

# Using ADOBE® FLASH® BUILDER™ 4.6

## **Legal Notices**

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

# Contents

## Chapter 1: About Flash Builder

Applications you can build with Flash Builder .....	1
Features that accelerate application development .....	2
Features for running and debugging projects .....	2
Flash Builder versions .....	3
Flash Builder configurations .....	3
Adobe Community Help Client (CHC) .....	4

## Chapter 2: Getting Started With Flash Builder

Flash Builder workbench .....	5
Flash Builder perspectives and views .....	6
Flash Builder editors .....	18
Projects in Flash Builder .....	30
Project resources .....	37

## Chapter 3: Code Development Tools in Flash Builder

Content Assist .....	44
Quick Assist .....	47
Override or implement methods .....	51
Code templates .....	52
Metadata code completion .....	55
Customize file templates .....	57
Generate from usage .....	64
Generate get and set accessor functions .....	66
Syntax error checking .....	67
Unidentified reference error highlighting .....	68
Find references and refactor code .....	69
Format, navigate, and organize code .....	70

## Chapter 4: Using Projects in Flash Builder

Create projects in Flash Builder .....	81
Export and import projects .....	88
Build projects .....	93
Run and debug applications .....	110
Export a release version of an application .....	117
Package AIR applications .....	121

## Chapter 5: Debugging Tools in Flash Builder

The Flash Debug perspective .....	124
How to debug your application .....	127

## Chapter 6: Profiling Tools in Flash Builder

The Flash Profiling perspective .....	133
Profiling an application .....	146

How the profiler works .....	147
Use the profiler .....	149
Garbage collection .....	154
Identify problem areas .....	155
Profiler filters .....	158

## **Chapter 7: Unit Testing Tools in Flash Builder**

FlexUnit test environment .....	160
Create FlexUnit tests .....	160
Run FlexUnit tests .....	162
Configure FlexUnit tests .....	164
View results of a FlexUnit test run .....	164
FlexUnit support for mobile projects .....	165

## **Chapter 8: Developing Applications with Flex**

Basic workflow to develop an application with Flex .....	167
Build user interfaces .....	168
Generate event handlers .....	198
Access data services .....	200
Monitor applications that access data services .....	208
Use Flex library projects .....	212
Create custom MXML components .....	216
Create modules .....	218
Integrate Flex with HTML applications .....	223

## **Chapter 9: Using Flash Builder with Flash Professional**

Create a Flash Professional project .....	228
What you can do in a Flash Professional project .....	228
Set project properties for Flash Professional projects .....	229
Create and edit Flash components .....	229
Manage Flash Player security .....	231

## **Chapter 10: Using Flash Builder with Flash Catalyst**

Workflows between Flash Catalyst and Flash Builder .....	233
Flash Catalyst and Flash Builder integration .....	237

## **Chapter 11: Customizing Flash Builder**

Adobe preferences .....	241
Customize the workbench .....	241
Flash Builder preferences .....	243
Extending Flash Builder .....	252



# Chapter 1: About Flash Builder

Adobe® Flash® Builder™ is an integrated development environment (IDE) for building cross-platform, rich Internet applications (RIAs) for the desktop and a wide variety of mobile devices. Flash Builder also includes testing, debugging, and profiling tools that lead to increased levels of productivity and effectiveness.

Flash Builder is built on top of Eclipse, an open-source IDE, and provides all the tools required to develop applications that use the Flex framework and ActionScript 3.0. Flash Builder runs on Microsoft Windows and Apple Mac OS X, and is available in several versions. Installation configuration options let you install Flash Builder as a set of plug-ins in an existing Eclipse workbench installation or create a stand-alone installation that includes the Eclipse workbench.

## Applications you can build with Flash Builder

You can build applications that use the Adobe Flex® framework, MXML, Adobe Flash Player, Adobe AIR, ActionScript 3.0, and ADEP (Adobe Digital Enterprise Platform) Data Services, and the Adobe® Flex® Charting components:

- **Flex projects** - Create Flex projects that can work with any backend server technology, including Adobe ColdFusion®, ADEP Data Services, and PHP.

See “[Flex projects](#)” on page 81 and “[Developing Applications with Flex](#)” on page 167.

- **ActionScript projects** - Create ActionScript projects that use the Flash API (not the Flex framework).

See “[ActionScript projects](#)” on page 84.

- **Mobile projects** - Create Flex mobile applications for the Google Android platform and ActionScript mobile applications for the Apple iOS platform.

See “[Flex mobile projects](#)” on page 83 and “[Create ActionScript mobile projects](#)” on page 84.

For more information on developing mobile applications with Flex and Flash Builder, see [Developing Mobile Applications with Flex and Flash Builder](#).

- **Library projects** - Create custom code libraries that you can deploy as component library (SWC) files to share among your applications or distribute to other developers.

See “[Use Flex library projects](#)” on page 212.

- **Flash Professional projects** - Create Flash Professional projects to edit and debug Flash FLA or XFL files created with Flash Professional CS5.

See “[Using Flash Builder with Flash Professional](#)” on page 228.

- **Flash Catalyst compatible projects** - Create Flash Catalyst compatible projects to let designers and developers collaborate and work on the same project. You can easily share a Flash compatible project between Flash Builder and Flash Catalyst through FXP or FXPL files.

See “[Flash Catalyst compatible projects](#)” on page 85.

Read about the workflows between Flash Builder and Flash Catalyst in the article, [Understanding Flash Catalyst and Flash Builder workflows](#) by Adobe Product Manager, Jacob Surber.

- **Custom MXML components** - Create custom components and then easily access them using the Flash Builder Components view.

See [“Create custom MXML components”](#) on page 216.

You can also create custom ActionScript components. See [“Create an ActionScript class”](#) on page 38.

## Features that accelerate application development

Flash Builder has the tools you need to develop applications that use the Flex framework and ActionScript 3.0. You can:

- **Access data services** - Use Flash Builder tools and wizards to create applications that access remote data services.  
See [Accessing data services overview](#).
- **Customize the Flash Builder workbench** - Customize the workbench to suit your individual development needs. For example, you can arrange the interface to include your favorite tools in a specific layout.  
See [“Customize the workbench”](#) on page 241.
- **Edit code** - Write and edit your application source code using the code development tools in Flash Builder. The code development tools include code refactoring, code hinting, streamlined code navigation, and automatic syntax error checking, and other features.  
See [“Code Development Tools in Flash Builder”](#) on page 44.
- **Build a user interface** - Use the MXML editor in Design mode to design applications using layout options, to drag components onto the design canvas and then reposition and resize them as needed, to simplify using view states, and so on.  
See [“Build user interfaces”](#) on page 168.
- **Publish source code** - Publish your application source code so that users and other developers can view it .  
See [“Publish source code”](#) on page 109.
- **Manage projects, folders, files, and other resources** - Create, modify, and delete projects and resources, link to resources outside your project, and so on.  
See [“Projects in Flash Builder”](#) on page 30 and [“Project resources”](#) on page 37.

## Features for running and debugging projects

Flash Builder includes building, testing, debugging, and profiling tools that help increase your productivity:

- **Build projects** - Flash Builder automatically compiles and builds your applications for debugging or production release. You can also create custom build scripts using Apache Ant.  
See [“Build projects”](#) on page 93.
- **Run applications and manage launch configurations** - Run your applications in a web browser, AIR, or the stand-alone Flash Player. Create custom launch configurations to control how your applications run.  
See [“Run and debug applications”](#) on page 110 and [“Manage launch configurations”](#) on page 111.
- **Debug applications** - Debug your applications using the integrated debugging tools.  
See [“Debugging Tools in Flash Builder”](#) on page 124.
- **Run and debug mobile applications** - Run and debug your applications on the desktop or on a device.  
See [Run and debug mobile applications](#).

- **Profile applications** - Identify performance bottlenecks and memory leaks by using the profiling tools in Flash Builder.  
See [“Profiling Tools in Flash Builder”](#) on page 133.
- **Monitor applications that access data services** - Use the Network Monitor to generate a detailed audit trail of all data passed between your application and the back end .  
See [“Monitor applications that access data services”](#) on page 208.

## Flash Builder versions

Flash Builder is available in two versions: Standard and Premium.

**Flash Builder Standard** Flash Builder Standard provides a full-featured IDE that allows you to create applications using the Flex framework and Flash API. Flash Builder Standard also includes MXML, ActionScript, and CSS editors, as well as debugging tools. Flash Builder Standard provides a library of interactive charts and graphs that enable you to create rich data dashboards, interactive data analysis, and data visualization components.

**Flash Builder Premium** In addition to the Standard version features, Flash Builder Premium includes memory and performance profiling and automated testing tools. Use the Network Monitor to view the data that flows between a client application and a data service. The FlexUnit test environment allows you to generate and edit repeatable tests. The repeatable tests can be run from scripts or directly within Flash Builder or outside the Flash Builder environment. The Command Line Build feature allows you to synchronize a developer’s individual build settings with a nightly build.

**Flash Builder for PHP Standard** Flash Builder for PHP Standard provides a full-featured PHP IDE that includes Flash Builder Standard integrated with Zend Studio 8. You can build mobile, web, and desktop applications using PHP, Flex, and ActionScript.

**Flash Builder for PHP Premium** In addition to Standard version capabilities, Flash® Builder™ for PHP Premium includes professional testing tools, including profilers, network monitoring, an automated testing framework, integration with FlexUnit testing, and Command Line Build support.

For more information on Flash Builder for PHP, see [Introducing Flash Builder for PHP](#).

## Flash Builder configurations

Flash Builder provides a single downloadable installer with the following two configuration options:

**Stand-alone configuration** Installs Flash Builder as a stand-alone integrated development environment (IDE). The stand-alone configuration is created specifically for developing applications that use the Flex framework and ActionScript 3.0. This configuration is ideal for new users and users who intend to develop applications using only the Flex framework and ActionScript 3.0.

**Plug-in configuration** Configures Flash Builder to run as a plug-in within an existing Eclipse™ installation. You must have Eclipse 3.6 32 bit (Windows) or Eclipse 3.6 Cocoa 32 bit (Mac) or a higher version of Eclipse installed on your computer to run the plug-in configuration.

The Flash Builder plug-in and stand-alone configurations provide the same functionality. If you aren't sure which configuration to use, follow these guidelines:

- If you already have Eclipse 3.6 or a higher version installed, use the plug-in configuration to add Flash Builder features to your existing copy of Eclipse.
- If you don't have Eclipse installed and your primary focus is on developing Flex and ActionScript applications, use the Flash Builder stand-alone configuration. This configuration also allows you to install other Eclipse plug-ins to expand the scope of your development work in the future.

For detailed information about installing Flash Builder, see the [Flash Builder 4.6 release notes](#).

## Adobe Community Help Client (CHC)

Flash Builder uses the Adobe Community Help Client (CHC) as the search engine. The CHC is an AIR-based application that replaces the Eclipse Help engine for Flash Builder and is the platform for the next generation of Adobe Help delivery. CHC features include:

- **Always online**

If you have a network connection, the CHC accesses content from the web. The CHC ensures that you access the most up-to-date-material. It can also work in local mode if there is no Internet connection.

- **Search-centric**

Use Community Help search, adobe.com search, or local search. Community Help search aggregates resources, including resources from third-party sites. adobe.com search includes refinements to narrow your scope.

- **In-context navigation**

Provides a dynamically generated set of related links for key pages.

- **Rating and commenting**

Comment on, rate, and contribute to Help content and Adobe online resources from within the CHC. You need a valid Adobe ID to provide any feedback and rating.

# Chapter 2: Getting Started With Flash Builder

## Flash Builder workbench

The Flash Builder workbench is a full-featured development environment to assist you in developing applications that use the Adobe Flex® framework, and ActionScript 3.0. You can develop applications for deployment on Adobe Flash Player, desktop applications for deployment on Adobe AIR®, and mobile applications for deployment on various mobile devices.

The information required to use Flash Builder is contained in the Flash Builder documentation. If you are using other Eclipse plug-ins (such as CVS or Java) with Flash Builder, or you want to extend the Flash Builder plug-ins, see [Adobe Flash Builder Extensibility Reference](#).

**Workbench** The term *workbench* refers to the Flash Builder development environment that contains all the tools to develop applications. The workbench contains three primary elements: *perspectives*, *editors*, and *views*. You use all three in various combinations at various points in the application development process.

**Note:** For more information about some of the Eclipse workbench features, see the *Eclipse Workbench User's Guide* at <http://help.eclipse.org/help31/index.jsp>.

**Perspective** A perspective is a group of views and editors in the workbench. Flash Builder contains two perspectives: The Flash Development perspective for developing applications, and the Flash Debug perspective for debugging applications. Flash Builder Premium also contains the Flash Profiling perspective.

If you use the Flash Builder plug-in configuration (see “[Flash Builder configurations](#)” on page 3), your workbench can contain additional perspectives such as a Java perspective that contains editors and views used to develop Java applications.

See “[Flash Builder perspectives and views](#)” on page 6.

**Editor** An editor lets you edit various file types. The editors available to you depend on the number and type of Eclipse plug-ins installed. Flash Builder contains editors for writing MXML, ActionScript 3.0, and Cascading Style Sheets (CSS) code.

See “[Flash Builder editors](#)” on page 18 and “[Code Development Tools in Flash Builder](#)” on page 44.

**Views** A view typically supports an editor. For example, when editing MXML, the Components and Flex Properties views are also displayed in the Flash Development perspective. These views support the development of applications and are therefore displayed when an MXML file is opened for editing.

See “[Work with views](#)” on page 17.

The term *view* is synonymous with the term *panel* as it is used in earlier versions of Flash Builder, Adobe Dreamweaver®, and other Adobe development tools.

**Workspace** Not to be confused with *workbench*, a *workspace* is a defined area of the file system that contains the resources (files and folders) that make up your application projects. You can work with only one workspace at a time; however, you can select a different workspace each time you start Flash Builder.

See “[Projects in Flash Builder](#)” on page 30.

**Resource** The term *resource* is used generically to refer to the files and folders within the projects in a workspace.

See “[Project resources](#)” on page 37.

**Project** All of the resources that make up your applications are contained within projects. You cannot build an application in Flash Builder without first creating a project. Flash Builder supports various types of projects, depending on the type of application you are building.

See “[Types of projects](#)” on page 31 and “[Using Projects in Flash Builder](#)” on page 81.

**Launch configuration** A *launch configuration* is created for each of your projects, and it defines project settings that are used when running and debugging your applications. For example, the names and locations of the compiled application SWF files are contained in the launch configuration, and you can modify these settings.

See “[Manage launch configurations](#)” on page 111.

### More Help topics

“[Customize the workbench](#)” on page 241

## Flash Builder perspectives and views

To support a particular task or group of tasks, editors and supporting views are combined into a *perspective*. When you open a file that is associated with a particular perspective, Flash Builder automatically opens that perspective.

The stand-alone configuration of Flash Builder contains three perspectives:

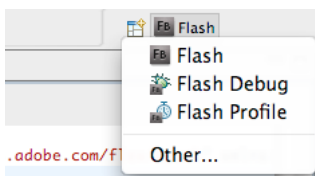
- Flash Development  
See “[The Flash Development perspective](#)” on page 7.
- Flash Debugging  
See “[The Flash Debug perspective](#)” on page 124.
- Flash Profiling  
See “[The Flash Profiling perspective](#)” on page 133.

**Note:** The Flash Profiling perspective is available with Flash Builder Premium.

### Open and switch perspectives

Perspectives change automatically to support the task at hand. For example, when you create a Flex project, the workbench switches to the Development perspective. When you start a debugging session, the Flash Debug perspective is displayed when the first breakpoint is encountered.

You can also manually switch perspectives yourself by selecting Window > Open Perspective. Or, you can use the *perspective bar*, which is located in the main workbench toolbar.



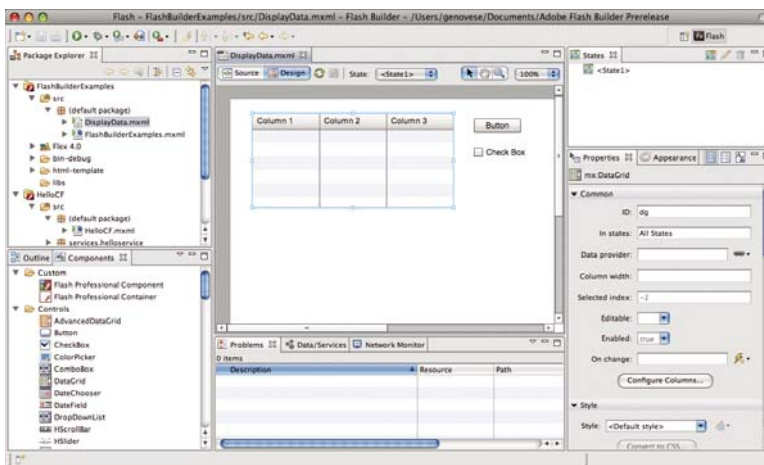
When the perspective opens, its title changes to display the name of the perspective you selected. An icon appears next to the title, allowing you to quickly switch back and forth between perspectives in the same window. By default, perspectives open in the same window.

If you use the plug-in configuration of Flash Builder and have other Eclipse plug-ins installed, you can have additional perspectives. While predefined perspectives are delivered with each Eclipse plug-in, you can customize them or create your own. To see a complete list of perspectives, select Window > Open Perspective > Other.

## The Flash Development perspective

The Flash Development perspective includes the editors and views you require to create applications for the Flex framework. When you create a project, Flash Builder switches into the Development perspective so you can begin developing your application.

The following example shows the Package Explorer, Outline, and Problems views:



The focal point of the perspective (and the workbench generally) is the editor area. The editor area contains all of the currently open documents in a multitab interface.

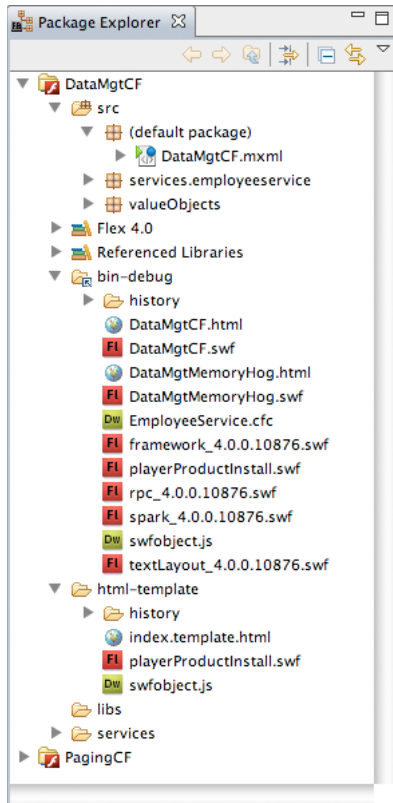
When you create a Flex project, the main MXML application file is opened in the editor area. You can then open and switch between any of the MXML, ActionScript, and CSS documents you are working in. For more information, see [“Flash Builder editors”](#) on page 18.

## Flash Development perspective in Source mode

In Source (code editing) mode, the Development perspective contains the editors and the following supporting views:

## Package Explorer view

The Package Explorer view contains all of the projects and resources in the workspace and is therefore an essential element of the Flash Builder workbench. It is always displayed in the Development and Debug perspectives.



When you work in the Package Explorer, you can select a resource and view its properties.

