.12	Optimize Modules by Using Linker Reports	526
<b>19.</b>	<b>The Adobe Integrated Runtime API .....</b>	<b>529</b>
19.1	Create an AIR Application Leveraging the Flex Framework	530
19.2	Understand the AIR Command-Line Tools	532
19.3	Open and Manage Native Windows	537
19.4	Create Native Menus	541
19.5	Read and Write to a File	544
19.6	Serialize Objects	546
19.7	Use the Encrypted Local Store	551
19.8	Browse for Files	553
19.9	Use the AIR File System Controls	555
19.10	Use the Native Drag-and-Drop API	558
19.11	Interact with the Operating System Clipboard	562
19.12	Add HTML Content	564
19.13	Cross-Script Between ActionScript and JavaScript	567
19.14	Work with Local SQL Databases	570
19.15	Detect and Monitor a Network Connection	575
19.16	Detect User Presence	577
19.17	Create System Tray and Dock Applications	578
<b>20.</b>	<b>Unit Testing with FlexUnit .....</b>	<b>581</b>
20.1	Create an Application That Uses the FlexUnit Framework	582
20.2	Create an Application to Run FlexUnit Tests	582
20.3	Create a FlexUnit Test Case	584
20.4	Add a Test Case to a Test Suite	587
20.5	Run Code Before and After Every Test	588
20.6	Share Test Data Between Test Cases	590
20.7	Handle Events in a Test Case	592
20.8	Test Visual Components with FlexUnit	595
20.9	Install and Configure Antennae	605
20.10	Generate Automated Test Suites	607
<b>21.</b>	<b>Compiling and Debugging .....</b>	<b>611</b>
21.1	Use Trace Statements Without Flex Builder	611
21.2	Use the Component Compiler	612
21.3	Install the Flex Ant Tasks	614

21.4	Use the compc and mxmhc Tasks in the Flex Ant Tasks	615
21.5	Compile and Deploy Flex Applications That Use RSLs	615
21.6	Create and Monitor Expressions in Flex Builder Debugging	618
21.7	Install the Ant View in the Stand-Alone Version of Flex Builder	620
21.8	Create an Ant Build File for Automating Common Tasks	621
21.9	Compile a Flex Application by Using mxmhc and Ant	622
21.10	Generate Documentation by Using ASDoc and Ant	624
21.11	Compile Flex Applications by Using Rake	625
21.12	Use ExpressInstall for Your Application	626
21.13	Use Memory Profiling with Flex Builder 3 to View Memory Snapshots	628
<b>22.</b>	<b>Configuration, Internationalization, and Printing</b>	<b>631</b>
22.1	Add an International Character Set to an Application	631
22.2	Use a Resource Bundle to Localize an Application	633
22.3	Use the ResourceManager for Localization	637
22.4	Use Resource Modules for Localization	639
22.5	Support IME Devices	642
22.6	Detect a Screen Reader	644
22.7	Create a Tabbing Reading Order for Accessibility	645
22.8	Print Selected Items in an Application	646
22.9	Format Application Content for Printing	647
22.10	Control Printing of Unknown Length Content over Multiple Pages	649
22.11	Add a Header and a Footer When Printing	650
<b>Index</b>		<b>655</b>

---

# Compiling and Debugging

Compiling Flex applications is most often done through Flex Builder or through invoking the MXML compiler (mxmmlc) on the command line, but there are many other tools that let you compile an application, move files, or invoke applications. Tools such as make, Ant, or Rake, for example, enable you to simplify an entire compilation and deployment routine so that you can invoke it from a single command.

Debugging in Flex is done through the debug version of the Flash Player, which enables you to see the results of `trace` statements. With Flex Builder 3, you can step through code line by line and inspect the properties of variables. Flex Builder 3 also introduces a new profiling view that lets you examine memory usage and the creation and deletion of objects. Outside of Flex Builder, numerous open source tools expand your options. With Xray and Console.as for Firebug, for example, you can inspect the values of objects, or you can view the output of `trace` statements with FlashTracer or the Output Panel utility instead of using the Flex Builder IDE. The recipes in this chapter cover debugging with both the tools provided in Flex Builder as well as tracing values and inspecting objects by using Xray and FlashTracer.

## 21.1 Use Trace Statements Without Flex Builder

### Problem

You want to create `trace` statements that will assist you in debugging your application, but you do not have Flex Builder 3.

### Solution

Download and use one of the many open source tracing tools available.

### Discussion

Since Adobe made the Flex 3 library and compiler freely available, developers have gained more options for viewing `trace` statements output by the Flash Player. No longer