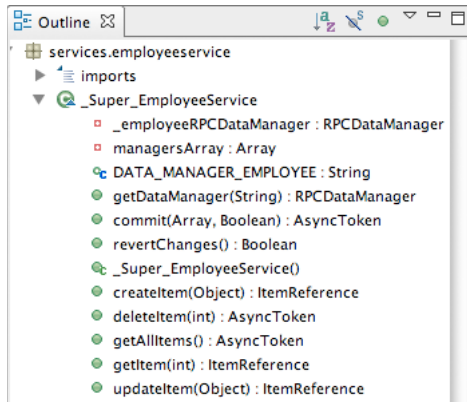
- 1 In the Package Explorer, select a resource.
- 2 Select File > Properties.

For more information about the Package Explorer and working with projects, see [“Manage projects”](#) on page 32.



## Outline view

In Source mode, the Outline view presents a hierarchical view of the code structure of the selected MXML or ActionScript document so that you can inspect and navigate the sections or lines of code in the document. The Outline view also displays syntax error alerts that the compiler generates. This view is also available when you use the ActionScript editor.

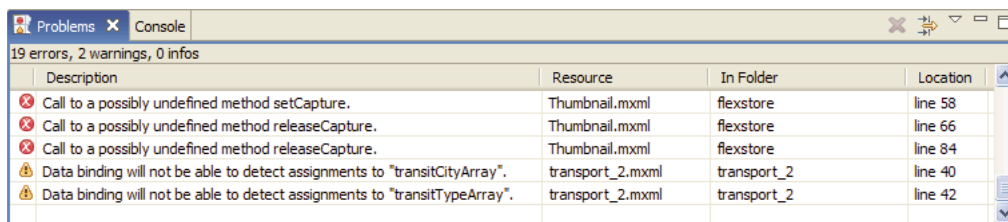


For more information about using the Outline view in Source mode, see [“Use the Outline view”](#) on page 73.

## Problems view

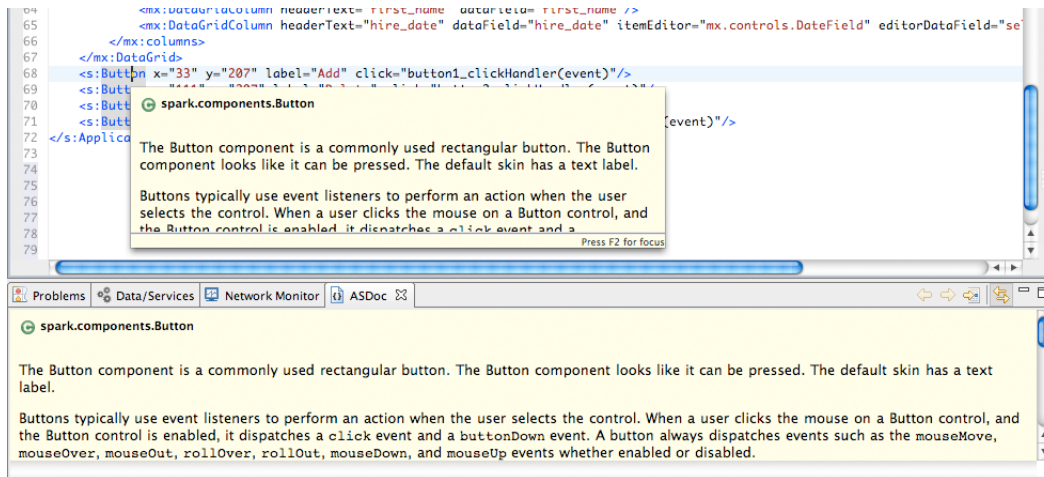
As you enter code, the Flash Builder compiler detects syntax and other compilation errors, and these errors are displayed in the Problems view. The Package Explorer marks nodes that contain errors.

When you debug your applications, errors, warnings, and other information are displayed in the Problems view. Each error or warning contains a message, the file and folder in which it is located, and its line number in the file. Problems remain in the Problems view until you correct them or they are otherwise resolved.



**Note:** You can also optionally add the Tasks and Bookmarks views. These views provide additional shortcuts for managing and navigating your code. For more information about these views, see [“Use markers”](#) on page 77. For an introduction to the optional views that are available in Flash Builder, see [“Other useful workbench views”](#) on page 15.

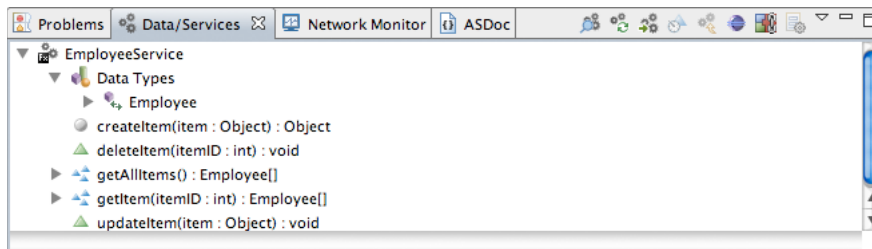
## ASDoc view



As you enter code, or hover over code elements, Flash Builder displays ASDoc content relevant to the code. Flash Builder also displays the ASDoc content for the selected code in the ASDoc view.

For more information, see [“Display ActionScript reference summary”](#) on page 47.

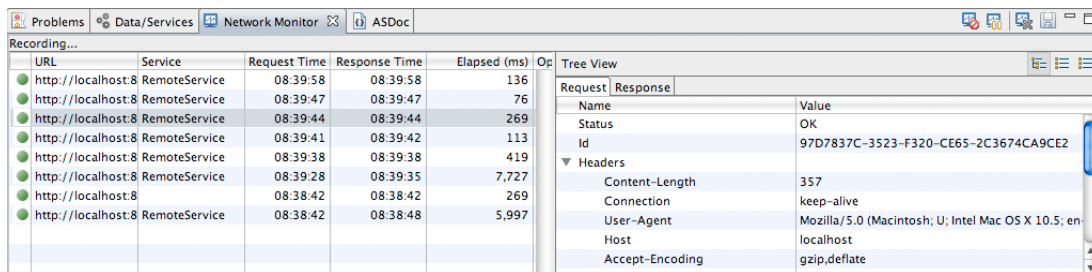
## Data/Services view



Flash Builder provides wizards and tools to connect to data services. Use the Data/Services view to connect to data services. Once you connect to a data service, the Data/Services view displays the service, data types for returned data, and available operations for the service. Use this view to configure access to the service and bind returned data to user interface components.

For more information, see [Building data-centric applications with Flash Builder](#).

## Network Monitor view

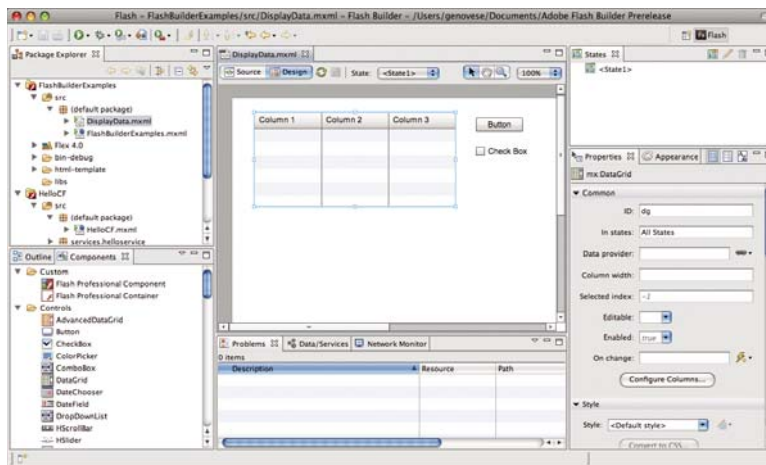


The Network Monitor allows you to examine the data that flows between an application and the data service or services. the Network Monitor is available with Flash Builder Premium.

For more information, see [“Monitor applications that access data services”](#) on page 208.

## Flash Development perspective in Design mode

The Design mode of the MXML editor is the visual representation of the code that you edit in Source mode. In Design mode, however, additional views are added to support design tasks. These views are the Components, Properties, Appearance, States, and Design Mode Problems view.



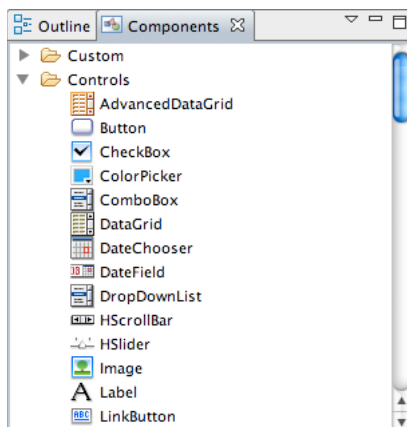
In addition, when you are in Design mode, the Outline view displays the MXML structure of your applications. For more information about designing applications in Flash Builder, see [“Build user interfaces”](#) on page 168.

**Note:** Design mode is not available when working with ActionScript projects. To preview the user interface of your ActionScript applications, build and run the applications. For more information about working with ActionScript, see [“ActionScript Projects”](#) on page 31 and [“Run and debug applications”](#) on page 110.

In Design mode, the development perspective contains the MXML editor and the following views:

### Components view

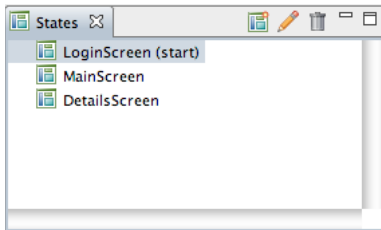
The Components view contains all of the standard Flex components, which you can select and add to the design area. As you create your own custom components, they are also displayed in the Components view.



For more information, see [“Create and edit Flash components”](#) on page 229.

### States view

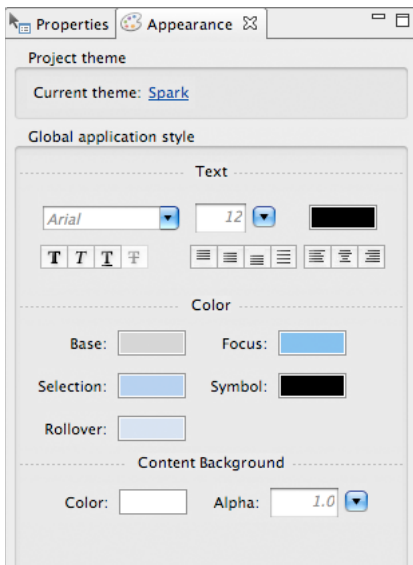
Flex allows you to create applications that change their appearance based on events that are triggered directly by the user or events that are generated programmatically. These user interface transformations are referred to as *view states*. You create and manage view states in the States view.



For more information about view states, see [“Add View states and transitions”](#) on page 182.

### Appearance view

The Appearance view allows you to set global application styles for text and colors. You can also specify the theme for applications in the project.

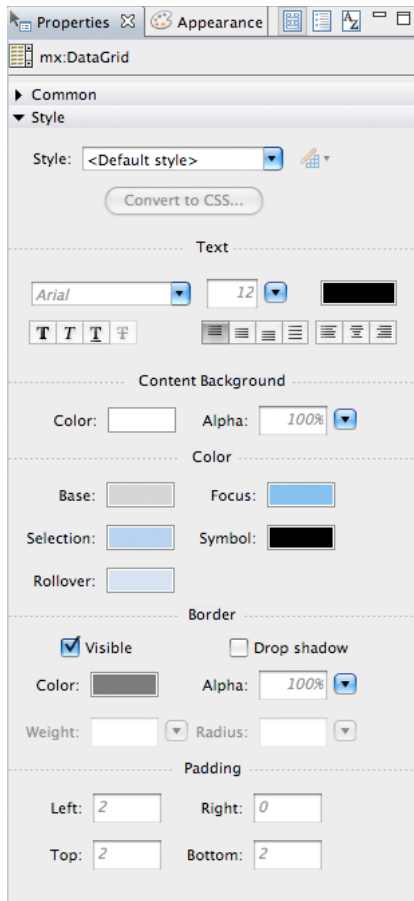


*Appearance panel*

For more information, see [“Apply styles to an application”](#) on page 177.

### Styles view

The Styles view allows you to set styles for specific components. The styles you can set vary, depending on the component.

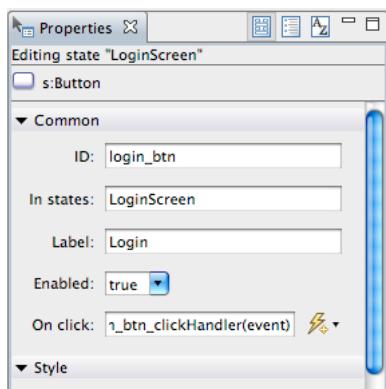


Style panel

For more information, see [“Apply styles to an application”](#) on page 177.

### Flex Properties view

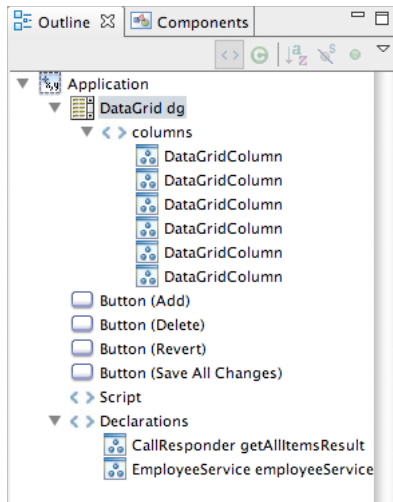
When a Flex component is selected, its properties are displayed in the Properties view. You can set and edit properties as appropriate. You can view a component’s properties graphically (as shown in the following example) and as a categorized or alphabetical list.



For more information, see [“Set component properties”](#) on page 24.

## Outline view

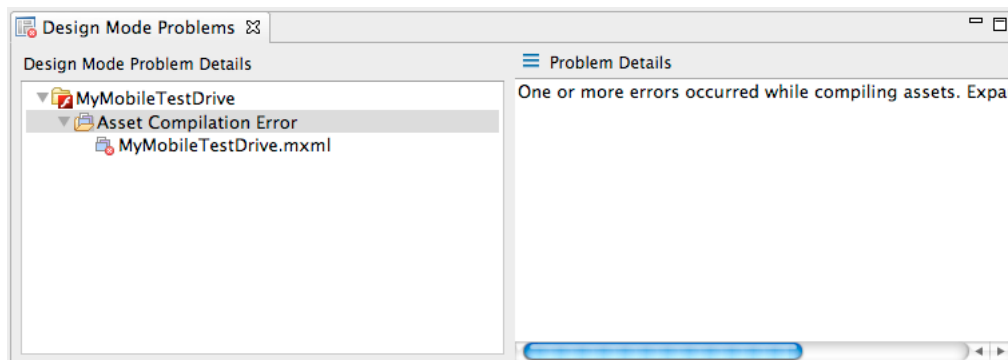
In Design mode, the Outline view presents a hierarchical view of the MXML structure of your applications. You can easily navigate the structure of an application by selecting individual MXML tag statements and components. When you select an item in the Outline view, it is highlighted in Design mode.



For more information about working with the Outline view in Design mode, see [“Inspect the structure of your MXML”](#) on page 27.

## Design Mode Problems view

When the Design View initializes, if any problems are detected with the editor or project, the problems are displayed in the Design Mode Problems view.



The Design Mode Problem Details panel displays the list of problems related to the initialized editor's Design mode. The problems are displayed in a tree structure format.

When you select a problem in the Design Mode Problem Details panel, details of the problem are displayed in the Problem Details panel. When you close a project, the problems related to that project are removed from the Design Mode Problems view.

**Note:** The Design Mode Problems view is displayed only when Design mode is enabled. If you disable Design mode (Deselect Window > Enable Design Mode), the Design Mode Problems view does not appear.

## Other useful workbench views

In addition to the editors and views associated with Flash Builder's default development, debugging, and profiling perspectives, the workbench contains other views. These optional views are categorized by type and are associated with distinct workbench functionality or with specific Eclipse plug-ins.

To access views that are not already displayed with a perspective and add them to the workbench, select **Window > Show View > Other**.

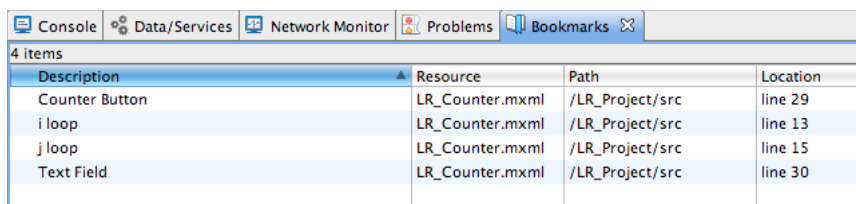
These views include the Tasks, Bookmarks, and Search views that help you streamline the application development process.

After you add a view to the current perspective, you can save that view as part of the perspective. For more information, see [“Customize a perspective”](#) on page 16.

You can also create fast views to provide quick access to views that you use often. For more information, see [“Create and work with fast views”](#) on page 17.

## Bookmarks view

The Bookmarks view is used to manage the bookmarks that you add to specific lines of code or to resources. As in a web browser, bookmarks are used as a convenience for tracking noteworthy items. Selecting a bookmark locates and displays it in the workbench.



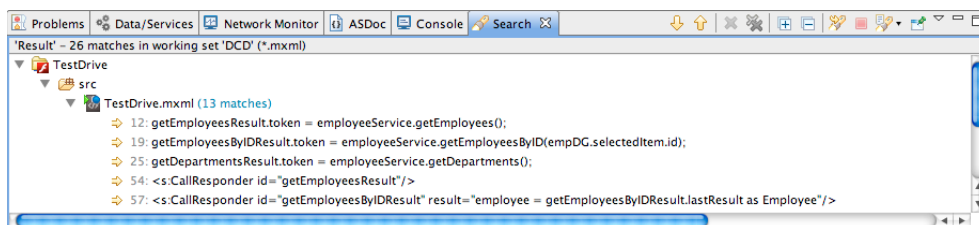
Description	Resource	Path	Location
Counter Button	LR_Counter.mxml	/LR_Project/src	line 29
i loop	LR_Counter.mxml	/LR_Project/src	line 13
j loop	LR_Counter.mxml	/LR_Project/src	line 15
Text Field	LR_Counter.mxml	/LR_Project/src	line 30

## More Help topics

[“Use markers”](#) on page 77

## Search view

The Search view is displayed automatically when you search the resources in the workspace. You can use it to define and recall previous searches and to filter the list of search results.



## More Help topics

[“Search for resources in the workbench”](#) on page 39

## Work with perspectives

### Set a default perspective

The word *default* in parentheses following the perspective name indicates the default perspective.

- 1 Open the Preferences dialog and select General > Perspectives.
- 2 Under Available Perspectives, select the perspective to define as the default, and click Make Default.
- 3 Click OK.

### Open a perspective in a new window

You can specify to open perspectives in a new window.

- 1 Open the Preferences dialog and select General > Perspectives.
- 2 Under Open a New Perspective, select In A New Window.  
To switch back to the default, select In The Same Window.
- 3 Click OK.

### Customize a perspective

To modify a perspective's layout, you change the editors and views that are visible in a given perspective. For example, you could have the Bookmarks view visible in one perspective, and hidden in another perspective.