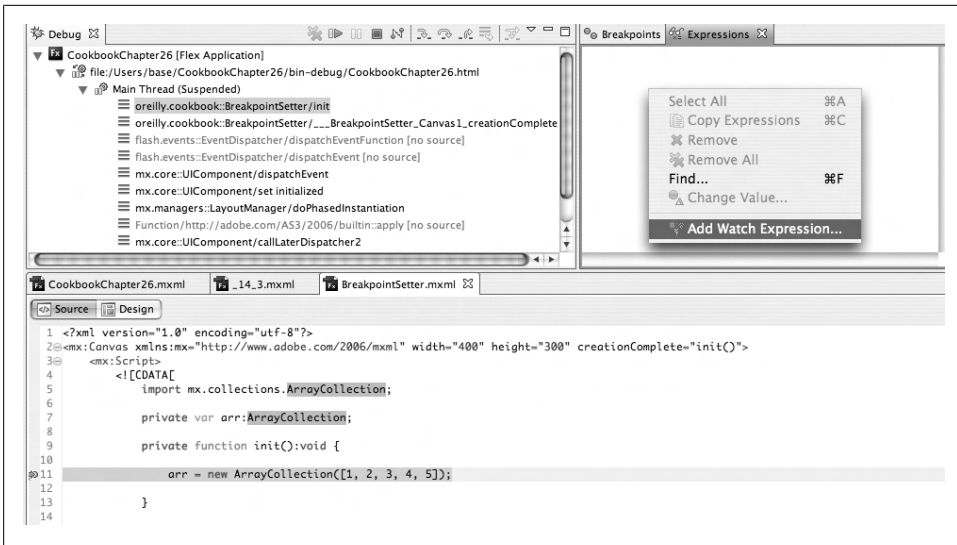


Figure 21-1. Viewing output with Xray

are you limited to using the Flash IDE or Flex Builder IDE; now you can choose from several tools. For example, Xray (developed by John Grden) creates `trace` statement viewers within Flash. Xray allows for not only the viewing of `trace` statements during application execution, but also the basic inspection of objects during execution (Figure 21-1).

A third option is the FlashTrace utility (developed by Alessandro Crugnola). Installing this plug-in in the Firefox browser enables you to receive any `trace` statements that are executed within the application. If you like, you can also log the results of the trace to a file.

You can download Xray from <http://osflash.org/xray#downloads> and FlashTrace from <http://www.sephiroth.it/firefox>.

## 21.2 Use the Component Compiler

### Problem

You want to compile a Flex component into a SWC file that can be used as a runtime shared library (RSL).

### Solution

Use the Component compiler (`compc`) and either pass command-line arguments to the compiler or pass a configuration XML file as the `load-config` argument.

## Discussion

To invoke the Component compiler, `compc`, use this syntax:

```
compc -source-path . -include-classes oreilly.cookbook.foo -output example.swc
```

Some of the most important options for the `compc` are as follows:

- benchmark**  
Indicates that the compiler should benchmark the amount of time needed to compile the SWC.
- compiler.debug**  
Indicates whether the generated SWC should have debugging information and functionality included with it.
- compiler.external-library-path [path-element] [...]**  
Indicates SWC files or directories to compile against but to omit from linking.
- compiler.include-libraries [library] [...]**  
Indicates libraries (SWCs) to completely include in the SWF.
- compiler.library-path [path-element] [...]**  
Indicates SWC files or directories that contain SWC files that should be used in compiling.
- compiler.locale [locale-element] [...]**  
Specifies the locale for internationalization.
- compiler.optimize**  
Enables postlink SWF optimization.
- compiler.services <filename>**  
Specifies the path to the Flex Data Services configuration file.
- compiler.theme [filename] [...]**  
Lists all CSS or SWC files to apply as themes within the application.
- compiler.use-resource-bundle-metadata**  
Determines whether resources bundles are included in the application.
- include-classes [class] [...]**  
Indicates all the classes that should be included in the RSL; can be repeated multiple times or have a wildcard path listed.
- include-file <name><path>**  
Indicates all the files that should be included in the RSL; can be repeated multiple times or have a wildcard path listed.
- include-resource-bundles [bundle] [...]**  
Sets whether a localization resource bundle should be included.
- load-config <filename>**  
Loads a file containing configuration options.

`-output <filename>`

Determines the name and location of the file that is generated by `compc`.

`-runtime-shared-libraries [url] [...]`

Indicates any external RSLs that should be bundled into the RSL generated by `compc` in this compilation.

`-runtime-shared-library-path [path-element] [rsl-url] [policy-file-url] [rsl-url] [policy-file-url]`

Sets the location and other information about an RSL that the application will use.

`-use-network`

Toggles whether the SWC is flagged for access to network resources.

Compiling many classes into a runtime shared library can result in a very long command. To simplify this, you can use either configuration files or manifest files.

As with the MXML compiler (`mxmlc`), you can use configuration files with `compc` by specifying a `load-config` option. Also like `mxmlc`, `compc` automatically loads a default configuration file called `flex-config.xml`. Unless you want to duplicate the entire contents of `flex-config.xml` (much of which is required), specify a configuration file in addition to the default by using the `+=` operator:

```
compc -load-config+=configuration.xml
```

Any flags passed to the compiler can be described in XML and passed to `compc` in the `-load-config` option:

```
<include-sources>src/./</include-sources>
```

## 21.3 Install the Flex Ant Tasks

### Problem

You want to use the Flex Ant tasks included with the Flex 3 SDK.

### Solution

Copy the `flex_ant/lib/flexTasks.jar` file to Ant's lib directory (`{ANT_root}/lib`).

### Discussion

To ensure that Ant always has access to all tasks included in the Flex Ant tasks library provided with the Flex 3 SDK, you must copy the tasks into the lib directory of the Ant installation. If you do not copy this file to the lib directory, you must specify it by using Ant's `-lib` option on the command line when you make a project XML file.

## 21.4 Use the `compc` and `mxm1c` Tasks in the Flex Ant Tasks

### Problem

You want to use the `mxm1c` or `compc` tasks that are included with the Flex Ant tasks to simplify compiling Flex applications and working with Ant.

### Solution

Install the Flex Ant tasks into your Ant libraries and then use either the `<mxm1c>` or `<compc>` tags, with the compile options passed to the tags as XML arguments.

### Discussion

The Flex Ant tasks greatly simplify working with Ant for compiling Flex applications by providing prebuilt common tasks for developers to use. All the options you can set for the command-line use of `mxm1c` or `compc` can be passed to the Flex Ant task. For example, after you declare the `mxm1c` task, you can declare the file output options as shown:

```
<mxm1c file="C:/Flex/projects/app/App.mxml" output="C:/Flex/projects/bin/App.swf">
```

Instead of needing to specify the location of `mxm1c` and set all the options as arguments to the executable, the `mxm1c` Ant task can be used, saving configuration time and making your build files far easier to read. Further options can be set as shown here:

```
<!-- Get default compiler options. -->
<load-config filename="${FLEX_HOME}/frameworks/flex-config.xml"/>
<!-- List of path elements that form the roots of ActionScript class hierarchies.
-->
<source-path path-element="${FLEX_HOME}/frameworks"/>
<!-- List of SWC files or directories that contain SWC files. -->
<compiler.library-path dir="${FLEX_HOME}/frameworks" append="true">
  <include name="libs" />
  <include name="./bundles/{locale}" />
</compiler.library-path>
</mxm1c>
```

The `<compc>` task for the Flex Ant tasks works similarly; all of the options for `compc` are passed through to the `<compc>` task:

```
<compc output="${output}/mylib.swc" locale="en_US">
```

## 21.5 Compile and Deploy Flex Applications That Use RSLs

### Problem

You need to deploy a Flex application that uses one or more runtime shared libraries (RSLs).

## Solution

Use the `external-library-path` compiler option to indicate the location of the RSL or RSLs after the application is compiled.

## Discussion

When it initializes, a Flex application needs to know the location of any necessary runtime shared libraries. The `external-library-path` compiler option contains this information; passing it to the compiler enables the Flash Player to begin loading the bytes for the RSL right away, without needing to load a separate SWF file before instantiating components or classes.

In order to use an RSL file, you need to first create an RSL. RSLs are stored within SWC files that are then accessed by the application at runtime. The SWC RSL file is compiled by using `compc`, and the application SWF file is compiled by using the `mxmlc` compiler. In order for the application to use the RSL, a reference to the location of the RSL must be passed to `mxmlc` by using the `runtime-shared-libraries` option. In this example, Ant is used to compile both the SWC file and the application that will access it, meaning that we'll need to use both `compc` and `mxmlc`. In the `build.xml` file that Ant will use, both of the compilers will need to be declared as variables:

```
<property name="mxmlc" value="C:\FlexSDK\bin\mxmlc.exe"/>
<property name="compc" value="C:\FlexSDK\bin\compc.exe"/>
```

Next, use `compc` to compile the RSL that the application will access and use the `move` task to place it in the `application/rsl` directory:

```
<target name="compileRSL">
  <exec executable="${compc}">
    <arg line="-load-config+=rsl/configuration.xml" />
  </exec>
  <mkdir dir="application/rsl" />
  <move file="example.swc" todir="application/rsl" />
  <unzip src="application/rsl/example.swc" dest="application/rsl/" />
</target>
```

Then compile the application SWF by using `mxmlc`. Note that we're passing an XML file called `configuration.xml` to the compiler by using `-load-config`. This file will contain all the information about how we want our application compiled, including, in this case, the location of the RSL:

```
<target name="compileApplication">
  <exec executable="${mxmlc}">
    <arg line="-load-config+=application/configuration.xml" />
  </exec>
</target>

<target name="compileAll" depends="compileRSL,compileApplication">
</target>
```

Note that both actual command-line calls to the compilers use a configuration.xml file containing information about the location of the runtime shared libraries that will be passed to mxmclc:

```
<flex-config>
  <compiler>
    <external-library-path>
      <path-element>example.swc</path-element>
    </external-library-path>
  </compiler>
  <file-specs>
    <path-element>RSLClientTest.mxml</path-element>
  </file-specs>
  <runtime-shared-libraries>
    <url>example.swf</url>
  </runtime-shared-libraries>
</flex-config>
```

In place of adding the external-library-path flag to the command-line invocation of mxmclc as shown here

```
mxmclc -external-library-path=example.swc
```

the configuration.xml file is passed as the load-config flag in the call to the compiler, and each option is read from the XML file.

A similar file can be passed to compc:

```
<flex-config>
  <compiler>
    <source-path>
      <path-element>.</path-element>
    </source-path>
  </compiler>
  <output>example.swc</output>
  <include-classes>
    <class>oreilly.cookbook.shared.*</class>
  </include-classes>
</flex-config>
```

The complete Ant file for this recipe is shown here:

```
<?xml version="1.0"?>
<project name="useRSL" basedir=".">

  <property name="mxmclc" value="C:\FlexSDK\bin\mxmclc.exe"/>
  <property name="compc" value="C:\FlexSDK\bin\compc.exe"/>

  <target name="compileRSL">
    <exec executable="${compc}">
      <arg line="-load-config+=rsl/configuration.xml" />
    </exec>
    <mkdir dir="application/rsl" />
    <move file="example.swc" todir="application/rsl" />
    <unzip src="application/rsl/example.swc" dest="application/rsl/" />
  </target>
```

```
<target name="compileApplication">
  <exec executable="{mxmlc}">
    <arg line="-load-config+=application/configuration.xml" />
  </exec>
</target>

<target name="compileAll" depends="compileRSL,compileApplication">
</target>

</project>
```

## 21.6 Create and Monitor Expressions in Flex Builder Debugging

### Problem

You want to track the changes to a value in your Flex application as the application executes.