To create a perspective, do the following:

- 1 Open an existing perspective.
- 2 Show views and editors as desired.
- 3 Select Window > Perspective > Save Perspective As (Window > Save Perspective As in the plug-in configuration of Flash Builder).
- 4 In the Save Perspective As dialog box, enter a new name for the perspective, then click OK.

To configure a perspective, do the following:

- 1 Open the perspective to configure.
- 2 Select Window > Customize Perspective.
- 3 Click the Shortcuts tab or the Commands tab, depending on the items you want to add to your customized perspective.
- 4 Use the check boxes to select which elements to see on menus and toolbars in the selected perspective.
- 5 Click OK.
- 6 Select Window > Save Perspective As.
- 7 In the Save Perspective As dialog box, enter a new name for the perspective and click OK.

When you save a perspective, Flash Builder adds the name of the new perspective to the Window > Open Perspective menu.

### Delete a customized perspective

You can delete perspectives that were previously defined. You cannot delete a perspective you did not create.

- 1 Open the Preferences dialog and select General > Perspectives.



- 2 Under Available Perspectives, select the perspective you want to delete.
- 3 Click Delete, then click OK.

## Reset a perspective

You can restore a perspective to its original layout after changing it.

- 1 Open the Preferences dialog and select General > Perspectives.
- 2 Under Available perspectives, select the perspective to reset.
- 3 Click Reset, then click OK.

## Work with views

### Move and dock views

You can move views to different locations in the workbench, docking or undocking them, as needed.

- 1 Drag the view by its title bar to the desired location.

As you move the view around the workbench, the pointer changes to a drop cursor. The drop cursor indicates where you'll dock the view when you release the mouse button.



*You can drag a group of stacked views by dragging from the empty space to the right of the view tabs.*

You can also move a view by using the view's context menu. Open the context menu from the view's tab, select Move > View, move the view to the desired location, and click the mouse button again.

- 2 (Optional) Save your changes by selecting Window > Save Perspective As.

### Rearrange tabbed views

In addition to docking views at different locations in the workbench, you can rearrange the views in a tabbed group of views.

- ❖ Click the tab of the view to move, drag the view to the desired location, and release the mouse button. A stack symbol appears as you drag the view across other view tabs.

### Switch between views

There are several ways to switch to a different view:

- Click the tab of a different view.
- Select a view from the Flash builder Window menu.
- Use a keyboard shortcut

Use Control+F7 on Windows, Command+F7 on Macintosh. Press F7 to select a view.



*For a list of all keyboard shortcuts, go to Help > Key Assist*

### Create and work with fast views

Fast views are hidden views that you can quickly open and close. They work like other views, but do not take up space in the workbench while you work.

Whenever you click the fast view icon in the shortcut bar, the view opens. Whenever you click anywhere outside the fast view (or click Minimize in the fast view toolbar), the view becomes hidden again.

**Note:** *If you convert the Package Explorer view to a fast view, and then open a file from the Package Explorer fast view, the fast view automatically is hidden to allow you to work with that file.*

To create a fast view, do the following:

- ❖ Drag the view you want to turn into a fast view to the shortcut bar located in the lower-left corner of the workbench window.

The icon for the view that you dragged appears on the shortcut bar. You can open the view by clicking its icon on the shortcut bar. As soon as you click outside the view, the view is hidden again.

To remove a fast view to normal view, do the following:

- ❖ From the view's context menu, deselect Fast View.

## Filter the Tasks and Problems views

You can filter the tasks or problems that are displayed in the Tasks or Problems views. For example, to see only problems that the workbench has logged, or tasks that you logged as reminders to yourself, filter items according to which resource or group of resources they are associated with. You can filter by text string in the Description field, by problem severity, by task priority, or by task status.

- 1 In Tasks or Problems view taskbar, click Filter.
- 2 Complete the Filters dialog box and click OK.

For more information about views, see [“Work with views”](#) on page 17.

## Flash Builder editors

When you develop applications in Flash Builder, you use the MXML, ActionScript 3.0, and CSS editors.

Editors are associated with resource types, and as you open resources in the workbench, the appropriate editor is opened. Each editor provides the functionality required to author the given resource type.

**MXML editor** You use the MXML editor to edit MXML and to embed ActionScript and CSS code in `<fx:Script>` and `<fx:Style>` tags. The MXML editor has two modes: Source and Design. Source mode is used for editing code. Design mode is used for visually laying out and designing your applications. The two modes are synchronized and changes in one mode are immediately reflected in the other.

**ActionScript editor** You use the ActionScript editor to edit ActionScript class and interface files. You can embed ActionScript functions into an MXML file by using the `<fx:Script>` tag. However, it is a common practice to define classes in separate ActionScript files and then import the classes into MXML files. Using this method, you can define most of your applications in ActionScript.

**CSS editor** You use the CSS editor to display and edit Cascading Style Sheets. You can then apply styles to the visual elements of your applications. For more information, see [Styles and themes](#).

**Note:** *The CSS editor is a single-purpose editor for projects that use the Flex 4 SDK. However, for projects using the Flex 3 SDK, the CSS editor operates in two modes (Source and Design).*

## More Help topics

[“Code Development Tools in Flash Builder”](#) on page 44

[Flash Builder Tips and Tricks](#)

## Use the MXML editor in Design mode

### View an MXML file in Design mode

- 1 If the MXML file is not already open in the MXML editor, double-click the file in the Package Explorer to open it.
- 2 If the MXML editor displays source code, click Design at the top of the editor area.

You can quickly switch between modes by pressing Control+` (Left Quote).

Switching between Source and Design modes automatically shows or hides design-related views like the Components, Properties, and States views. To turn this behavior on and off, from the Preferences dialog, select Flex> Editors > Design Mode, then select the Automatically Show Design-related Views option.

### Pan and scroll in the design area

- ❖ Click the Pan Mode button on the right side of the editor toolbar. Press H to enter Pan Mode from the keyboard. To temporarily enter Pan Mode, press and hold the spacebar on the keyboard. You cannot select or move items in Pan Mode.

### Zoom in the design area

There are several ways to use the zoom tool. You can select percentages from the main and pop-up menus, click the Zoom Mode button in the toolbar, or use keyboard shortcuts. The current magnification percentage is always displayed in the toolbar.

- From the main menu, select Design > Zoom In or Design > Zoom Out. You can also select the Magnification submenu and choose a specific percentage.
- Click the Zoom Mode button in the toolbar or press Z from the keyboard. A plus symbol cursor appears in the design area.
- Select a percentage from the pop-up menu next to the Select, Pan, and Zoom Mode buttons on the editor toolbar. The design area changes to the selected percentage or fits to the window.
- Right-click in the design area to select Zoom In, Zoom Out, or the Magnification submenu. The design area changes to your selection.

You can always use the following keyboard shortcuts from the design area:

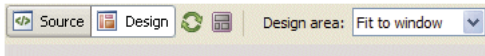
- Zoom In: Ctrl+= (Command+= on Mac OS)
- Zoom Out: Ctrl+- (Command+- on Mac OS)

For more keyboard shortcuts, select Help > Key Assist.

## Refresh Design mode to render properly

If necessary, you can refresh the MXML and CSS editors' Design mode to render your layout properly. The rendering of your layout can become out of date in certain situations. For example, if you modify a visual element in a dependent Flash component (SWC). Styles and skins are not rendered properly sometimes because Flash Builder has to rebuild the file.

- ❖ Click the Refresh button in the editor toolbar.



## Work in both Source and Design modes simultaneously

You can create a split view so that you can work in both Source and Design modes simultaneously.

- 1 From the option menu on the editor's tab, select New Editor.  
You now have two editor tabs for the same file.
- 2 Drag one of the tabs to the right to position the editor windows side-by-side.
- 3 Set one of the editors to Design mode, and set the other editor to Source mode.

To work only in the Source mode disabling the Design mode, from the Window menu, deselect Enable Design Mode.

## Work with components visually in MXML Design Mode

The MXML editor in Design mode lets you work with components visually so you can see what your application looks like as you build it. You can view, select, pan, move, resize, scroll, and magnify items in the design area. Doing so is especially useful when you have embedded container components.

### Use the Components view

In Design mode of the MXML editor, Flash Builder displays the Components view, which you use to drag components to the design area as you lay out your application.

By default, the Components view displays recommended components for a project. The version of the Flex SDK for a project determines which components appear as recommended components.

For example, when creating a project, Flash Builder, by default, uses the Flex 4.6 SDK. Recommended components for the Flex 4.6 SDK are Spark components. MX components that do not have an equivalent Spark component also appear as recommended components.

The Components view groups components according to category, such as Controls, Layout, Navigators, and Charts. There is also a Custom category that lists components defined in separate MXML and ActionScript files. For more information, see [“Create custom MXML components”](#) on page 216.

**Note:** The Components view lists visible components.

### Modify how components appear in Components view

Use the View menu within the Components view to modify how components appear.

- 1 To show all components, deselect Only Show Recommended Components.  
When you specify to show all components, Components view groups the components according to Spark and MX components.
- 2 To show fully qualified class name, select Show Fully Qualified Class Names.

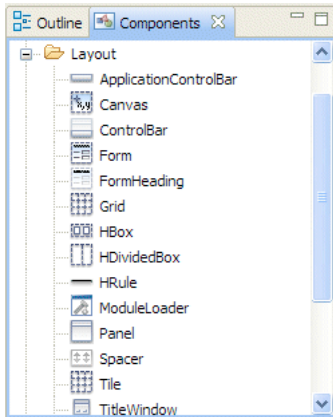
## Add components in MXML Design mode

- 1 In the MXML editor's Design mode, open the MXML file in which you want to insert the component.

An MXML file must be open in Design mode to use the Components view. The MXML file can be the main application file (a file with an Application container) or a custom MXML component file.

- 2 In the Components view, locate the component that you want to add.

If the Components view is not open, select Window > Components.



Components view organizes components by category. The Components view also displays recommended components for a project. You can change the view settings to list all components. For more information, see [“Components view”](#) on page 11.

- 3 Drag a component from the Components view into the MXML file.

The component is positioned in the layout according to the layout rule of the parent container.

## Select and move components in the design area

- ❖ Click the Select Mode (arrow) button on the right side of the editor toolbar. Select Mode is activated by default when you open a document. Press V on the keyboard to enter Select Mode. Drag-and-drop a component to the desired place. You can also drag to resize and click to select.



## Select and deselect multiple components

You can select more than one component in an MXML file. Selecting multiple components can be useful if you want to set a common value for a shared property.

- Control-click (Command-click on Macintosh) each component in the layout.
- Click the page background and draw a box that overlaps the components.
- In Outline view (Window > Outline), Control-click (Command-click on Macintosh) the components in the tree control.

To deselect multiple components, do one of the following:

- Click the background container.
- Click a deselected component.

- Click in the gray margin around the root component.

## Position components

You can change the position of components visually depending on the layout property of the parent container. If the parent container specifies absolute positioning, you can drag-and-drop components to reposition them. Otherwise, repositioned components follow the layout rules of the parent container.

By default, Spark containers use the `BasicLayout` property, which allows absolute positioning. For some MX containers, you can also specify a layout property of `absolute`. For more information, see MX layout containers.

**Note:** *You can also use layout constraints to position a component in a container. Layout constraints specify how to reposition components when a container resizes. However, they can also be used to specify position in containers that have a fixed size. See [Using constraints to control component layout](#).*

- 1 In the MXML editor's Design mode, select the component in the layout and drag it to a new position.

The component is positioned in the layout according to the layout rules of the parent container.

If the container has absolute positioning, you can drag and position components anywhere in the container.

If you move a VGroup container in an HGroup container, the VGroup container is positioned into the horizontal arrangement with the other child containers (if any).

- 2 In Design mode, select the component's parent container and edit the component's layout properties in the Properties View.

In some cases, you can change the position of child components by changing the properties of the parent container. For example, you can use the `verticalGap` and `horizontalGap` properties of a container to set the spacing between child components and the `direction` property to specify either a row or column layout.

## Size components

You can change the size of a component in Design mode by dragging a resize handle, using the Design menu, or by editing its properties in the Properties View.

**Note:** *Use a constraint-based layout to specify how a component dynamically resizes when a container resizes. For more information, see [Using constraints to control component layout](#).*

### Size a component visually

- ❖ In the MXML editor's Design mode, click the component and drag a resize handle to resize the component.
  - To constrain the proportions of the component, hold down the Shift key while dragging.
  - If snapping is enabled, as you resize, snap lines appear to line up the edges of the component with nearby components. To enable or disable snapping from the main menu, select Design > Enable Snapping.

### Make two or more components the same width or height

- 1 In Design mode, select two or more components.
- 2 In the Design menu, select one of the following sizing options:

**Make Same Width** Sets the `width` property for all selected components to that of the component you selected first.

**Make Same Height** Sets the `height` property for all selected components to that of the component you selected first.

If all selected components are in the same container, and the first component you select has a percent width or height specified, all items are set to that percent width or height. Otherwise, all components are set to the same pixel width or height.

### Size a component by editing its properties

- 1 In Design mode, select the component.

You can Control-click (Shift-click on Mac OS) more than one component to set their sizes simultaneously.

- 2 In the Properties View (Window > Properties), set the `height` or `width` property of the selected component or components.

The Properties View provides three views for inspecting a component's properties: a standard form view, a categorized table view, and an alphabetical table view. You can switch between them by clicking the view buttons in the toolbar.

### Use snapping to position components

When you drag a component visually in a container that has absolute positioning, the component can snap into place, depending on where you drop it relative to existing components. The components can line up vertically or horizontally.

By default, Spark containers use the `BasicLayout` property, which allows absolute positioning. For some MX containers, you can also specify a layout property of `absolute`. For more information, see [MX layout containers](#).

You can disable snapping for one component or for all components.

### Enable or disable snapping

- ❖ With the MXML file open in the MXML editor's Design mode, select (or deselect) Design > Enable Snapping.

### Enable or disable snapping as a preference

- 1 Open the Preferences dialog.
- 2 Select Flash Builder > Editors > Design Mode in the sidebar of the Preferences dialog box.
- 3 Select or deselect the Enable Snapping option.

### Align components

You can visually align components relative to each other in a container that has absolute positioning. By default, Spark containers use the `BasicLayout` property, which allows absolute positioning. For some MX containers, you can also specify a layout property of `absolute`.

You can also center components in a container by using a constraint-based layout. For more information, see [Using constraints to control component layout](#).

### Align components in a container that has absolute positioning

- 1 In the MXML editor's Design mode, select two or more components in the container.

For more information, see [“Select and deselect multiple components”](#) on page 21.

- 2 Select one of the following alignment options from the Design menu:

**Align Left** Positions all selected components so that their left edges align with the edges of the first component that you selected.

**Align Vertical Centers** Positions all selected components so that their vertical center lines are aligned with the vertical center line of the first component you selected.

**Align Right** Positions all selected components so that their right edges align with the edges of the first component that you selected.

**Align Top** Positions all selected objects so that their top edges align with the edges of the first component you selected.

**Align Horizontal Centers** Positions all selected components so their horizontal center lines are aligned with the horizontal center line of the first component you selected.

**Align Bottom** Positions all selected components such that their bottom edges align with the edges of the first component you selected.

**Align Baselines** Positions all selected components so that their horizontal text baselines are aligned with the text baselines of the first component that you selected. For components that have no text baseline (such as HGroup), the bottom edge is considered the baseline.

For objects with no layout constraints, Flash Builder adjusts the *x* property to change the vertical alignment, and adjusts the *y* property to change the horizontal alignment.

For objects with layout constraints, Flash Builder adjusts the left and right constraints to change the vertical alignment and adjusts the top and bottom constraints to change the horizontal alignment. Only existing constraints are modified; no new constraints are added.

For example, suppose component A has a left constraint and no right constraint, and component B has both a left and right constraint. If you select component A and B and then select Design > Align Vertical Centers, Flash Builder adjusts the left constraint of object A and both the left and right constraints of object B to align them. The unspecified right constraint of object A remains unspecified.

## Nudge components

You can fine-tune the position of components in a container that has absolute positioning by adjusting the components one pixel or ten pixels at a time in any direction with the arrow keys.

### Nudge components by one pixel

- ❖ Select one or more components in the MXML editor's Design mode and press an arrow key.

### Nudge components by ten pixels

- ❖ Select one or more components in the MXML editor's Design mode and press an arrow key while holding down the Shift key.



*Holding down the arrow key continues to move the component.*

## Hide container borders

By default, Flash Builder shows the borders of containers in the MXML editor's Design mode. However, you can hide these borders.

- ❖ Select Design > Show Container Borders.

This command is a toggle switch. Select it again to show the borders.

## Set component properties

You visually set the properties of components in the design area or in the Properties view.



### **Edit the text displayed by a component**

- ❖ To edit text displayed by a component such as a Label or TextInput control, double-click the component and enter your edits.

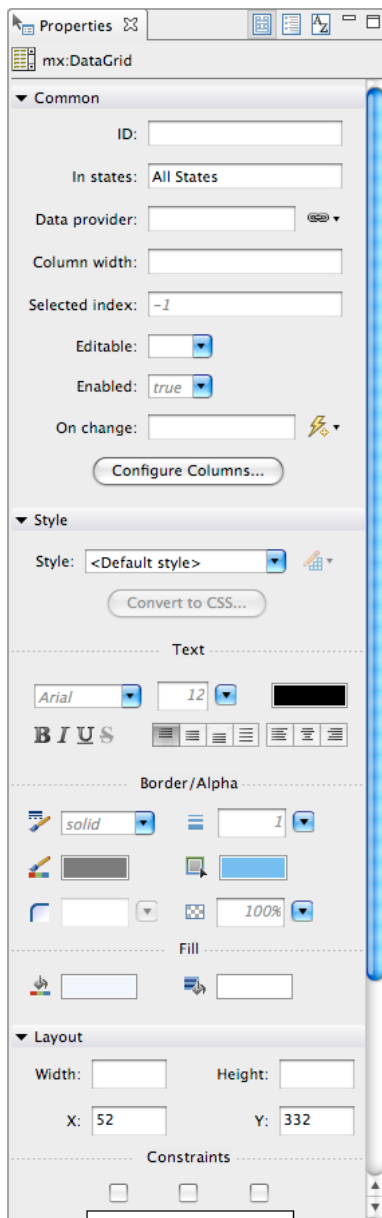
### **Change text in the ID field**

When you change text in the ID field, you are prompted to update all references in the workspace with the new ID. You can suppress this dialog box on the Design Mode preferences page:

- 1 Open the Preferences window.
- 2 Select Flash Builder > Editors > Design Mode.
- 3 Select or deselect Always Update References When Changing IDs in the Properties View.

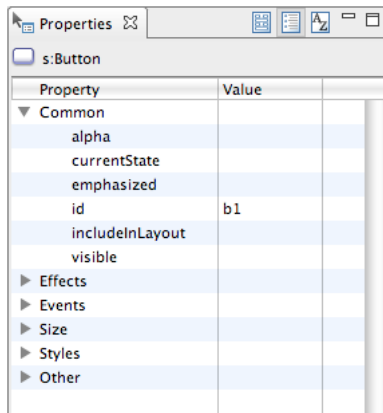
### **Set other properties of a component**

- ❖ Select the component and set its properties in the Properties view (Window > Show View > Other > Flash Builder > Properties).



Properties view for a DataGrid component

To set the properties in Category view or Alphabetical view, click the view buttons in the toolbar:



Category view for a Button component

**Note:** To apply your last edit, press Enter or Tab, or click outside the view.

## Copy components to other MXML files

You can visually copy and paste components from one MXML file to another.

- 1 Make sure that the two MXML files are open in the MXML editor's Design mode.
- 2 Select the component or components in one file. Then select Edit > Copy.
- 3 Switch to the other file, click inside the desired container, and select Edit > Paste.

## Delete components

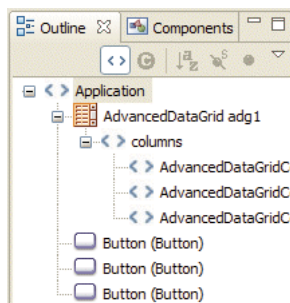
You can delete components from your user interface. Select a component and do any of the following:

- Press the Delete key on your keyboard
- From the context menu for a component, select Delete.
- From the Flash Builder Edit menu, select Delete.

## Inspect the structure of your MXML

Use Outline view (Window > Outline) in Design mode to inspect the structure of your design and to select one or more components. When you have multiple view states, Outline view shows you the structure of the current view state.

- 1 With the MXML file open in Design mode, select Outline view.



- 2 In Outline view, select one or more components.

The selected components in Outline view are also selected in Design mode of the editor.

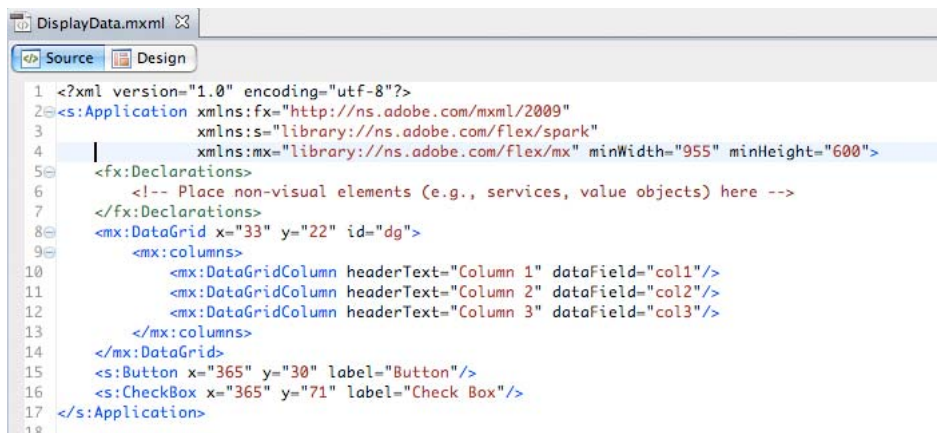
In Source mode of the editor, only the first selected component in Outline view is also selected in the editor.

## More Help topics

[Flash Builder Tips and Tricks \(Developer Center Article\)](#)

## Work with components in MXML Source mode

When using Source mode of the MXML editor, Content Assist helps you add standard Flex containers and controls to your user interface.



The following example shows how to use Content Assist to insert an `<s:VGroup>` container into an `<s:HGroup>` container. Because `HGroup` is a Spark component, Content Assist recommends a `VGroup` container, and not `<mx:VBox>`.

- 1 Open an MXML file in the MXML editor's Source mode.

The MXML file can be the main application file (a file with an `Application` container) or a custom MXML component file.

- 2 Place the insertion point in the parent container tag.

For example, to insert a `VGroup` container inside an `HGroup` parent container, place the insertion point after the opening `<s:HGroup>` tag:

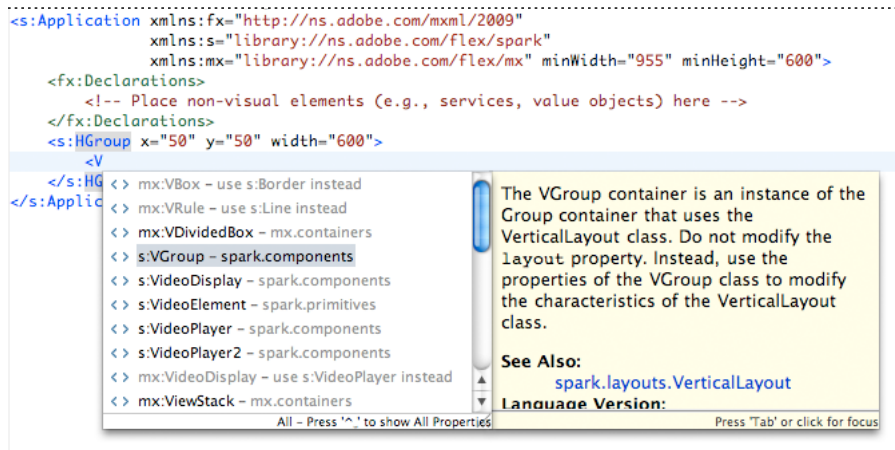
```
<s:HGroup>
    insertion point here
</s:HGroup>
```

- 3 Type the component tag.

As you type the tag, Content Assist appears suggesting possible entries. Recommended components are listed in the normal black type color. Components that are not recommended appear gray.

In this example, `VGroup` is recommended and `VBox` is not recommended.

- 4 If necessary, use the arrow keys to select your tag from the menu, then press Enter.



In addition to the standard components, Content Assist lists custom components you defined in separate MXML and ActionScript files and saved in the current project or in the source path of the current project.

Content Assist can also suggest properties, events, effects, and styles. Press Control+Space to cycle through the recommendations in Content Assist.

You can change the type and order of recommendations for Content Assist. From the Preferences dialog, select Flash Builder > Editors > MXML Code > Advanced.

## More Help topics

“Content Assist” on page 44

“Create custom MXML components” on page 216

## Associate editors with file types

You can associate editors with various file types in the workbench.

- 1 Select Window > Preferences.
- 2 Click the plus button to expand the General category.
- 3 Click the plus button to expand the Editors category, and then select File Associations.
- 4 Select a file type from the File Types list.

To add a file type to the list, click Add, enter the new file type in the New File Type dialog box, and then click OK.

- 5 In the Associated Editors list, select the editor to associate with that file type.

To add an internal or external editor to the list, click Add and complete the dialog box.

- 6 Click OK.




You can override the default editor preferences from the context menu for any resource in one of the navigation views. Select Open With from the context menu.

## Edit files outside the workbench

You can edit an MXML or ActionScript file in an external editor and then use it in Flash Builder. The workbench performs any necessary build or update operations to process the changes that you made to the file outside the workbench.

### Refresh an MXML or ActionScript file edited outside the workbench

- 1 Edit the MXML or ActionScript file in the external editor of your choice.
- 2 Save and close the file.
- 3 Start Flash Builder.
- 4 From one of the navigation views in the workbench, select Refresh from the context menu.

 *If you work with external editors regularly, you can enable auto-refresh. To do so, select Window > Preferences, expand the General category, select Workspace, and check Refresh Automatically. When you enable this option, the workbench records any external changes to the file. The speed with which the auto-refresh happens depends on your platform.*

## Tiling editors

The workbench lets you open multiple files in multiple editors. But unlike views, editors cannot be dragged outside the workbench to create new windows. You can, however, tile editors in the editor area, so that you can view source files side by side.

- 1 Open two or more files in the editor area.
- 2 Select one of the editor tabs.
- 3 Drag the editor over the left, right, upper, or lower border of the editor area.

The pointer changes to a drop cursor, indicating where the editor appears when you release the mouse button.

- 4 (Optional) Drag the borders of the editor area of each editor to resize the editors as desired.

## Maximize a view or editor

There are several ways you can maximize (restore) a view or editor so that it fills the workbench window.

- From the context menu on the view or editor's title bar, select Maximize (Restore).
- Double-click the tab of a view.
- From the Flash Builder menu, select Window > Maximize/Restore.
- Click the Maximize/Restore icons in the upper-right corner of the view or editor.

## Projects in Flash Builder

Flash Builder groups the resources (folders and files) that constitute an application into a container called a *project*.

To manage projects, you use the Package Explorer view, which lets you add, edit, and delete resources. You can also close projects within a workspace, import resources, and link to external resources.

## Types of projects

Flash Builder supports the following various types of projects, depending on the type of application you are building.

### Flex Projects

Use a Flex Project to build either a web application or a desktop application that is based on the Flex framework. A web application runs in Adobe Flash Player, while a desktop application runs in Adobe AIR. When creating the project, you specify whether the project is for web or desktop applications.

A Flex project contains a set of properties that control how the application is built, where the built application resides, how debugging is handled, and the relationships to other projects in the workspace.

A Flex Project contains a default MXML application file. Other MXML files in the project can also be application files.

See [“Flex projects”](#) on page 81 and [“Developing Applications with Flex”](#) on page 167

### Flex Mobile Projects

Use a Flex Mobile Project to create an AIR application targeted for a Mobile Platform. The application is based on the Flex framework. You can use Flash Builder to preview, debug, and profile mobile applications from the desktop or on a device.

A Flex Mobile Project has a single default MXML application file. Typically, a mobile application has a set of View components that display content on a device. The default MXML application file launches the default View component.

Flash Builder uses the AIR Debug Launcher (ADL) to preview mobile applications on the desktop. Although not a true emulation, the ADL allows you to view the application layout and behavior, with options to rotate the application.

You can preview a mobile application on a device connected to the development computer’s USB port. When previewing on a device, Flash Builder exports the application to the device.

See [“Flex mobile projects”](#) on page 83 and [Developing Mobile Applications with Flex and Flash Builder](#).

### ActionScript Projects

Use an ActionScript Project to create web or desktop applications that are based on either the Flash or Adobe AIR APIs. When creating the project, you specify whether the project is for a web or a desktop application.

Because these projects do not use MXML to define a user interface, you cannot view the application layout and design in Design mode. You work exclusively in the source editor, the debugging tools as necessary, and then build the project into SWF files to preview and test your application.

When you create an ActionScript project or a stand-alone ActionScript file to contain functions, a class, or interface, the Flex development perspective is modified to support the ActionScript editor. The primary supporting views of the ActionScript editor are the Outline and Problems views.

See [“ActionScript projects”](#) on page 84.

### ActionScript Mobile Projects

Use an ActionScript Mobile Project to create mobile applications that are based on the Adobe AIR API. When creating the project you specify a target Mobile Platform and some mobile applications settings. You can use Flash Builder to preview the mobile application from the desktop or on a device.

Flash Builder uses the AIR Debug Launcher (ADL) to preview, debug, and profile mobile applications on the desktop. Although not a true emulation, the ADL allows you to view the application layout and behavior, with options to rotate the application.

You can preview a mobile application on a device connected to the development computer's USB port. When previewing on a device, Flash Builder exports the application to the device. You can use Flash Builder to debug the application exported to a device.

See [“Create ActionScript mobile projects”](#) on page 84.

### **Flex Library Projects**

Use a Flex Library Project to build custom code libraries that you share between applications or distribute to other developers. Typically, you use library projects to package and distribute components and resources to other developers.

A library project generates a SWC file, which is an archive file for Flex components and other resources.

See [“Use Flex library projects”](#) on page 212.

### **Flash Catalyst Compatible Projects**

Use a Flash Catalyst Compatible Project if you plan to share project files with Adobe Flash Catalyst projects. This feature lets designers and developers collaborate and work in-tandem on the same project.

See [“Flash Catalyst compatible projects”](#) on page 85.

### **Flash Professional Projects**

Use a Flash Professional Project to edit, build, or debug FLA or XFL files created in Adobe Flash Professional. This feature allows Flash Professional developers to take advantage of the editing and debugging environment available with Flash Builder. Flash Professional projects are available in Flash Builder only if you have Flash Professional installed.

See [“Using Flash Builder with Flash Professional”](#) on page 228.

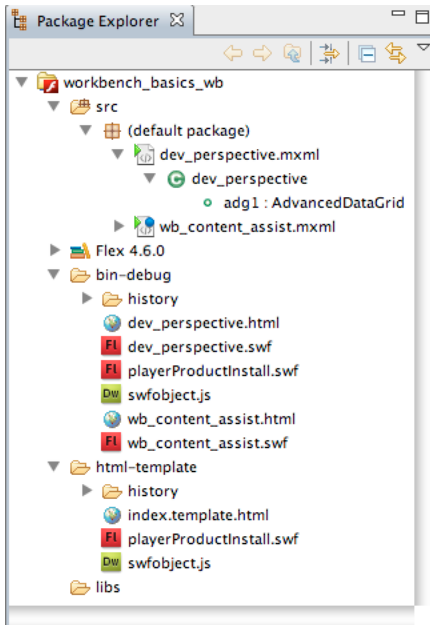
## **Manage projects**

You use the Package Explorer to add and import resources into projects, export projects, and move and delete resources.



## Projects in Package Explorer

All projects in a workspace are displayed in the Package Explorer, as the following example shows. The Package Explorer provides a tree view of projects from both a physical view and logical (flat) view. Using this view, you manage your projects by adding and deleting resources (folders and files), importing and linking to external resources, and moving resources to other projects in the workspace.



Highlights of the Package Explorer include:

- Displaying ActionScript packages in either a hierarchical or flat presentation.  
Use the Package Explorer's menu to specify the package presentation.
- Project libraries are represented in two top-level nodes, one node for the Flex SDK and the other for referenced libraries.  
You can expand a library's contents and open editors to view attachments.
- Error and warning badges on Package Explorer nodes notify you of problems within a package.
- You can limit which projects and resources are visible.  
You can create a working set (a collection of resources), create display filters, and sort resources by name and type. These options are available from the Package Explorer menus. For more information about modifying views, see ["Customize the workbench"](#) on page 241.
- You can expand ActionScript, MXML, and CSS files and see a tree view of their contents.

From the Package Explorer, you can open the project resources for editing. For example, you can edit MXML and ActionScript in `<fx:Script>` blocks and CSS in `<fx:Style>` blocks, or you can switch to Design mode and visually manipulate components and controls to create the application's layout and behavior. For more information about working with the Flash Builder editors, see ["Flash Builder editors"](#) on page 18 and ["Build user interfaces"](#) on page 168.

Then you add projects, files, and folders, and organize and manage them as needed. See ["Projects in Flash Builder"](#) on page 30.

Most menu commands that you use in the Package Explorer view are also available from the view's context menu.

## More Help topics

[“Manage projects”](#) on page 32

[“Project resources”](#) on page 37

## Move a project from one workspace to another

You use a combination of deleting and importing operations to move a project from one workspace to another. When you delete a project from a workspace, you can remove it from the workspace but leave it in the file system (see [“Delete a project”](#) on page 34). After you remove a project from one workspace, you can import it into another.

## Specify an SDK for a project

When creating a Flex project, you can specify which Flex SDK to use. However, you can later modify the SDK settings by selecting Project > Properties > Flex Compiler > Use a specific SDK.

If you want to compile your project against a version of the Flex SDK that is not available in your Flash Builder installation, you can download the SDK and add it to your installation. For example, download the SDK, and add it to Flash Builder using Project > Properties > Flex Compiler > Configure Flex SDKs.

## Delete a project

When you delete a project, you remove the project from the current workspace. You can also remove the project from the file system at the same time.

Instead of deleting the project from the workspace, you can close the project. Closing the project lets you keep a reference to it in your workspace and also free some system resources. For more information, see [“Close and open projects”](#) on page 34.

- 1 In the Package Explorer, select the project to delete.
- 2 Select Edit > Delete from the main menu.
- 3 Select an option:

**Also Delete Contents Under Directory** Permanently removes the project from the workspace and the file system.

**Do Not Delete Contents** Removes the project from the workspace but not from the file system.

## Close and open projects

To save memory and improve build time without deleting a project, you can close it. When you close a project, you collapse the project and its resources, however, the name remains visible in the Package Explorer. A closed project requires less memory than an open project, and is excluded from builds. You can easily reopen the closed project.

- 1 In the Flex Package Explorer, select the project to close or reopen.
- 2 From the Package Explorer context menu, select Close Project or Open Project.

## Switch the main application file

When you create a project, the main application file is generated for you. By default, it is named after the project. The main application file is the entry point into your applications and becomes the basis of the application SWF file. However, as you add files to your application, you might want to designate a different file as the main application file.

If you prefer to set multiple files as application files so that each application file is built in to a separate SWF file, see [“Manage project application files”](#) on page 35.

- 1 In the Package Explorer, select the MXML application file that you want to make the main application file.

- 2 From the Package Explorer context menu, select Set as Default Application.

You can manage the application files in your project by selecting Project > Properties > Flex Applications (or ActionScript Applications if you're working with an ActionScript project).

## Manage project application files

Usually, a project has a single main application file, which serves as the entry point to your application. The Flash Builder compiler uses this file to generate the application SWF file.

For example, you can have a complex application with many custom MXML components that represent distinct but interrelated application elements. You can create an application file that contains a custom component and then build, run, and test it separately.

By default, whenever you add an MXML application file to your Flex project, you can run the application, and it is added to the list of project application files. All files defined as application files must reside in your project's source folder.

You can manage the list of application files by selecting a project and viewing its properties.

- 1 In the Package Explorer, select a project.
- 2 Select Project > Properties from the main menu or select Properties from the context menu.
- 3 In the Project Properties dialog box, select Flex Applications (or ActionScript Applications if you are working with an ActionScript project).
- 4 Add and remove application files as needed. Click OK.

## Switch the workspace

You can work in only one workspace at a time. When you install and run Flash Builder for the first time, you are prompted to create a workspace, which becomes the default workspace. You can create other workspaces and switch among them by either selecting the workspace when you start Flash Builder or by selecting File > Switch Workspace.

## Set Flex project properties

Each Flex project has its own set of properties. To set these properties, select the project in the Package Explorer view. Then select Project > Properties from the main menu. You can also select Properties from the context menu for the project.

You can set the following project-specific preferences in Flash Builder:

**Resource** Displays general information about the project, settings for text encoding, and the operating system line delimiter.

**Builders** Specifies the build tool to use. A standard builder is included in Flash Builder. You can use Apache Ant (an open-source build tool) to create build scripts or import existing Ant build scripts.

See [“Customize builds with Apache Ant”](#) on page 101.

**Data Model** Available only with ADEP Data Services (formerly known as LiveCycle Data Services) and higher. Specifies the location of the data model file, which contains service and data type information for ADEP Data Services ES.

**Data/Services** For projects that access data services, specifies whether to use the default code generator for accessing services. You can also specify whether to use a single server instance when accessing services.

See [Extending service support in Flash Builder](#) for information on extending Flash Builder to use custom code generation.

See [Using a single server instance](#) for information on using a single server instance when accessing services.

**Flex Applications** Displays the names of the project files that are set as application files, which can be compiled, debugged, and run as separate applications.

See [“Manage project application files”](#) on page 35.

**Flex Build Path** Specifies the build path, which specifies where external source and library files are located. You can modify the build path and also change the name of the output folder.

See [“Set up a project output folder”](#) on page 98 and [“Build projects manually”](#) on page 100.

**Flex Compiler** Specifies optional compiler preferences, such as generating an accessible SWF file, enabling compiler warnings and type checking, specifying additional compiler arguments, Flex SDK version, and sets HTML wrapper settings.

See [“Advanced build options”](#) on page 100.

**Flex Modules** Specifies modules to build and optimize for the project. For more information about using modules in Flash Builder, see [“Create a separate project for modules in Flash Builder”](#) on page 220.