### Solution

Use the Flex Builder Debugger to run your application and set a breakpoint where the variable that you would like to inspect is within scope. In the Expressions window of the Flex Builder Debugger, create a new expression.

### Discussion

The use of expressions is a powerful debugging tool that lets you see the value of any variable within scope. Any object within the scope where the breakpoint is set can be evaluated by creating an expression, as shown in Figure 21-2.

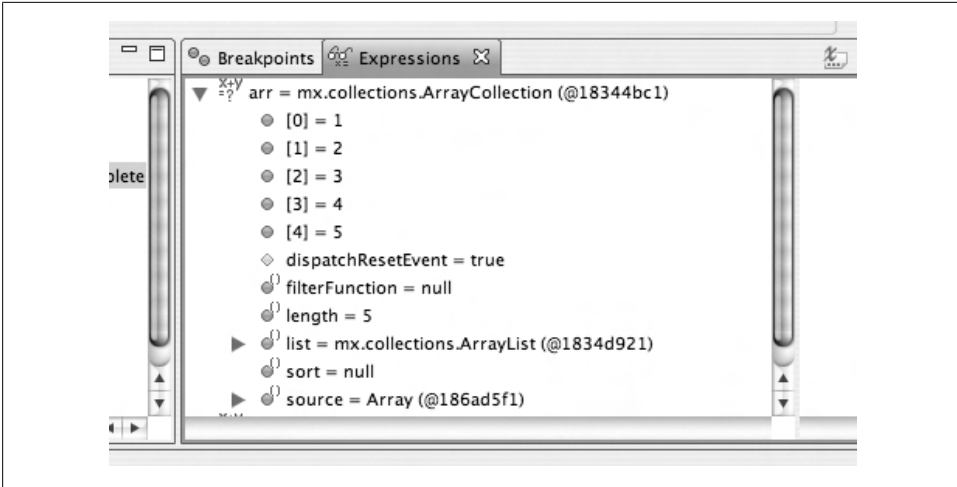


Figure 21-2. Creating an expression

For example, if you place a breakpoint at the line where the array is instantiated, marked here with *breakpoint here*

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300" creationComplete="init()">
  <mx:Script>
    <![CDATA[
      import mx.collections.ArrayCollection;

      private var arr:ArrayCollection;

      private function init():void {
        arr = new ArrayCollection([1, 2, 3, 4, 5]); //breakpoint here
      }

      private function newFunc():void {
        var newArr:ArrayCollection = new ArrayCollection([3, 4, 5, 6]);
      }

    ]]>
  </mx:Script>
</mx:Canvas>
```

the expression `arr` will evaluate to `null`. When you advance the application by pressing the F6 key, the expression will evaluate to an `ArrayCollection` wrapping an `Array` of five integers (Figure 21-3).

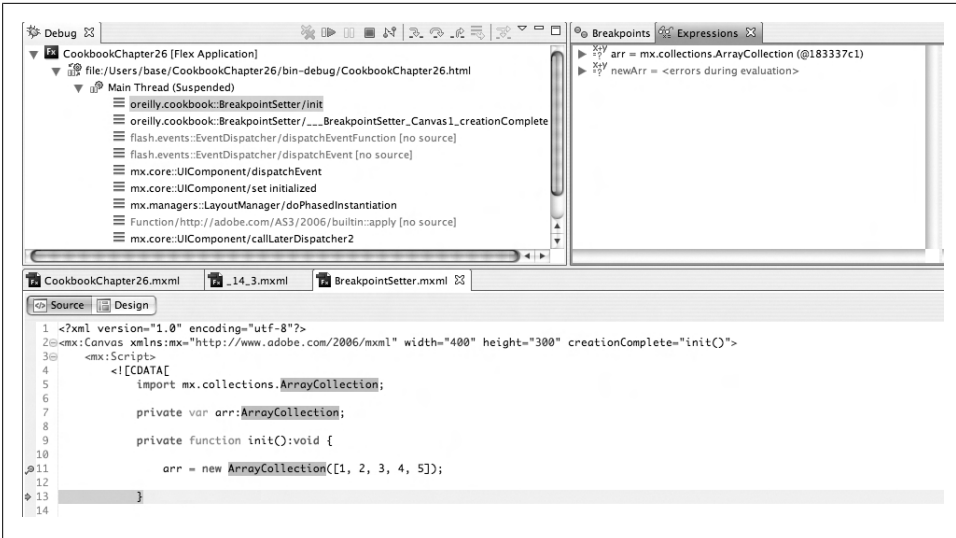


Figure 21-3. The expression showing the variable evaluated

The expression `newArr` evaluates to `null`, however, because the variable `newArr` will not be in scope (Figure 21-4).

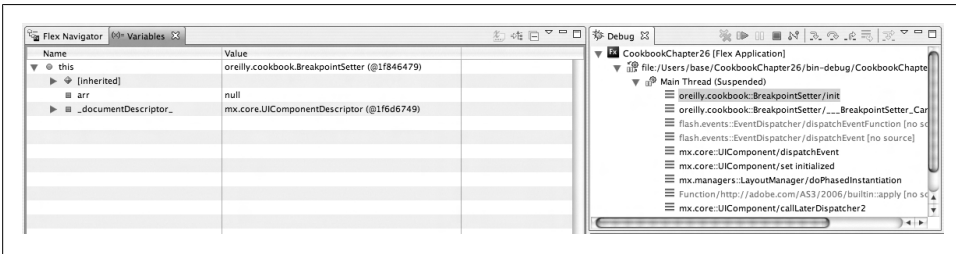


Figure 21-4. Only variables in scope can be evaluated.

If you instead place a breakpoint at line 17, the expressions `newArr` and `arr` both evaluate to `ArrayCollections`, because both variables will be in the current scope.

## 21.7 Install the Ant View in the Stand-Alone Version of Flex Builder

Contributed by Ryan Taylor

### Problem

You can't find the Ant view in the stand-alone version of Flex Builder.

## Solution

Install the Eclipse Java Development Tools.

## Discussion

To access Ant in Flex Builder's stand-alone version, you must install the Eclipse Java Development Tools. To do so:

1. In the Flex Builder menu bar, choose Help → Software Updates → Find and Install.
2. Select the Search for New Features to Install option and then click Next.
3. Choose The Eclipse Project Updates in the dialog box and then click Finish.
4. A menu appears, asking you to select a location from which to download the files. Select any location, preferably one that is geographically near you for faster download times, and then click OK.
5. Browse the various SDK versions in the Eclipse Project Updates tree until you find Eclipse Java Development Tools. Select the check box next to it and then click Next.
6. After the Update Manager finishes downloading the necessary files, you will be prompted with a feature verification dialog box. Click Install All.
7. After installation is completed, restart Flex Builder.

You can now find the Ant view in Flex Builder by browsing to Window → Other Views → Ant.

## 21.8 Create an Ant Build File for Automating Common Tasks

Contributed by Ryan Taylor

### Problem

You want to leverage the capabilities of Ant to help automate common tasks such as compiling and generating documentation.

### Solution

Create an Ant build file in which tasks can be added for automating your processes.

### Discussion

Creating an Ant build file is easy and the first step toward using Ant to automate common tasks. Simply create a new XML document named `build.xml` and save it in a directory named `build` in the root of your project directory. Saving the file in this directory is not mandatory, but a common convention.

The root node in your build file should look something like this:

```
<project name="MyAntTasks" basedir="..">
</project>
```

You will want to set the `name` attribute to something unique for your project. This is the name that will show up inside the Ant view in Eclipse. For the `basedir` attribute, make sure it is set to the root of your project directory. You will use the `basedir` property frequently when defining other properties that point toward files and directories inside your project folder.

Next, you will likely want to create some additional properties for use throughout the various tasks that you may add later. For instance, to create a property that points toward your project's source folder, you could do something like this:

```
<project name="MyAntTasks" basedir="..">
  <property name="src" value="${basedir}/src" />
</project>
```

The preceding example also demonstrates how to use a property after it has been defined, with the syntax `${property}`.

If you find that you are defining a lot of properties and you would like to keep your build file as clean as possible, you can declare properties in a separate file instead. To do this, create a new text file named `build.properties` and save it in the same directory as your `build.xml` file. Inside this file, declaring properties is as simple as this:

```
src="${basedir}/src"
```

That's all there is to it. Some examples of useful properties to define are paths to your source folder(s), your bin folder, and the Flex 3 SDK directory. You'll catch on pretty quickly to what you need. From here, you are ready to start adding tasks to your build file.

## See Also

Recipe 21.3.

## 21.9 Compile a Flex Application by Using mxmhc and Ant

Contributed by Ryan Taylor

### Problem

You want to add tasks to your Ant build file for compiling your application.

### Solution

Add executable tasks to your Ant build file that use the MXML compiler to compile your files.

## Discussion

Compiling targets are by far the most common and useful types of targets you will add to your Ant build files. Flex applications are compiled by using mxmcl, which is the free command-line compiler included with the Flex 3 SDK. By adding targets for compiling to your build file, you can automate the build process: Ant will compile all your files without you ever having to open up the command prompt or terminal.

The MXML compiler (mxmcl) included in multiple formats. You can use the executable version of it by creating a target similar to this:

```
<!-- COMPILE MAIN -->
<target name="compileMain" description="Compiles the main application files.">
  <echo>Compiling '${bin.dir}/main.swf'...</echo>
  <exec executable="${FLEX_HOME}/bin/mxmcl.exe" spawn="false">
    <arg line="-source-path '${src.dir}'" />
    <arg line="-library-path '${FLEX_HOME}/frameworks'" />
    <arg line="'${src.dir}/main.mxml'" />
    <arg line="-output '${bin.dir}/main.swf'" />
  </exec>
</target>
```

Alternatively, you can use the Java version by writing a task such as this one:

```
<!-- COMPILE MAIN -->
<target name="compileMain" description="Compiles the main application files.">
  <echo>Compiling '${bin.dir}/main.swf'...</echo>
  <java jar="${FLEX_HOME}/lib/mxmcl.jar" fork="true" failonerror="true">
    <arg value="+flexlib=${FLEX_HOME}/frameworks" />
    <arg value="-file-specs='${src.dir}/main.mxml'" />
    <arg value="-output='${bin.dir}/main.swf'" />
  </java>
</target>
```

The final (and perhaps best) approach is to use the optional mxmcl tasks that are included with the Flex 3 SDK. Installing these is described in Recipe 21.3. To access them in your build file, first you will need to add a task definition:

```
<!-- TASK DEFINITIONS -->
<taskdef resource="flexTasks.tasks" classpath="${FLEX_HOME}/ant/lib/flexTasks.jar" />
```

By importing the optional Flex tasks, you can now compile by using an even more intuitive syntax, plus leverage error detection in a tool such as Eclipse as you write out the task. For example:

```
<!-- COMPILE MAIN -->
<target name="compileMain" description="Compiles the main application files.">
  <echo>Compiling '${bin.dir}/main.swf'...</echo>
  <mxmcl file="${src.dir}/main.mxml" output="${bin.dir}/main.swf">
    <source-path path-element="${src.dir}" />
  </mxmcl>
</target>
```

In all of these examples, the same basic rules apply. You need to define properties that point toward your project's src and bin directories, as well as the Flex 3 SDK. All of the

properties in the examples use suggested names, except for `FLEX_HOME`, which is a mandatory name. The `FLEX_HOME` property *must* be set to the root of the Flex 3 SDK before using the `mxmcl` task. If you're using the EXE or JAR versions of `mxmcl`, you can use a property name other than `FLEX_HOME`.

The true power of compiling your project via Ant lies in the ability to chain targets together. For instance, you could create a `compileAll` target that calls each individual compile target one by one:

```
<!-- COMPILE ALL -->
<target name="compileAll" description="Compiles all application files." depends="compileMain, compileNavigation, compileGallery, compileLibrary">
  <echo>Finishing compile process...</echo>
</target>
```

All of this may seem a little intimidating at first; however, after you spend a little bit of time using Ant and configuration files, you will find that they can greatly improve your workflow. By letting a third-party tool such as Ant automate your compile process, you are no longer tied to using one particular development tool. You will easily be able to call on Ant to build your project from the development tool of your choice, that is, Flex Builder, FDT, TextMate, or FlashDevelop.

## See Also

Recipe 21.3.

# 21.10 Generate Documentation by Using ASDoc and Ant

Contributed by Ryan Taylor

## Problem

You want to easily generate documentation for your application.

## Solution

Add an executable task to your Ant build file that uses ASDoc (included with the Flex 3 SDK) to generate the documentation for you.

## Discussion

ASDoc is a free command-line utility that is included with the Flex 3 SDK. If you have ever used Adobe's LiveDocs, you are already familiar with the style of documentation that ASDoc produces. Though opening up the command prompt or terminal and using it isn't terribly difficult, a better solution is to add a target to your Ant build file for automating the process even further.

Before creating a target for generating your documentation, it is a good idea to create an additional target that cleans out your docs directory. When you define the `docs.dir` property, simply point it toward your project's docs directory:

```
<!-- CLEAN DOCS -->
<target name="cleanDocs" description="Cleans out the documentation directory.">
  <echo>Cleaning '${docs.dir}'...</echo>
  <delete includeemptydirs="true">
    <fileset dir="${docs.dir}" includes="**/*" />
  </delete>
</target>
```

With the target for cleaning out the docs directory in place, you are ready to create the target that actually generates the documentation. Notice in the sample code that the `depends` attribute mandates that the `cleanDocs` target is executed before the instructions for generating the documentation:

```
<!-- GENERATE DOCUMENTATION -->
<target name="generateDocs" description="Generates application documentation using ASDoc." depends="cleanDocs">
  <echo>Generating documentation...</echo>
  <exec executable="${FLEX_HOME}/bin/asdoc.exe" failOnError="true">
    <arg line="-source-path ${src.dir}" />
    <arg line="-doc-sources ${src.dir}" />
    <arg line="-main-title ${docs.title}" />
    <arg line="-window-title ${docs.title}" />
    <arg line="-footer ${docs.footer}" />
    <arg line="-output ${docs.dir}" />
  </exec>
</target>
```

The `FLEX_HOME` property needs to point toward the root directory of the Flex 3 SDK on your machine. The `src.dir` and `docs.dir` properties represent your project's `src` and `docs` directories, respectively. Last but not least are the `docs.title` and `docs.footer` properties, which set the title and footer text that appears in the documentation. A common convention for the documentation title is *Your Project Reference*, where *Your Project* is the name of the project you are working on. The footer is a good place to put a copyright and URL.

ASDoc will successfully generate documentation from your code even if you haven't written a single comment. It is, of course, highly recommended that you thoroughly document your code by using Javadoc commenting. Not only will this produce much more in-depth documentation, but programmers unfamiliar with your code can also follow along inside the code itself.

## 21.11 Compile Flex Applications by Using Rake

### Problem

You want to compile Flex applications by using Rake, the Ruby make tool.

## Solution

Download and install Ruby 1.9 if you have not already, and then download and install Rake.

## Discussion

Although written completely in Ruby, Rake functions very similarly to the classic make utility used by C++ and C programmers. After you've downloaded and installed both Ruby and Rake, you can write a simple Rake file like so:

```
task :default do
  DEV_ROOT = "/Users/base/flex_development"
  PUBLIC = "#{DEV_ROOT}/bin"
  FLEX_ROOT = "#{DEV_ROOT}/src"
  system "/Developer/SDKs/Flex/bin/mxmlc --show-actionscript-warnings=true --strict=true
-file-specs #{FLEX_ROOT}/App.mxml"
  system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
end
```

All tasks in Rake are similar to targets in Ant, that is, they define an action to be done. The default action is always performed, and any extra actions can be optionally called within a different task. Within the task itself, variables can be declared, and system arguments can be called, as shown here:

```
system "/Developer/SDKs/Flex/bin/mxmlc --show-actionscript-warnings=true --strict=true
-file-specs #{FLEX_ROOT}/App.mxml"
```

This is the actual call to the MXML compiler that will generate the SWF file. Because an item in the Rake task won't be run until the previous task returns, the next line can assume that the SWF has been generated and can be copied to a new location:

```
system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
```

The rest of the Rake file declares variables that will be used to place files in the appropriate folders. The file can now be saved with any name and run at the command line by using the rake command. If you save the file as Rakefile, you can now run it by entering the following:

```
rake Rakefile
```

## 21.12 Use ExpressInstall for Your Application

### Problem

You want to ensure that if a user does not have the correct version of Flash Player installed to view a Flex application, the correct version can be installed.

## Solution

Use the `ExpressInstall` option when compiling to let the SWF file redirect the user to the Adobe website where the most current version of Flash Player can be installed.

## Discussion

To use the Express Install, you can set the Use Express Install option in Flex Builder, in the application options (Figure 21-5).

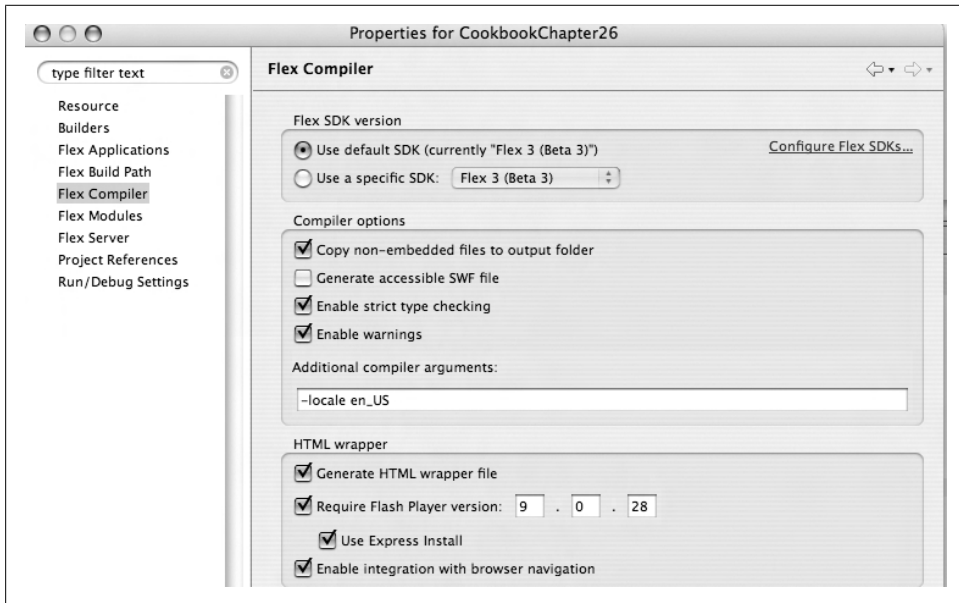


Figure 21-5. Setting the Express Install option

If you are not using Flex Builder for development, then simply set the `pluginspage` variable in the `Object` tag on the `embed` tag of the HTML page that the SWF file is embedded within the `ExpressInstall`:

```
pluginspage="http://www.adobe.com/go/getflashplayer"
```

A sample `<embed>` statement for a Netscape-based browser such as Firefox is shown here:

```
<embed src="CookbookChapter26.swf" id="CookbookChapter26" quality="high" bgcolor="#869c  
a7" name="CookbookChapter26" allowscriptaccess="sameDomain" pluginspage="http:  
//www.adobe.com/go/getflashplayer" type="application/x-shockwave-flash" align="middle"  
height="100%" width="100%">
```

## 21.13 Use Memory Profiling with Flex Builder 3 to View Memory Snapshots

### Problem

You want to view all the objects allocated in the Flash Player's memory at runtime.

### Solution

Use the Memory Profiler view in Flex Builder 3 to run your application and observe the objects being created and destroyed.

### Discussion

The Flex Profiler is a new addition to Flex Builder 3 and is a powerful tool that enables you to watch an application as it allocates and clears memory and objects. It connects to your application with a local socket connection. You might have to disable antivirus software to use it, however, if your antivirus software prevents socket communication.

As the Profiler runs, it takes a snapshot of data every few milliseconds and records the state of the Flash Player at that snapshot, a process referred to as *sampling*. By parsing the data from sampling, the Profiler can show every operation in your application. The Profiler records the execution time of those operations, as well as the total memory usage of objects in the Flash Player at the time of the snapshot. When an application is run in the Profiler, you'll see the Connection Established dialog box (Figure 21-6). Here you can enable memory profiling to help identify areas of an application where memory allocation problems are occurring, as well as enable performance profiling to help improve the performance of an application.

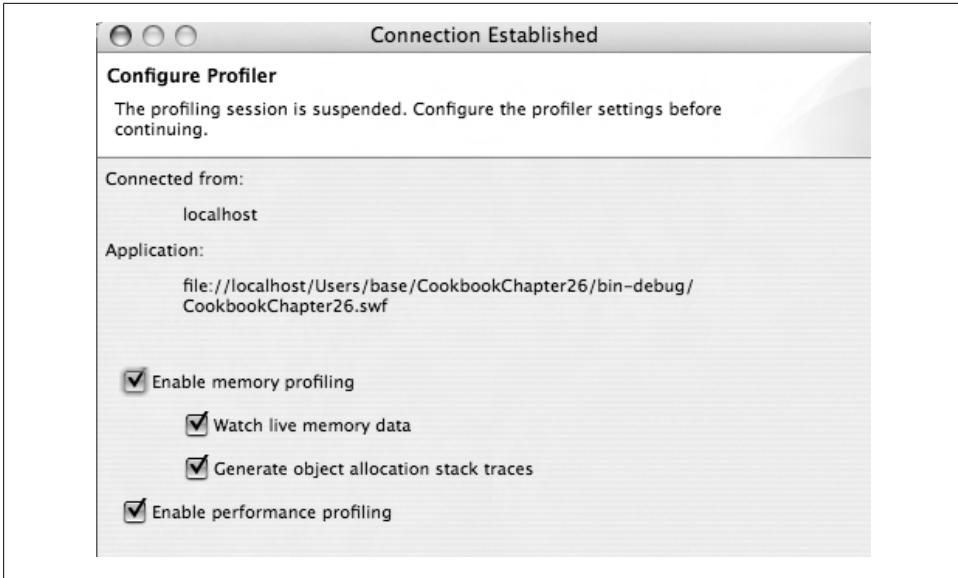


Figure 21-6. Selecting a profiling type

If you turn on the Watch Live Memory Data check box, the Profiling view displays live graphs of the objects allocated in the Flash Player (Figure 21-7).

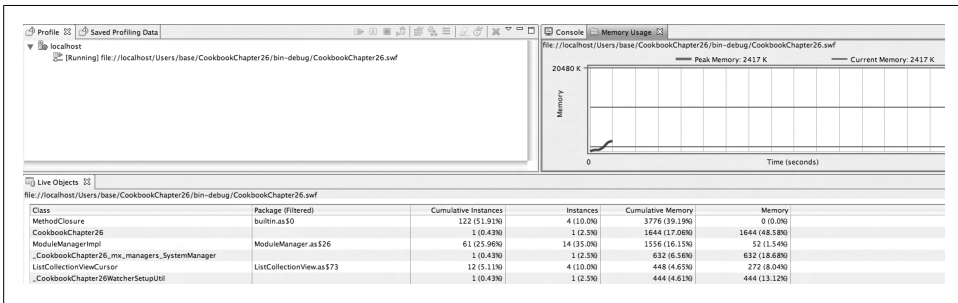


Figure 21-7. Live object and memory allocation data in the Flex Profiling view

The Profiler provides memory snapshots that can be taken at any time and provide in-depth data about the number of instances of any object and the amount of memory that they require (Figure 21-8).

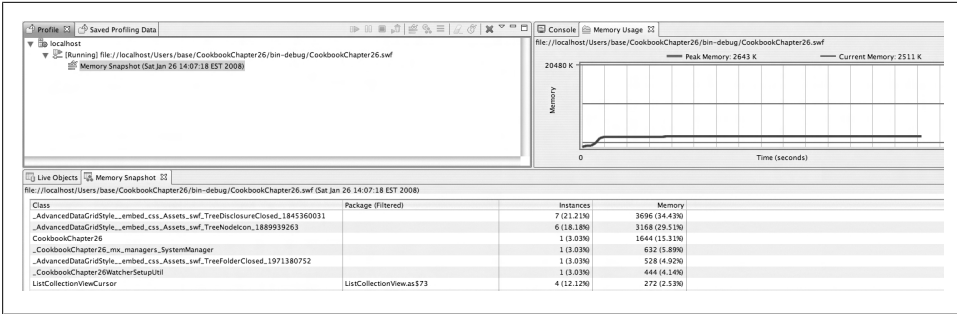


Figure 21-8. Viewing the number of instances and memory consumption in a memory snapshot

Finally, you can compare any two memory snapshots from different times in the application to find *loitering objects*, that is, objects that were created after the first memory snapshot and exist in the second. Information about the class name, memory size, and number of instances are all included in the Loitering Objects view (Figure 21-9).

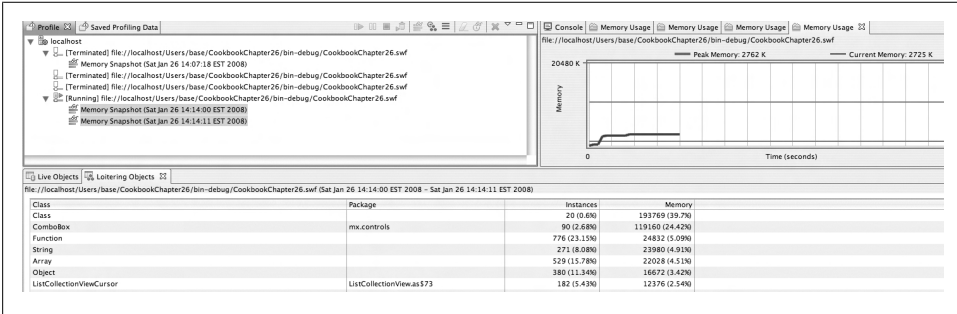


Figure 21-9. Viewing all generated objects in the Loitering Objects screen