**Flex Server** Specifies the application server type for the project. When you create a project, you specify the application server type. You can change the application server type for a project here. If you change the application server type for a project, you may not be able to access data services previously configured.

See [“Flex projects”](#) on page 81 and [Creating a Flex project to access data services](#).

**Flex Theme** Specifies the theme to use for all applications in the project. You can specify one of the themes available with Flash Builder or import a theme.

See [“Apply themes”](#) on page 173.

**Project References** Lists the projects that the current project references.

**Run/Debug Settings** Manages launch configuration settings.

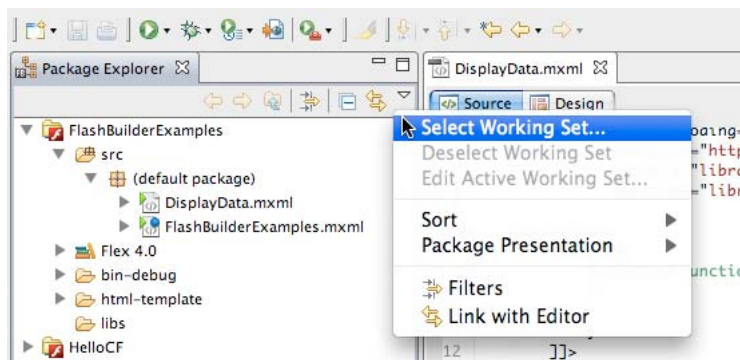
See [“Manage launch configurations”](#) on page 111.

## Create working sets

If your workspace contains many projects, you can create a working set to group selected projects together. You can then view separate working sets in the Package Explorer and Task views and also search working sets rather than searching everything in the workspace.

### Create a working set

- 1 In the Package Explorer view, open the toolbar menu and select **Select Working Set**.



2 Select New.

Flash Builder provides two set types: breakpoints (used in debugging) and resources.

3 Select the resources type and click Next.

4 Enter the working set name and then choose the projects in the workspace that you want to include in the working set.

5 Click Finish.

The working set is immediately applied to the Package Explorer view and only those projects and resources contained in the set are displayed.

### Display all projects in the workspace

- ❖ In the Package Explorer view, open the toolbar menu and choose Deselect Working Set.

## Project resources

Projects consist of resources (folders and files) that you can manage from the Package Explorer. Projects are contained within a workspace. The Package Explorer provides a logical representation of the workspace in the file system. The Package Explorer is refreshed each time you add, delete, or modify a resource.

You can also edit project resources directly in the file system, bypassing Flash Builder and the Package Explorer.

### Create or delete files and folders in a project

An application in Adobe® Flex® typically consists of an MXML application file, one or more views (mobile projects only), and one or more standard Flex components. You can also have one or more custom components defined in separate MXML, ActionScript, or Flash component (SWC) files. By dividing the application into manageable chunks, you can write and test each component independently from the others. You can also reuse a component in the same application or in other applications, which increases efficiency.

You can use Adobe Flash® Builder™ to build custom MXML and ActionScript components and then insert them into your applications. For information on building ActionScript components, see [“Create an ActionScript class”](#) on page 38.

You can also build an MXML component directly using code. For more information, see Simple MXML components.

### Create a file

- 1 In the Package Explorer, select File > New > File.
- 2 If you have multiple projects in your workspace, select the project to which you want to add the file.
- 3 Enter the filename and click Finish.

You can also add folders and files that are located outside the current project; for more information, see [“Link to resources outside the project workspace”](#) on page 40.

### Create MXML application files

- 1 In the Package Explorer, select the project to which you want to add the MXML application file.
- 2 From the Package Explorer context menu, select New > MXML Application.
- 3 The application file is created in the src folder, by default. You can select a different folder within the project, if necessary.

- 4 Specify a name for the application file, and select a Spark Layout, if necessary. For more information about Spark Layouts, see About Spark layouts
- 5 The following apply only for mobile projects:
  - a Specify an application ID.
  - b Specify the application template.  
See Choose an application template.
  - c Specify the mobile application permissions for the Google Android platform.  
See Choose mobile application permissions.
  - d Specify the platform settings.  
See Choose platform settings
  - e Specify the application settings.  
See Choose application settings.
- 6 Click Finish.

### Create an ActionScript class

You can use a wizard in Flash Builder to quickly create ActionScript classes for your Flex and ActionScript projects. The wizard also provides an easy way to generate stubs for functions that must be implemented.

- 1 Select File > New > ActionScript Class.
- 2 Specify the basic properties of your new class in the dialog box, and then click Finish.  
After clicking Finish, Flash Builder saves the file in the specified package and opens it in the code editor.  
  
If you saved the file in the current project or in the source path of the current project, Flash Builder also displays the component in the Components view so that you can rapidly insert it in your applications. For more information, see [“Add components in MXML Design mode”](#) on page 21.
- 3 Write the definition of your ActionScript class.  
For more information, see Create simple visual components in ActionScript .

### Create an ActionScript interface

You can use a wizard in Flash Builder to quickly create ActionScript interfaces for your Flex and ActionScript projects. An *interface* is a collection of constants and methods that different classes can share.

- 1 Select File > New > ActionScript Interface.
- 2 Specify the basic properties of your new interface in the dialog box, and then click Finish.
- 3 Add any constants or methods to your ActionScript interface that different classes share.

### Create an ActionScript skinnable component

You can use a wizard in Flash Builder to quickly create ActionScript components for your Flex projects.

- 1 Select File > New > ActionScript Skinnable Component.
- 2 Specify the basic properties of your new skinnable component, and then click Finish.

Flash Builder creates an `ActionScript` class that extends from the base component or the skinnable component. By default, the base component is `spark.components.supportClasses.SkinnableComponent`. You can, however, change the base component to any other component that is a subclass of the skinnable component.

You can also add any `ActionScript` interfaces that the new skinnable component can implement. For more information on creating `ActionScript` interfaces, see [“Create an `ActionScript` interface”](#) on page 38.

The `SkinnableComponent` is created in the selected location and package. A reference to the component is added automatically, by default, in the main `MXML` file.

## Create folders in a project

You can add folders to your project as needed. For example, you can create a folder to store all of your data models. Or, you can organize all the assets that make up the visual design of your application.

- 1 In Package Explorer select **File > New > Folder**.

- 2 If you have multiple projects in your workspace, select the project to add to the stand-alone folder.

If you create the folder in the source path folder, it is treated like a package name. You can then place source files inside that are recognized by the compiler.

If you create the folder outside the source path folder, you can later make it the root of a package structure by adding it to your source path. After you complete this procedure, select **Project > Properties** and then select **Flex Build Path**. Click **Add Folder** and navigate to the newly created folder.

- 3 Enter the folder name and click **Finish**.

## Delete folders and files

Deleting folders and files from your project removes them from the workspace and, therefore, from the file system.

**Note:** If you delete a linked resource, you delete only the link from your project, not the resource itself (see [“Link to resources outside the project workspace”](#) on page 40). However, if you’ve linked to a folder and you delete any of the files in it, they are removed from the file system.

- 1 In the Package Explorer, select the resource to delete.

- 2 Select **Edit > Delete** or press the **Delete** key, and click **Yes**.

The resource is deleted from the file system.

## Search for resources in the workbench

Flash Builder provides a search tool that lets you quickly locate resources.

To search the document that is currently open in the editor, do the following

- 1 Open the document to search.

- 2 Do either of the following:

- Press **Control+F** (Windows) or **Command+F** (Mac OS)
- Select **Edit > Find/Replace**.

- 3 Enter the text string to locate.

- 4 (Optional) Enter the replacement text string.

- 5 (Optional) Set the advanced search criteria.

- 6 Click **Find**, **Replace**, **Replace All**, or **Replace/Find**.

If the text string is located in the document, it is highlighted and, optionally, replaced.

**Note:** To do an incremental find, press *Control+J* (Windows) or *Command+J* (Mac OS).

To search all resources in the workspace, Flash Builder includes advanced search features that are more powerful than find and replace. Flash Builder lets you find and mark references or declarations to identifiers in ActionScript and MXML files, projects, or workspaces. For more information, see [“Find references and refactor code”](#) on page 69.

## Search for files

- ❖ Select *Search > Search* to conduct complex searches for files.

**Note:** Click *Customize* to define what kinds of search tabs are available in the Search dialog box.

## Use the Search view

The Search view displays the results of your search.

### Open a file from the list

- ❖ Double-click the file.

### Remove a file from the list

- ❖ Select the file to remove and click *Remove Selected Matches*.

### Remove all files from the list

- ❖ Click *Remove All Matches*.

### Navigate between matched files

- ❖ Click *Show Next Match* or *Show Previous Match*.

### View previous searches

- ❖ Click the *Down Arrow* next to *Show Previous Searches* and select a search from the pull-down list.

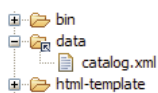
### Return to the Search view after closing it

- 1 Select *Window > Show View > Other*.
- 2 Expand the *General* category, select *Search*, and click *OK*.

## Link to resources outside the project workspace

You can create links to resources outside the project and workspace location. You can link to folders and files anywhere on the file system. This option is useful when you have resources that are shared between your projects. For example, you can share a library of custom Flex components or ActionScript files among many different Flex projects.

Folders that contain linked resources are marked in the Package Explorer (as the following example shows), so that you can distinguish between normal and linked resources.



Other examples for linking resources include a folder of image file assets, or situations when the output folder is not in the project root folder.



When you work with shared resources, the changes you make to the source folders and files affect all of the projects that are linked to them. Be cautious when you delete linked resources from your projects. In some cases you merely delete the link reference, and in others you delete the source itself. For more information, see [“Delete folders and files”](#) on page 39.

**Note:** *A best practice is to link other projects to your Library Project. Linked resources are encouraged only for third-party libraries with a SWC file file.*

#### Link to resources outside the project workspace

- 1 In the Package Explorer, select the project to add linked resources to.
- 2 Select File > New > Folder (or File).
- 3 Select the project or project folder to add the linked resources to.
- 4 Enter the folder or filename. The folder or filename you enter can be different from the name of the folder or file you are linking to.
- 5 Click the Advanced button.
- 6 Select Link to folder in the file system. Enter or browse to the resource location.
- 7 Click Finish to link the resource to your project.

#### Use a path variable to link to resources

Rather than linking to resources by entering the full path to the local or network folder where you store your files, you can define path variables. For more information, see [“Create a path variable”](#) on page 99.

- 1 In the Package Explorer, select the project to add linked resources to.



*Path variables can also be used in certain project settings, such as the library path and source path.*

- 2 Select File > New > Folder (or File if you want to add files).
- 3 Select the project or project folder to add the linked resources to.
- 4 Click the Advanced button to display the advanced options.
- 5 Select Link to folder in the file system. Click the Variables button.
- 6 Select a defined path variable, or click New to create a path variable.

If you selected a defined path variable, skip to step 9. If you clicked New, you'll see the New Variable dialog box.

- 7 Enter the path variable name and enter or browse to the file or folder location.

Click OK to create the path variable.

- 8 Select the new path variable in the Select Path Variable dialog box and click OK.
- 9 Click Finish to complete the link to the resource.



*You can also define and manage path variables by using the Flash Builder workbench preferences (Open the Preferences dialog and select General > Workspace > Linked Resources).*

## Move or share resources between projects in a workspace

### Move resources

When you work with multiple projects in a workspace, you can move resources from one project to another.

- 1 In the Package Explorer, select the resource to move.
- 2 Do one of the following:
  - Drag the resource to a new project.
  - Cut and paste the resource to another project.

When moving resources between projects, you can choose to update references.

**Note:** You can move both normal resources and linked resources. For information about linking resources, see [“Link to resources outside the project workspace”](#) on page 40.

### Share resources

To share resources between projects, place all shared resources into folders that can then be linked to each project by using the project’s source path. This is the best method for using shared resources such as classes, MXML components, and images. Updates to these resources are immediately available to all projects that use them. When your projects are compiled, the shared resources are added to the SWC file.

### Add an external resource folder to the source path

- 1 Select the project in the Package Explorer.
- 2 Select Project > Properties > Flex Build Path (or ActionScript Build Path if you are working with an ActionScript project).
- 3 On the build path properties page, select the Source Path tab.
- 4 Click the Add Folder button.
- 5 Enter or browse to the folder’s path, and click OK.

The folder is added to the source path.

You can also use the Source Path properties tab to edit, delete, or reorder items in the source path.

Folders that are added to the source path are marked in the Package Explorer.

### Refresh resources in the workspace

As you edit, add, or delete a project’s resources, the workbench automatically refreshes the various views that display these resources. For example, when you delete a file from your project, that change is immediately reflected in the Package Explorer.

You can also edit resources outside Flash Builder, directly in the file system. These changes are visible only inside Flash Builder after you refresh the workspace.

By default, in the stand-alone configuration of Flash Builder, the workspace is refreshed automatically. This option is configurable in Flash Builder preferences. Open the Preferences dialog and select General > Workspace. You can also change the Flash Builder default behavior so that it never refreshes the workspace automatically.

### **Manually refresh the workspace**

- ❖ In the Package Explorer, select Refresh from the context menu. All project resources in the workspace are refreshed.

### **Turn off automatic refresh preference**

- 1 Open the Preferences dialog and select General > Workspace.
- 2 Deselect Refresh Automatically.

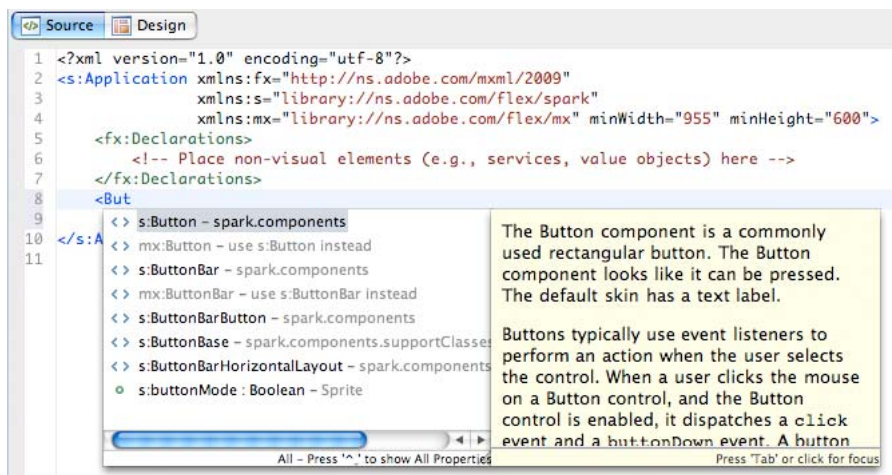
# Chapter 3: Code Development Tools in Flash Builder

You edit MXML, ActionScript, and CSS code in Adobe® Flash® Builder™ with separate editors. The Flash Builder workbench is both project- and document-centric. The appropriate editor opens automatically because the editors are associated with resource types. The Flash Builder editors share capabilities, including code hinting, navigation, formatting, refactoring, and other productivity-enhancing features.

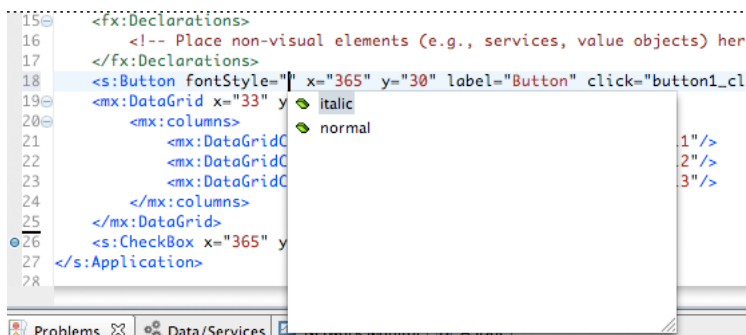
## Content Assist

As you enter MXML, ActionScript, and CSS code, hints and ASDoc reference documentation displays to help you complete your code. This feature is called *Content Assist*.

In the MXML editor, if you type within an MXML component, you are prompted with a list of all properties of that component. The following example shows code hints for properties of an MXML component:



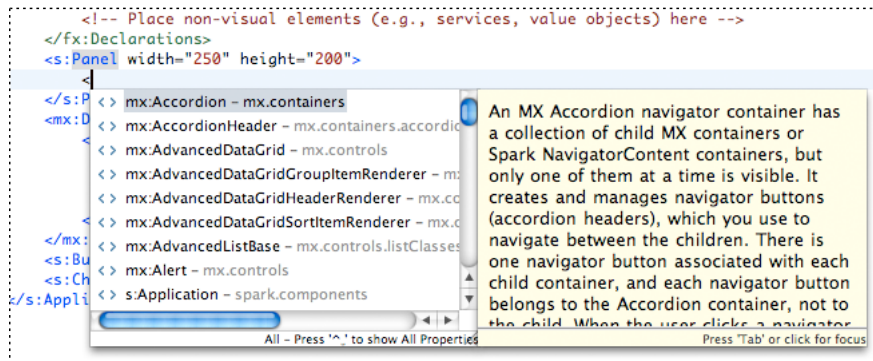
Selecting and entering properties displays possible property values (if predefined values exist). The following example shows code hints for property values:



Content Assist works similarly for the ActionScript editor and CSS editor.

## Content Assist in the MXML editor

In the MXML editor, code hints appear automatically as you enter your code. The following example shows the code hints that are displayed when you add a tag to a Panel tag. It also shows the ASDoc reference documentation. Click inside the ASDoc content, or press F2 to display the content in a separate, scrollable window. Click outside the ASDoc window to close it.



Content Assist categorizes code hints by type, showing you both visual and nonvisual MXML components, properties, events, and styles.

By default, Content Assist only displays code hints for *recommended types*. Recommended types are those components available in the declared namespace or are otherwise available, depending on the enclosing tags. The components that are available to you in an application depend on the namespace declaration of the application and also the tags enclosing the insertion point in the editor.

For example, in some contexts only Spark components are allowed. In other contexts, both Spark and Halo components are allowed. Content Assist filters the code hints, depending on the context.

Press Control+Space multiple times to cycle through filters for displayed code hints, as listed below:

- Initial display: Recommended types
- All components
- Properties
- Events
- Effects
- Styles
- Return to recommended types

Which code hints to display, and the order for cycling through code hints, is a user preference. To change the default setting from the Preferences dialog, see “[MXML code](#)” on page 247.

## Content Assist in the ActionScript editor

Code hints appear automatically as you enter your ActionScript code in the ActionScript editor.

Content Assist filters the code hints, depending on the context. Press Control+Space multiple times to cycle through filters for displayed code hints, as listed below:

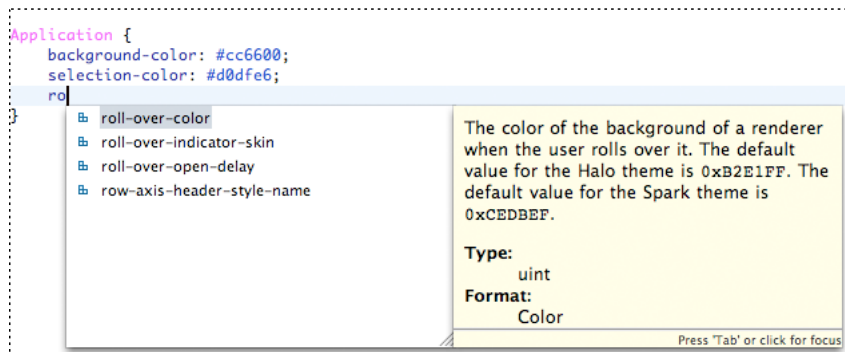
- Templates
- Variables
- Functions
- Classes and Interfaces
- Packages
- Namespaces

Which code hints to display, and the order for cycling through code hints, is a user preference. To change the default setting from the Preferences dialog, see “[ActionScript code](#)” on page 245.

## Content Assist in the CSS editor

Content Assist provides hints for CSS styles within embedded `<fx:Style>` tags or in stand-alone CSS documents, as the following example shows:

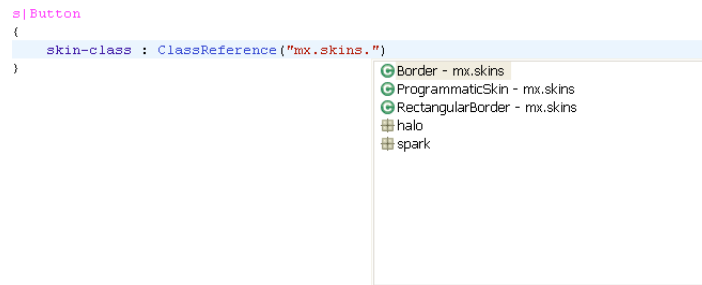
**Note:** Code hints in CSS documents do not appear until you press Control+Spacebar.



Content Assist also provides hints for component classes within the ClassReference in a CSS document, as the following example shows:



You can also choose to type the full-qualified class name, then the available classes and packages are displayed as follows:



The ClassReference tag provides hyperlink navigation to let you navigate to the referenced component or skin class. To do so, press the Control key (Command key for Mac) and move the cursor over the ClassReference tag, the class name becomes a hyperlink. For more information on hyperlink navigation, see [“Open code definitions”](#) on page 72.

## Display ActionScript reference summary

- 1 Begin entering a line of code that contains either an MXML or ActionScript class. You can also hover over the class. As you type, ActionScript reference summary for the class displays next to the code hints. If you hover over a class, just the ActionScript reference summary appears.
- 2 Click inside the ActionScript reference summary, or press F2 to display the ActionScript reference content in a separate, scrollable window. When you finish reading the documentation, click outside the window. The ASDoc window closes.
- 3 To display the ASDoc view, press Ctrl+3, and type asdoc, and select Views:

## Quick Assist

The Quick Assist feature in Adobe Flash Builder provides context-based assistance to help you quickly perform a task. Quick Assist lets you select an action from a list of actions that are applicable to your current code fragment.

To invoke Quick Assist, select Quick Assist from the context menu in the editor, and select the required action. You can also use the keyboard shortcut Control+1 (Windows) or Command+1 (Mac OS).

Currently, the following Quick Assist options are available:

### Rename in File

Use Quick Assist to rename code elements that have the Rename/Refactor action enabled. That is, variables, methods, class names, parameters, imports, and states and ActionScript code inside MXML tags.

To rename all instances of a variable or method in your file, place your cursor in the selected variable or method name, and press Control+1. Then, select the Rename In File option to rename the variable or method name. Similarly, you can change the ID property of an MXML component.

### Rename in Workspace

Use Quick Assist to rename code elements that have the Rename/Refactor action enabled across files in your workspace.

To rename all instances of a variable or method in your workspace, place your cursor in the selected variable or method name, and press Control+I. Then, select the Rename In Workspace option to rename the variable or method name. You can update all references to the variable or method in your workspace.

### Organize Imports

Place the cursor on any import statement, and press Control+I. Then, select the option to organize imports. For more information on organizing import statements, see [“Organize import statements”](#) on page 79.

### Generate Import Statement

If you have an undefined variable, you can place your cursor anywhere within the line of code and press Control + I. You get an option to import the component. If the component has MX and Spark equivalents, both the options appear.

For example, if you have an undefined variable `btn` in your code, as follows:

```
var btn:Button;
```

Place your cursor anywhere in the line of code, and press Control +I. You get options to import the Button component. If you select the Spark Button component, the import statement is created as follows:

```
import spark.components.Button;
```

### Split Variable Declaration

Use Quick Assist to split a variable into two parts: variable declaration and variable initialization.

Quick Assist appears in the following contexts.



Context	Example
Local variables inside a function	<p>If you have a function, as follows:</p> <pre>public function TestAS() {     var i:int=10; }</pre> <p>To split the variable using Quick Assist, place your cursor anywhere within the variable declaration <code>var i:int=10;</code> and press Control+1. Then, select the option to split the variable declaration.</p> <p>The variable is split as follows:</p> <pre>public function TestAS() {     var i:int;     i=10; }</pre>
Multiple variables inside a function	<p>If you have a function as follows:</p> <pre>public function TestAS() {     var i:int=10, j:int=20; }</pre> <p>To split the variables using Quick Assist, place your cursor anywhere within the variable declaration <code>var i:int=10, j:int=20;</code> and press Control+1. Then, select the option to split the variable declaration.</p> <p>You can split the variable as follows:</p> <pre>public function TestAS() {     var i:int, j:int=20;     i=10; }</pre> <p>You can further split the variable <code>j:int=20;</code> by placing your cursor anywhere within the variable declaration, and press Control+1. Then, select the option to split the variable declaration.</p> <p>The variable is split as follows:</p> <pre>public function TestAS() {     var i:int, j:int;     j=20;     i=10; }</pre>

### Assign to Variable

When you evaluate an ActionScript expression, if the expression returns a value, you can invoke Quick Assist to create a variable for that expression. Quick Assist is not available for methods that do not have a return type or if the return type is `void`.

The name of the variable that is created is derived from the name of the function or identifier in the expression. If the derived name exists, then the name is incremented.

For example, if you have code as follows:

```
var i:int;
i;
```

Place your cursor after `"i"`, and press Control+1. Then, select `Assign Statement To A New Local Variable`. A variable `"i2"` is created. Because the variable `i` is already present, the new variable name is incremented to `"i2"` as follows:

```
var i:int;
var i2:int = i;
```

For example, Assign to Variable appears in the following contexts:

Context	Example
Literal Expression	<p>Quick Assist supports a wide range of simple and complex expressions.</p> <p>For example, if you have an expression as follows:</p> <pre>100+150</pre> <p>Place your cursor anywhere within the expression, and press Control+1, then you get code like the following:</p> <pre>var number2:Number = 110 + 500;</pre>
Method Call	<p>If you have the following code within a method:</p> <pre>var ac:ArrayCollection = new ArrayCollection(); ac.createCursor();</pre> <p>Place your cursor within <code>ac.createCursor()</code>; and press Control+1. Then, select the option to assign the statement to a new local variable. The local variable is created as follows:</p> <pre>var createCursor:IViewCursor = ac.createCursor();</pre>
Property Access	<p>If you have the following code within a property:</p> <pre>var ac:ArrayCollection = new ArrayCollection(); ac.source;</pre> <p>Place your cursor anywhere within <code>ac.source</code>, and press Control +1. Then, select the option to assign the statement to a new local variable. The local variable is created as follows:</p> <pre>var source:Array = ac.source;</pre>

### Convert Local Variable to Field

When you have local variables inside a function, Quick Assist lets you create a field in the class.

For example, if you have a variable inside a function, as follows:

```
var i:int = 10;
```

Place your cursor anywhere within the variable definition, and press Control+1. Then, select Convert Local Variable To Field. The class field is created as follows:

```
private var i:int;
```

The name of the new field is the same name as the local variable, provided there are no conflicts in the class scope. If the name exists, then the field name is incremented, and you can rename the variable or method in your file.

### Generate Getter/Setter

Quick Assist lets you generate getters and setters (get and set accessor functions) for class variables.

For example, if you have code like the following:

```
private var result:ResultEvent;
```

Place your cursor in the variable, `result`, and press Control+1. Then, select the Quick Assist option to create the getter and setter for `result`. The Generate Getter/Setter dialog box appears. Using this dialog box, you can specify a new getter and setter function. For more information on generating getter and setters, see [“Generate get and set accessor functions”](#) on page 66.

### Generate functions

Quick Assist lets you generate special functions such as `labelFunction`, `iconFunction`, and such.

For example, to create a `labelFunction` for the following code:

```
<mx:DataGrid labelFunction="lblfunc" dataTipFunction="tipFunc" />
```

Place your cursor within `"lblfunc"`, and press **Control+I**. Then, select the Quick Assist option to create the `labelFunction`. Stub code for the function is generated as follows:

```
protected function lblfunc(item:Object, column:DataGridColumn):String
{
    // TODO Auto-generated method stub
}
```

### Generate event handlers from MXML code

Quick Assist lets you generate event handlers with custom handler names within MXML code.

For example, to generate an event handler on clicking an `<mx:Button>` tag, enter the following code:

```
<mx:Button click="clickHandler" />
```

Then, place your cursor within `"clickHandler"`, press **Control+I**, and select **generate event handler**. The event handler and stub code are generated as follows:

```
<mx:Button click="clickHandler(event)" />

protected function clickHandler(event:MouseEvent):void
{
    // TODO Auto-generated method stub
}
```

You can also generate an event handler for the following code, by pressing **Control+I** anywhere within `"clickHandler(event)"`.

```
<mx:Button click="clickHandler(event)" />
```

To customize the predefined stub code that Flash Builder generates, see [“Code templates”](#) on page 52.

## Override or implement methods

Flash Builder provides you with an option to select and override parent class methods or implement interface methods.

- 1 Open the **Override/Implement Methods** dialog box by selecting **Override/Implement Methods** from the **Source** menu. You can also select **Source > Override/Implement Methods** from the context menu of the MXML or **ActionScript** editor.
- 2 The methods for each parent class are displayed in a tree structure format. For each class, you can select the methods that you want to override and the methods that you want to implement.
- 3 You can select the insertion point to insert the selected methods. The default insertion point option depends on where you place your cursor in the editor while opening the **Override/Implement Methods** dialog box. The variable or method closest to the cursor location appears as an insertion point option.

Flash Builder generates the stub code for the selected methods.

To customize the predefined stub code that Flash Builder generates, see [“Code templates”](#) on page 52.

# Code templates

Code templates speed-up your coding efforts by letting you auto-insert frequently used coding patterns.

Flash Builder includes a number of predefined code templates. You can also define additional code templates for commonly used code patterns. To see all the available code templates, open the Preferences dialog box, and select Flash Builder > Editors > Code Templates.



Adobe Community Professional, Paul Robertson, blogged about [using code templates](#).

## MXML, ActionScript, and CSS code templates

The ActionScript, CSS, and MXML code templates are context-based and can be called on pressing Control+Space. You can use the pre-defined templates that are shipped with Flash Builder or you can create your own.

### Insert code templates

To insert a code template in the code editor, type the name of the template in the code editor, and press Control+Space.

For example, when writing ActionScript code, say, you use the `for` loop repeatedly. You can then define a code template for the `for` loop as follows:

```
for (var i:int = 0; i < array.length; i++) { }
```

When you use a code template, you don't have to type the complete code for the `for` loop. Instead, in the ActionScript class, type `'for'` and press Control+Space. A template option to create the `for` loop appears. On selecting the code template, the code that you defined in the template is inserted.

Templates can also contain template variables. A template variable is defined within `${ }`. The variable is resolved based on the corresponding variable definition in the editor.

For example, if you define a code template for the `for` loop as follows:

```
for (var ${index}:int = 0; ${index} < ${array}.length; ${index}++) { ${cursor} }
```

And you invoke the code template after defining a variable `myArr` as follows:

```
{  
    var myArr:ArrayCollection = null;  
}
```

Then, `${array}` in the code template resolves to `myArr`, and the resulting code looks like:

```
{  
    var myArr:ArrayCollection = null;  
    for (var ${index}:int = 0; ${index} < myArr.length; ${index}++) { ${cursor} }  
}
```

### Create and edit code templates

- 1 Open the Preferences dialog box, and select Flash Builder > Editors > Code Templates.
- 2 Code templates are categorized as ActionScript, MXML, and CSS code templates. Under each category, you find a set of predefined code templates. You can edit an existing template or add a new one.
- 3 To add a new template, select the code template category, and click Add. In the New Template dialog box, enter a name and brief description for the code template. Then, specify a context in which the code template must be called.

You can specify contexts for ActionScript and MXML code templates.

You can select from the following ActionScript contexts:

- **ActionScript:** Inserts the code template anywhere in the ActionScript document
- **ActionScript statement:** Inserts the code template within functions and within elements of a class
- **ActionScript members:** Inserts the code template only within elements of a class
- **ActionScript Package Scope:** Inserts the code template within a package, as follows:

```
Package
{
    /* insert code template*/
}
```

You can select from the following MXML contexts:

- **MXML:** Inserts the code template anywhere in the MXML document
  - **MX Component:** Inserts the code template within MX components available for Flex 3 SDK
  - **Spark Components:** Inserts the code template within Spark components available for Flex 4 SDK
  - **MXML attributes:** Inserts the code template for MXML attributes within MX and Spark components.
- 4 Enter the code for the template in the Pattern section. To insert variables in the code, click Insert Variable, and select from a list of predefined variables. The variables are contextual to the template category.

ActionScript templates contain predefined variables that include `array`, `enclosing_method`, `enclosing_package`, `enclosing_type`, `field`, `local_var`, and `var`. MXML templates contain predefined variables that include `fx`, `mx`, `s`, and `tag`.

- 5 If you don't want Flash Builder to automatically insert the code template into your code, deselect the Automatically Insert Into Code option.
- 6 To customize an existing code template, select the template, and click Edit. After editing the template, click OK.

For more information about customizing file templates and template variables, see “[Customize file templates](#)” on page 57 and “[Template variables](#)” on page 59.

At any point, you can remove the custom template and restore the predefined code template by clicking Revert To Default.

By default, you can call all the predefined templates using Content Assist. If you, however, don't want a specific template to appear in the Content Assist options, deselect that template in the Existing Templates section.

You can also import and export code templates. You can select one or more templates and export them. The templates are exported as an XML file.



Adobe Community Professional, Paul Robertson, blogged about [sharing code templates](#).

## Flash Builder code templates

Flash Builder can automatically generate predefined code in the following scenarios:

- “[Generate event handlers](#)” on page 198
- “[Generate get and set accessor functions](#)” on page 66
- “[Generate from usage](#)” on page 64 (placeholder stub code for an undefined method)
- “[Override or implement methods](#)” on page 51

You can customize the predefined code template that Flash Builder generates.

## Customize the code template

- 1 Open the Preferences dialog box, and select Flash Builder > Editors > Code Templates > Flash Builder.
- 2 Select the name of the code template that you want to customize, and click Edit. For example, to customize the code that is generated when you generate an event handler, select the event handler template, and click Edit.
- 3 You can customize the name of the template, the description, and the code pattern.
- 4 To insert a variable within the code, click Insert Variable, and select the variable. For more information about the available code variables, see “Use code variables” on page 54.
- 5 At any point, you can discard the changes by clicking Revert To Default.
- 6 You can also import and export the code template. You can select one or more templates and export them. The templates are exported as an XML file.

## Use code variables

### Code variables for event handlers

Variable	Description	Example
<code>\${component_id}</code>	Resolves to the unique ID of the component.	If the ID of the button component is <code>test</code> , the event handler that is generated is <code>test_clickHandler</code> .  If you have not specified an ID value, the auto-generated values are <code>component1</code> , <code>component2</code> , and so on.
<code>\${component_name}</code>	Resolves to the name of the tag.	<pre><code>\${namespace} \${modifiers}function \${:method_name('\${component_id}_\${event_name}Handler ')} (\${event}:\${event_type}):\${return_type} { // TODO Auto-generated method stub \${cursor} }</code></pre>
<code>\${event_name}</code>	Specifies the name of the event.	<code>clickEvent, onHover</code>
<code>\${event_type}</code>	Resolves to the event handler type.	Flash Builder designates a default event type for each user interface component.  On generating a click event for a button component, an event handler of type <code>MouseEvent</code> is generated as follows:  <code>button1_clickHandler(event:MouseEvent)</code>
<code>\${modifiers}</code>	Specifies the modifiers for the generated function.	<code>static</code>
<code>\${method_name}</code>	Resolves to the event handler name.	For a click event for the Button component, the event handler name can be <code>button1_clickHandler</code>
<code>\${namespace}</code>	Defines the namespace value for the generated function.	The namespace value can be any of the following: <ul style="list-style-type: none"> <li>• <code>protected</code></li> <li>• <code>public</code></li> <li>• <code>private</code></li> </ul> The default value is <code>protected</code> .

Example using code variables for event handler functions:

```

${namespace} ${modifiers}function
${:method_name('$_{component_id}_${event_name}Handler')}
${event}:${event_type}):${return_type}
{
    // TODO Auto-generated method stub
    ${cursor}
}

```

### Code variables for get and set accessor functions

Variable	Description	Example
\${metadata}	Specifies the metadata tags that are generated	Generating get and set accessor functions for a Bindable variable
\${asdoc}	Specifies the ASDoc that are generated for get and set accessor functions	<pre> \${metadata} \${asdoc}\${namespace} \${modifiers}function get \${method_name}() \${return_type} {     return \${property}; } </pre>
\${return_type}	Resolves to the variable type.  If the variable type is not specified, the generated get accessor function does not have a return type.	<pre> \${metadata} \${asdoc}\${namespace} \${modifiers}function get \${method_name}() \${return_type} {     return \${property}; } </pre>
\${property}	Resolves to the property name in the getter/setter dialog.	For a variable <code>var i:int</code> , <code>i</code> resolves to <code>_i</code>
\${argument_type}	Resolves to the data type of the generated set function.	

Example using code variables for get and set accessor functions:

```

${asdoc}
${namespace} ${modifiers}function set ${method_name}(value:${argument_type}):void
{
    if( ${property} !== value)
    {
        ${property} = value;
        dispatchEvent(new Event("${event_name}"));
    }
}

```

### Code variables for functions in an undefined class

Variable	Description	Example
\${params}	For an undefined function that accepts a specified number of arguments, the generated function has the same number of arguments with the type.	

## Metadata code completion

Flash Builder displays code completion hints for metadata that you use in your MXML and ActionScript documents.

In an MXML document, metadata code hints are displayed within embedded `<fx:Metadata>` and `<fx:Script>` tags. In an ActionScript document, metadata code hints are also displayed for ActionScript language elements like class names, variables, getters, and setters.

The code hints are contextual to the MXML and ActionScript document, and the code in which the metadata is used. For example, when you invoke Content Assist within two blank lines of an ActionScript statement, code hints applicable to only that ActionScript statement appear. To see all the valid code hints applicable to the ActionScript or MXML document, press Control+Space multiple times to cycle through the available code hints.

## Use metadata code completion in MXML documents

In an MXML document or class, you can use metadata code completion in the following ways:

- Enter '[' within `<fx:Metadata>` tags, as follows:

```
<fx:Metadata>
    [
</fx:Metadata>
```

- Enter '[' within `<fx:Script>` tags, as follows:

```
<fx:Script>
    <![CDATA[
        [
    ]]>
</fx:Script>
```

## Use metadata code completion in ActionScript documents

In an ActionScript document, you can use metadata code completion when you enter '[' before a class name, variable, getter, or setter, as follows:

```
[
    class GetSet
    {
        [
            private var privateProperty:String;
            [
                public function get publicAccess():String
                {
                    return privateProperty;
                }
            [
                public function set publicAccess(setValue:String):void
                {
                    privateProperty = setValue;
                }
            ]
        ]
    }
}
```

## Code completion for custom metadata tags

Flash Builder supports code completion for custom metadata tags that are introduced on using third-party Flex frameworks.



To enable code hints for custom metadata tags in your project, you generate a SWC file containing a `metadata.xml` file, as follows:

- 1 Create a library project. The New Flex Library Project wizard guides you through the steps, prompting you for the project name, location, and build path information. For more information, see [“Create Flex library projects”](#) on page 213.

- 2 Add the `metadata.xml` file in the `src` folder under the root folder of your library project. Include all the metadata tags that you want in the `metadata.xml` file.

Add the `metadata.properties` file (if any) in the appropriate locale folder. For example, `locale/en_US` or `locale/ja_JP`.

For more information about metadata tags, see [About metadata tags in the Flex documentation](#).

- 3 Include the `metadata.xml` file in the library SWC file, as follows:

- a Select Project > Properties > Flex Library Build Path.

The `metadata.xml` file that you added appears in the Assets tab.

- b Select the `metadata.xml` file to include in the SWC file, and click OK.

The SWC file is compiled and generated in the output (bin) folder of the library project.

- 4 Select the locale folder to which you added the `metadata.properties` file (if any).

- 5 After generating the SWC file, add it to the build path of your project, as follows:

- 1 Select Project > Properties > Flex Build Path.

- 2 Click Add SWC.

- 3 Enter or browse to the location of the SWC file, and click OK.

Once you add the SWC file to your build path, metadata code completion hints appear for the metadata tags defined in the `metadata.xml` file. You can share the SWC file between your applications or distribute to other developers.

## Customize file templates

Flash Builder allows you to customize the default information contained in new MXML, ActionScript, and CSS files. Examples of information you can specify include variables for specifying author and date, variables for opening and closing tags and attributes, variables for various ActionScript declarations, namespace prefixes, and just about any content you want to include in a template file. File templates are especially useful for specifying introductory comments and copyright information.

The content of a new file is specified in a file template available from Preferences > Flash Builder > File Templates. Templates are available for the following types of files:

ActionScript	ActionScript file ActionScript class ActionScript interface ActionScript skinnable component
MXML	MXML web application MXML desktop application MXML component MXML module MXML skin ItemRenderer for Spark components ItemRenderer for MX components ItemRenderer for MX DataGrid ItemRenderer for Advanced DataGrid ItemRenderer for MX Tree
FlexUnit	FlexUnit TestCase class FlexUnit TestSuite class FlexUnit4 TestCase class FlexUnit4 TestSuite class
CSS	CSS file

After modifying a template, you can export the template so it can be shared with other members of your team.

### Modify a file template

- 1 Select Preferences > Flash Builder > File Templates
- 2 Expand the file categories and select a file template to modify.
- 3 Select Edit and modify the template.

You can type directly in the Template editor or select Variables to insert pre-defined data into the template.

- 4 Click OK to save the changes.

Changes apply to new files.

### Export and Import File Templates

- 1 Select Preferences > Flash Builder > File Templates
- 2 Expand the file categories and select a file template.
- 3 Select Export to export a template to the file system, or Import to import a previously exported template.

Templates are exported as XML files.

### Restore defaults

**Note:** Restoring defaults restores all file templates to the default values. You cannot restore a single template to the default value.

- ❖ To restore the default templates, open Preferences > Flash Builder > File Templates and select Restore Defaults

## Template variables

### Template variables for all file types

Variable	Description	Example
\${date}	Current date	Feb 15, 2009
\${year}	Current year	2009
\${time}	Current time	3:15 PM
\${file_name}	Name of the newly created file	HelloWorld.mxml
\${project_name}	Name of the Flex or ActionScript project	Hello_World_Project
\${user}	Username of the author	jdoe
\$\$	Dollar symbol	\$
\$(dollar)		

### Template variables for MXML files

Variable	Description	Example
\${application} \${component} \${module}	<p>Specifies the application, component, or module MXML tag names.</p> <p>For a web application, \${application} expands to "Application."</p> <p>For a desktop application, \${application} expands to "WindowedApplication."</p> <p>\$(component) expands to "Component."</p> <p>\$(module) expands to "Module."</p> <p>These tags are typically used to position the starting and closing tags of a file.</p>	<p>The following:</p> <pre>&lt;\${application} \${xmlns}\${wizard_attributes}\${min_size}&gt; \${wizard_tags}  &lt;/\${application}&gt;</pre> <p>expands to:</p> <pre>&lt;s:Application xmlns:fx="http://ns.adobe.com/mxml/2009" xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024" minHeight="768"&gt; &lt;s:layout&gt; &lt;s:BasicLayout/&gt; &lt;/s:layout&gt; &lt;/s:Application&gt;</pre>
\$(xml_tag)	XML version	<?xml version="1.0" encoding="utf-8"?>
\$(xmlns)	Resolves to the namespace definition, based on the project's Flex SDK type and the namespace prefix defined in Preferences.	For a Flex 4 SDK project: xmlns="http://ns.adobe.com/mxml/2009"
\$(min_size)	Minimum size of an MXML web application.	minWidth="1024" minHeight="768"
\$(ns_prefix)	Namespace prefix for the project's Flex SDK.  You cannot change the default values for this variable.	For Flex 3: mx:  For Flex 4: fx:
\$(wizard_attributes)	Specifies the position of the attributes defined by the New File wizard.	For a new web application:  \${application} \${xmlns}\${wizard_attributes}>  expands to:  <Application xmlns="http://ns.adobe.com/mxml/2009" layout="vertical">

Variable	Description	Example
<code>\${wizard_tags}</code>	Specifies the layout property for containers defined by the New <i>File</i> wizard	For a new application using Flex 4 SDK:  <pre>&lt;s:layout&gt; &lt;s:BasicLayout/&gt; &lt;/s:layout&gt;</pre>
<code>\${fx}</code>	Prefix for the MXML 2009 language document namespace. The prefix is as specified in the MXML document.	When you use the following Library tag template in an MXML document:  <pre>&lt;\${fx}Library&gt;   &lt;\${fx}Definition id="\${def}"&gt;     \${cursor}   &lt;/\${fx}Definition&gt; &lt;/\${fx}Library&gt;</pre> <p>A library tag is created as follows:</p> <pre>&lt;fxLibrary&gt; &lt;fxDefinition id="def"&gt; &lt;/fxDefinition&gt; &lt;/fxLibrary&gt;</pre>
<code>\${mx}</code>	Prefix for the MX document namespace. The prefix is as specified in the MXML document.	When you use the following Combobox template in the MX Components context:  <pre>&lt;\${mx}ComboBox id="\${comboBox}" rowCount="\${rowCount:values(5)}" dataProvider="\${dataProvider}"/&gt; \${cursor}</pre> <p>A combobox is created as follows:</p> <pre>&lt;mx:ComboBox id="comboBox" rowCount="5" dataProvider="dataProvider"/&gt;</pre>
<code>\${s}</code>	Prefix for the Spark document namespace	When you use the Spark Button template in the Spark Components context:  <pre>&lt;\${s}Button id="\${btn}" label="\${myButton}" click="\${onClick} (\${event})"/&gt; \${cursor}</pre> <p>A Spark button is created as follows:</p> <pre>&lt;s:Button id="btn" label="myButton" click="onClick(event)"/&gt;</pre>
<code>\${tag}</code>	Fully-qualified tag name for the project's MX components.	When you use the List template in the MXML context:  <pre>&lt;\${list:tag}(mx.controls.List) id="\${myList}"&gt; &lt;\${dp:tag}(dataProvider)&gt;  &lt;\${arraycollection:tag}(mx.collections.ArrayCollection)&gt;   \${cursor} &lt;/\${arraycollection}&gt; &lt;/\${dp}&gt; &lt;/\${list}&gt;</pre> <p>A list is created as follows:</p> <pre>&lt;s:List id="myList"&gt; &lt;s:dataProvider&gt; &lt;s:ArrayCollection&gt; &lt;/s:ArrayCollection&gt; &lt;/s:dataProvider&gt; &lt;/s:List&gt;</pre>

## Template variables for ActionScript files

Variable	Description	Example
<code>\${package_declaration}</code>	Generates the package declaration.	For a file in the com/samples package, generates: package com.samples
<code>\${import_declaration}</code>	For a new ActionScript class or ActionScript Interface, generates required import declarations .	For a subclass of TextBox, generates: import flex.graphics.TextBox;
<code>\${interface_declaration}</code>	For a new ActionScript interface, generates the interface declaration.	For a new Interface that extends IButton interface, generates: public interface IMyButton extends IButton
<code>\${class_declaration}</code>	For a new ActionScript class, generates the class declaration.	For a new subclass of CheckBox, generates: public class MyCheckBox extends CheckBox
<code>\${class_body}</code>	Generates all the required statements for a new class.	For a new subclass of Button that implements the IBorder interface, generates the following for the class body:  <pre>public function MyButton() {     super(); } public function get borderMetrics():EdgeMetrics {     return null; }</pre>
<code>\${interface_name}</code> <code>\${class_name}</code> <code>\${package_name}</code>	Specifies the interface, class, or package name.  Typically used when generating comments.	For example, the following template specification:  <pre>/*  * \${class_name} implements. . .  */</pre> generates the following code:  <pre>/*  * MyButton implements. . .  */</pre>
<code>\${array}</code>	Specifies the value of an array	The following Fore template  <pre>for each (var \${index}:\${type} in \${array}) {     \${line_selection}     \${cursor} }</pre> iterates over the value of an array using enumeration as follows:  <pre>for each (var i:type in array) { }</pre>
<code>\${enclosing_method}</code>	Specifies the enclosing method's name	The traceMethod template traces the method as follows:  <pre>trace("\${enclosing_type}.\${enclosing_method}");</pre>

Variable	Description	Example
<code>\${enclosing_package}</code>	Specifies the package name such as 'xx.yy' of 'xx.yy.class'	The package template creates a package as follows:  <pre>package \${enclosing_package} { /**  * @author \${user}  */ class \${enclosing_type} { \${cursor} } }</pre>
<code>\${enclosing_type}</code>	Specifies the type name such as "class" of "xx.yy.class"	The package template creates a package specifying the class name as follows:  <pre>package \${enclosing_package} { /**  * @author \${user}  */ class \${enclosing_type} { \${cursor} } }</pre>
<code>\${field}</code>	Specifies the class variables	The do template creates the do-while loop as follows:  <pre>do {     \${line_selection}     \${cursor} } while (\${condition:local_var(Boolean)});</pre>
<code>\${local_var}</code>	Specifies the local variable visible within the block scope	if template creates an if statement as follows:  <pre>if (\${condition:local_var(Boolean)}) { \${cursor} }</pre>
<code>\${var}</code>	Specifies all the visible variables	fori template iterates over the value of an array as follows  <pre>for (var \${index}:int = 0; \${index} &lt; \${array}.length; \${index}++) { \${cursor} }</pre>

## Template variables for CSS files

Variable	Description	Example
<code>\${css_namespaces}</code>	Defines namespaces for Spark and Halo style selectors.	<p>Default values for Flex 3:</p> <pre>" "</pre> <p>(In Flex 3, namespace declarations are not required in CSS files)</p> <p>Default values for Flex 4:</p> <pre>@namespace s "library://ns.adobe.com/flex/spark"; @namespace mx "library://ns.adobe.com/flex/halo";</pre>

## Template file examples

The following shows an example of an MXML Component file template, followed by a new MXML Component file generated from the template.

### Example File Template for an MXML Component file

```

${xml_tag}
<!--
* ADOBE SYSTEMS Confidential
*
* Copyright ${year}. All rights reserved.
*
* ${user}
* ${project_name}
* Created ${date}
*
-->
<${component} ${xmlns}${wizard_attributes}>
    ${wizard_tags}

    <${ns_prefix}Script>
    <![CDATA[

        ]]>
    </${ns_prefix}Script>
</${component}>

```

### New MXML Component file generated from the example template

```
<?xml version="1.0" encoding="utf-8"?>
<!--
* ADOBE SYSTEMS Confidential
*
* Copyright 2009. All rights reserved.
*
* jdoe
* FileTemplates
* Created Jul 13, 2009
*
-->

<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo" width="400" height="300">
    <s:layout>
        <s:BasicLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[

            ]]>
        </fx:Script>
</s:Group>
```

## Generate from usage

Use Quick Assist to generate stub code for an undefined method, variable, or class in your code. The generated stub code can be used as a placeholder for the code that you want to implement later without rendering your code incompatible. To customize the predefined stub code that Flash Builder generates, see [“Code templates”](#) on page 52.

To invoke Quick Assist, you can use the keyboard shortcut Control+I (Windows) or Command+I (Mac OS).

Use Quick Assist to generate stub code in the relevant class or MXML script block by selecting any of the following actions:

**Generate Method** Creates a method

For example, if you have code as follows:

```
private function genFunc():void
{
    bar();
}
```

Place your cursor anywhere in the line of code containing `bar()`; and press Control+I. An option to create a function appears. On selecting this option, a new function is generated as follows:

```
private function bar():void{}
```

You can also generate a function for an undefined function in a referenced class. For example, if you have an undefined function `setValue()` in the referenced class `MyClass`, as follows:

```
MyClass cls = new MyClass();
cls.setValue(5);
```



Place your cursor anywhere in the line of code containing "setValue" and press Control+1. An option to create a function appears. On selecting this option, a new function `setValue(int)` is generated in the referenced "MyClass" as follows:

```
private function setValue(param0:int):void
{
    // TODO Auto Generated method stub
}
```

**Generate Variable** Creates a property

For example, if you have code, where, `i` is an undefined variable, as follows:

```
public function MyClass
{
    i;
}
```

Place your cursor anywhere in the line of code containing `i`; and press Control+1. You get options to create a local variable or a field.

Selecting the option to create a local variable, creates the variable as follows:

```
var i:Object;
```

Selecting the option to create a field, creates a class-level variable as follows:

```
private var i:Object;
```

You can also generate a property for an undefined variable in a referenced class. For example, if you have an undefined variable "aValue" in the referenced class "MyClass", as follows:

```
MyClass cls = new MyClass();
cls.aValue = "str";
```

Place your cursor anywhere in the line of code containing `aValue`, press Control+1, and select Create Field Variable. A property `aValue` of type string is generated in the referenced "MyClass" as follows:

```
private var aValue:String;
```

**Generate Class/Interface** Creates a class or interface

For example, if you have code, where `Foo` is undefined, as follows:

```
public function myFunction():Foo;
{
}
```

Place your cursor anywhere in the line of code containing `Foo`; and press Control+1. Options to create a class or interface named `Foo` appear. Select one of these options to open either the New ActionScript Class wizard or New ActionScript Interface wizard. Enter the necessary details and click Finish. After clicking Finish, a class or interface with the name `Foo` is created.

When you generate a new class, you can create an ActionScript class with a parametrized constructor.

For example, if you have code like the following:

```
Private function func(): void {
    New NewClass("str1");
}
```

Place your cursor anywhere in the line of code containing `NewClass("str1")`, press `Control+1`, and select **Create Class**. An `ActionScript` class with a parametrized constructor is created. If you, however, specify a superclass for the `ActionScript` class, then a parameterized constructor is not generated.

Adobe Community Professional, Paul Robertson, blogged about [using Quick Assist for external classes and interfaces](#).

**Generate Event Handler** Generates event handler functions

For example, if you have code like the following:

```
public function MyClass
{
    Var button:Button = new Button();
    button.addEventListener(DragEvent.DRAG, dragHandlerFunction);
}
```

Place your cursor anywhere in the line of code containing `dragHandlerFunction`, and press `Control+1`. Then, select the **Quick Assist** option to create the event handler. The event handler function is created as follows:

```
protected function dragHandlerFunction (event:DragEvent):void
{
}
```

**Generate Import Statement From Usage** Creates an import statement

For example, if you have code where the variable type `Button` is undefined, as follows:

```
<fx:Script>
    <![CDATA[
        var btn:Button;
    ]]>
</fx:Script>
```

Place your cursor anywhere in the line of code containing `var btn:Button`, and press `Control+1`. You get an option to import `Button` if a class named `Button` is available in the project. The import statement is created as follows:

```
import spark.components.Button;
```

You can generate import statements for function arguments, function return types, and such.

## Generate get and set accessor functions

Get and set accessor functions (getters and setters) let you keep class properties private to the class. They allow users of the class to access those properties as if they were accessing a class variable (rather than calling a class method).

Flash Builder can generate `ActionScript` get and set accessor functions for class variables. You can select a bindable property and generate get and set accessor functions for that property. You can also specify a custom event name at the time of generating code.

### How to generate get or set accessor functions

- 1 With an `ActionScript` file open in the Source Editor, place the cursor on a class variable.
- 2 Select **Source > Generate Getter/Setter** from either the Flash Builder menu or the context menu.
- 3 In the **Generate Getter/Setter** dialog, specify details for the accessor functions and click **OK**.

**Note:** To view the code that is generated, select **Preview** before clicking **OK**.

To customize the predefined code that Flash Builder generates, see [“Code templates”](#) on page 52.

When you generate getters and setters, Flash Builder provides the following options:

- Make the class variable private.  
Typically, class variables have private access.
- Rename the class variable, suggesting a leading underscore for the variable name.  
By convention, private class variables have a leading underscore.
- Rename the accessor functions.
- Specify a bindable property and a custom event name.

When you specify a bindable property, a `[Bindable]` tag is defined above the generated accessor function in the generated code.

- Specify whether to generate both getter and setter accessor functions.
- Specify the namespace value for the accessor function.
- Specify the placement of the accessor function in any of the following locations:
  - Before the first method
  - After the last method
  - Before variable declarations
- Preview the code that is generated.

For more information about get and set accessor functions, see [Get and set accessor methods](#) in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

## Generate get or set accessor functions

# Syntax error checking

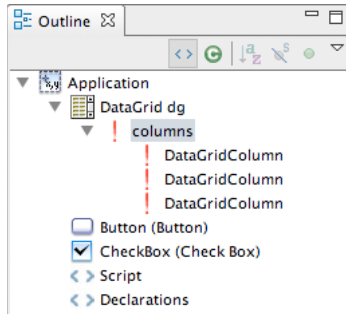
As you enter code, Flash Builder compiler identifies syntax errors and reports them so that you can correct them before running the application. You can easily adjust syntax coloring preferences.

When code syntax errors are encountered, you are notified in the following ways:

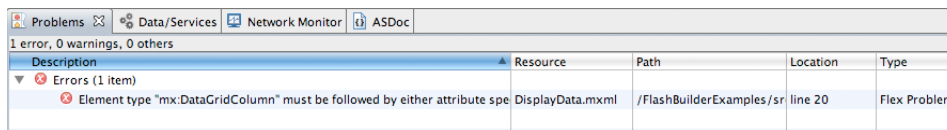
- An error indicator is added next to the line of code, as the following example shows:

```
17 </fx:Declarations>
18 <mx:DataGrid x="33" y="22" id="dg">
19 <mx:columns>
20 <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
21 <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
22 <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
23 </mx:columns>
24 </mx:DataGrid>
25 <s:Button x="365" y="30" label="Button" click="button1_clickHandler(event)"/>
26 <s:CheckBox x="365" y="71" label="Check Box"/>
27 </s:Application>
```

- The Outline view indicates the error with an exclamation mark in the affected lines of code, as the following example shows:



- The Problems view lists an error symbol and message. Double-clicking the error message locates and highlights the line of code in the editor, as the following example shows:



Coding syntax errors are identified when your projects are built. If you do not fix syntax errors before you run your application, you are warned that errors exist. Depending on the nature and severity of the errors, your application might not run properly until the errors are corrected.

## Apply syntax coloring preferences

- ❖ Open the Preferences dialog and select Flash Builder > Editors > Syntax Coloring.

Default font colors can also be configured on the Text Editors and Colors and Fonts Preferences pages (see Preferences > General > Appearance > Colors and Fonts. See also Preferences > General > Editors > Text Editors).

## Unidentified reference error highlighting

As you type ActionScript code, Flash Builder automatically generates error annotations for undefined identifiers in your code. The error annotations are indicated by .

You can quickly identify undefined variables and methods before you save the file and the compiler generates the error.

Unidentified reference error highlighting also helps you identify places in the code where you can generate stub code for an undefined method, variable, or class. For more information, see [“Generate from usage”](#) on page 64.

Flash Builder turns on unidentified reference error highlighting, by default. To turn it off, select Flash Builder > Editors in the Preferences dialog box, and deselect Report Problems As You Type.

## Find references and refactor code

Flash Builder includes advanced search features that are more powerful than find and replace. To help you understand how functions, variables, or other identifiers are used, Flash Builder lets you find and mark references or declarations to identifiers in ActionScript and MXML files, projects, or workspaces. You can use refactor to rename the following identifiers in your code, and then update all references to them:

- Variables
- Functions
- Types (interface, class)
- Accessors (getter/setter)
- Attributes
- Metadata in MXML (effects, events, styles)

### Mark references

- 1 In Source mode, click the Mark Occurrences button in the toolbar.
- 2 Click an identifier in the editor. All instances are marked, depending on settings in Preferences.

To change the appearance of marked references, in the Preferences dialog, select General > Editors > Text Editors > Annotations. For more information on Markers, see [“Use markers”](#) on page 77.

### Find all references or declarations

- 1 In Source mode, click an identifier in the editor.
- 2 Select Search > References or Search > Declarations from the main menu. Then select File, Project, or Workspace. Matches appear in the Search view.

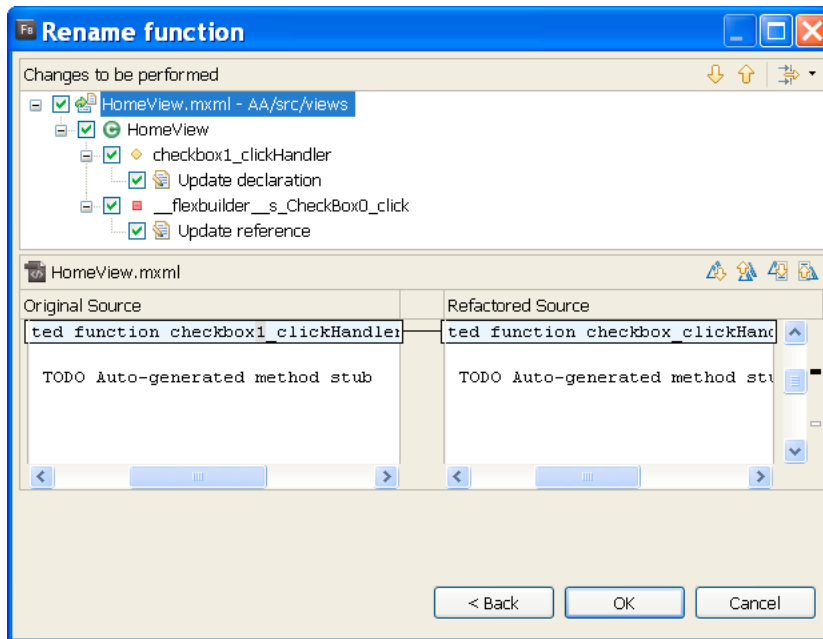
### Refactor your code

- 1 In Source mode, click an identifier in the editor.
- 2 Select Source > Refactor > Rename from the main menu.
- 3 Enter a new name.

Flash Builder checks rename preconditions and prompts you to confirm problems before the rename operation occurs. Preconditions include the following:

- References cannot be renamed in read-only files.
- All files must be saved.
- If a project has build errors, a warning appears.
- The new name must be within scope, which is determined by the type of element and its location. Name-shadowing errors are also noted.
- The new name must be a valid identifier.
- The reference defined in a SWC file must include a source attachment.

- 4 To review the change, click Preview to see the original and refactored source, or click OK to proceed with the change to your code.



### Refactoring in CSS files

When you have references to ActionScript or MXML files in your CSS files, and you rename or move the ActionScript or MXML files to a different location. Then, Flash Builder automatically updates the CSS files with references to the new name or location. You can review the changes in the Preview dialog, and click OK to proceed with the change to your code.

## Format, navigate, and organize code

The Flash Builder editors provide many shortcuts for navigating your code, including folding and unfolding code blocks, opening the source of code definitions, and browsing and opening types. Code navigation includes the ability to select a code element (a reference to a custom component in an MXML application file, for example) and go to the source of the code definition, wherever it is located in the project, workspace, or path.

Multiple line code blocks can be collapsed and expanded to help you navigate, view, and manage complex code documents. In Flash Builder, expanding and collapsing multiple-line code statements is referred to as *code folding* and *unfolding*.

### Code formatting, indenting, and commenting

As you write code, Flash Builder automatically indents lines of code to improve readability, adds distinguishing color to code elements, and provides many commands for quickly formatting your code as you enter it (adding a block comment, for example).

To change the default formatting, in the Preferences dialog box, select Flash Builder > MXML Code > Formatting. You can change the order and grouping of the attributes.

When you paste MXML or ActionScript code into the code editor, Flash Builder automatically indents the code according to your preferences. You can also specify indenting for a selected block of code.

To specify the indentation preferences, in the Preferences dialog, select Flash Builder > Editors. You can specify the indent type and size.

## Set, fold, and unfold code blocks

- 1 In the editor, click the fold symbol (-) or the unfold symbol (+) in the editor's left margin.

```
10 public function set cartItems(items:ShoppingCart):Void {  
11     _cartItems = items;  
12     dg.dataProvider = _cartItems.items;  
13 }
```

Folding a code block hides all but the first line of code.

```
10 public function set cartItems(items:ShoppingCart):Void {  
14 }
```

Unfolding the code block to make it visible again. Hold the mouse over the unfold (+) symbol to show the entire code block in a tool tip.

```
10 public function set cartItems(items:ShoppingCart):Void {  
14     _cartItems = items;  
15     dg.dataProvider = _cartItems.items;  
16 }  
17  
18 if (_cartItems.items.length == 0)
```

- 2 By default, code folding is turned on in Flash Builder. To turn off code folding, open the Preferences dialog and select Flash Builder > Editors, and then deselect the Enable Code Folding option.

## Indent code blocks

The editor automatically formats the lines of your code as you enter it, improving readability and streamlining code writing. You can also use the Tab key to manually indent individual lines of code.

When you copy and paste code blocks into Flash Builder, Flash Builder automatically indents the code according to your indentation preferences.

If you want to indent a block of code in a single action, you can use the Shift Right and Shift Left editor commands.

### Shift a code block to the left or right

- 1 In the editor, select a block of code.
- 2 Select Source > Shift Right or Source > Shift Left.
- 3 Press Tab or Shift Tab to indent or unindent blocks of code.

### Set indent preferences

- 1 Open the Preferences dialog and select Flash Builder > Indentation.
- 2 Select the indent type (Tabs or Spaces) and specify the IndentSize and Tab Size.

## Add comments and comment blocks

You can add or remove comments using options in the Source menu or by using keyboard shortcuts. You can add the following types of comments:

- Source comment for ActionScript (`//`)
- Block comment for ActionScript (`/* */`)
- ASDoc comments for ActionScript (`/** */`)
- Block comment for MXML (`<!-->`)
- CDATA block for MXML (`<![CDATA[ ]]>`)

Comments in ActionScript code can be toggled on or off.



Adobe Community Professional, Paul Robertson, blogged about [using block selection mode](#).

### Toggle comments in ActionScript code

- 1 In the editor, select one or more lines of ActionScript code.
- 2 Press Control+Shift+C (Windows) or Command+Shift+C (Mac OS) to add, or remove, C-style comments.
- 3 Press Control+/ (Windows) or Command+/ (Mac OS) to add, or remove, C++ style comments.

### Add XML comments in MXML code

- 1 In the editor, select one or more lines of MXML code.
- 2 Press Control+Shift+C (Windows) or Command+Shift+C (Mac OS) to add a comment.

### Add CDATA blocks in MXML code

- 1 In the editor, select one or more lines of MXML code.
- 2 Press Control+Shift+D (Windows) or Command+Shift+D (Mac OS) to add a comment.

## Navigate and inspect code

With applications of any complexity, your projects typically contain many resources and many lines of code. Flash Builder provides several features that help you navigate and inspect the various elements of your code.

### Open code definitions

Flash Builder lets you open the source of an external code definition from where it is referred to in your code. For example, if you create a custom MXML component and import it into your MXML application you can select the reference to the MXML component and open the source file in the editor.

#### Open the source of a code definition

- 1 Select the code reference in the editor.
- 2 From the Navigate menu, select Go To Definition.

You can use the keyboard shortcut, F3.

The source file that contains the code definition opens in the editor.

Flash Builder also supports hyperlink code navigation.



### Open the source of a code definition using hyperlink navigation

- 1 Locate the code reference in the editor.
- 2 Press and hold the Control key (Windows) or Command key (Mac OS) and hold the mouse over the code reference to display the hyperlink.
- 3 To navigate to the code reference, click the hyperlink.

### Use the Outline view

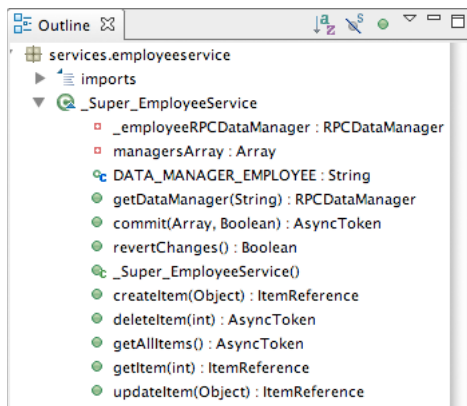
The Outline view is part of the Flash Development perspective (see [“The Flash Development perspective”](#) on page 7), and, therefore, is available when you edit code and design your application. You use the Outline view to more easily inspect and navigate the structure of your MXML, ActionScript, and CSS documents.

The Outline view contains three modes: Class, MXML, and CSS. In Class mode, the Outline view displays the structure of your code (classes, member variables, functions, and so on). In MXML mode, the Outline view displays the MXML structure (tags, components, controls, and so on). In CSS mode, CSS selectors and nested properties within them are displayed.

Selecting an item in the Outline view locates and highlights it in the editor, which makes it much easier to navigate your code.

### Outline view in Class mode

When you edit an ActionScript document (or ActionScript contained in an MXML document), the Outline view displays the structure of your code. This includes import statements, packages, classes, interfaces, variables not contained in functions, and functions. This view does not include metadata, comments, namespace declarations, and the content of functions.



In the Outline view, nodes and items in the tree structure represent both the different types of language elements and their visibility. For example, red icons indicate private elements, green indicates public elements, and yellow indicates that the element is not private or public.

### Outline view toolbar in Class mode

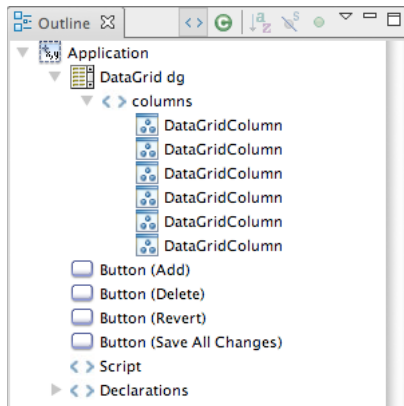
In Class mode, the Outline view toolbar contains the sort and filter commands, as the following example shows:



### Outline view in MXML mode

When you edit an MXML document, which can contain both MXML and ActionScript code, both the Class and MXML modes are available in the Outline view.

In MXML mode, each item in the Outline view represents an MXML tag and the following types of tags are displayed: components, controls, nonvisual tags such as `WebService` or `State`, component properties that are expressed as child tags (layout constraints, for example), and compiler tags such as `Model`, `Array`, and `Script`.



The Outline view in MXML mode does not show comments, CSS rules and properties, and component properties expressed as attributes (as opposed to child tags, which are shown).

### Outline view toolbar in MXML mode

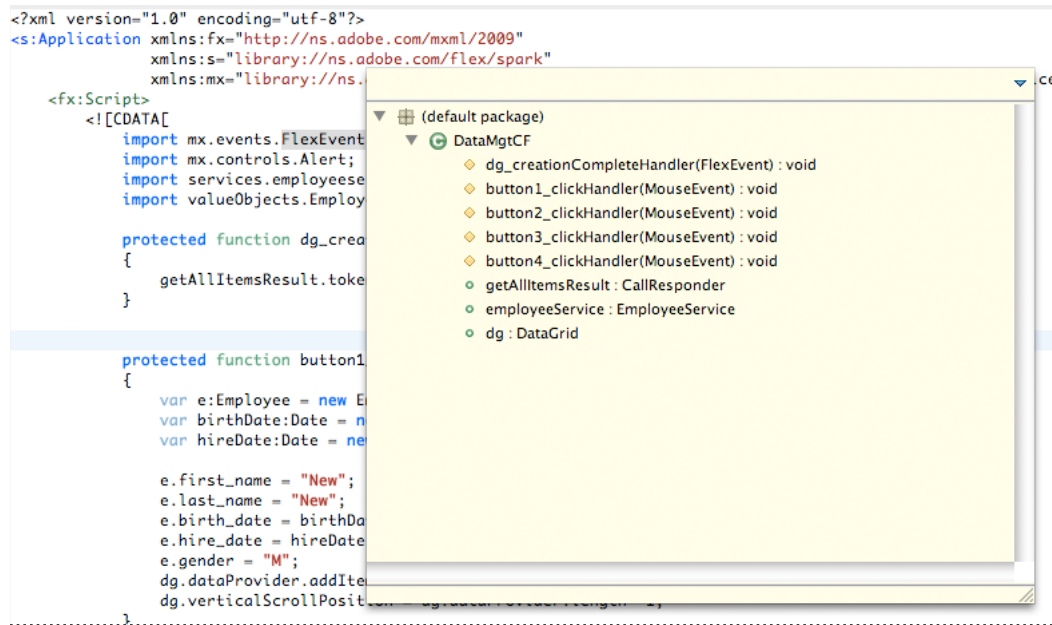
When the Outline view is in MXML mode, the toolbar menu contains additional commands to switch between the MXML and class views.



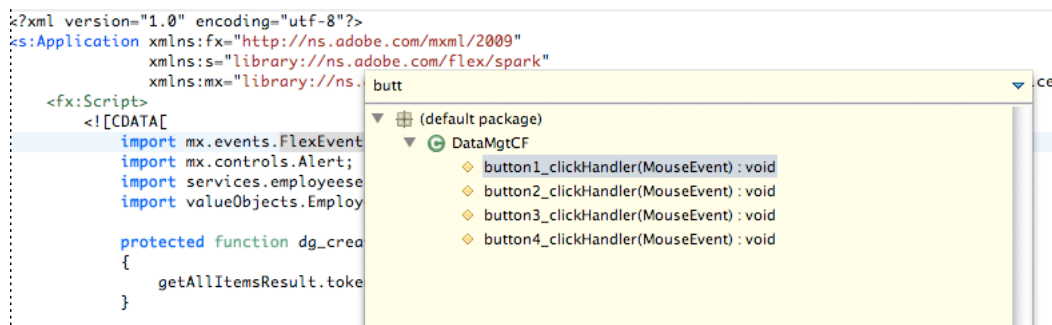
To switch between the two views, from the toolbar menu, select **Show MXML View** or **Show Class View**.

## Use Quick Outline view in the editor

Within the ActionScript and MXML editors, you can access the Quick Outline view, which displays the Outline view in Class mode. The Quick Outline view is displayed in a pop-up window within the editor itself, not as a separate view, and you can use it to quickly navigate and inspect your code.



The Quick Outline view contains the same content as the Class mode, but it also includes a text input area that you can use to filter the displayed items. For example, entering an item name into the Quick Outline view displays only the items that contain those characters.



The Quick Outline view does not contain the commands that let you alphabetically sort or hide items.

As in the Outline view, selecting an item locates and highlights it in the editor.

### Open the Quick Outline view

- ❖ With an ActionScript or MXML document open in the editor, from the Navigate menu, select Quick Outline.

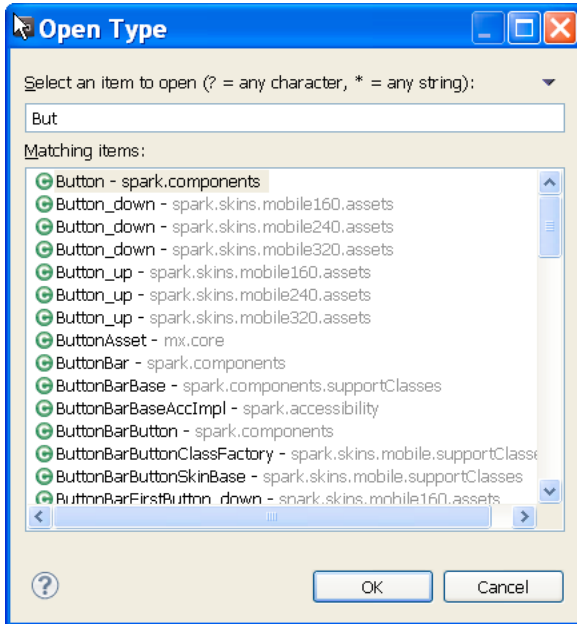
You can also use the keyboard shortcut, Control+O.

### Close the Quick Outline view

- ❖ Navigating outside the Quick Outline view closes the view. You can also press ESC to close the Quick Outline view.

## Browse and view classes

The Open Type dialog is available for browsing all available classes (including Flex framework classes) in your project. Select a class in the Open Type dialog to view the implementation.



Open Type dialog

The Open Type dialog is also available for selecting classes as the base class for a new ActionScript class or a new MXML component.

The Open Type dialog lets you filter the classes that are displayed according to the text and wild cards that you specify. The dialog uses color coding to indicate recommended types and excluded types. Recommended types display as gray. Excluded types appear brown.

*Recommended types* are those classes available in the default namespace for a project. For example, in some contexts only Spark components are allowed. In other contexts, both Spark and Halo components are allowed.

*Excluded types* are those classes that are not available in the default namespace for a project.

### Open the Open Type dialog

- (Browse classes) To browse classes and view their implementation:
  - 1 From the Flash Builder menu, select Navigate > Open Type.
  - 2 (Optional) Type text or select filters to modify the classes visible in the list.
  - 3 Select a class to view the source code.

You cannot modify the source code for classes in the Flex framework.
- (New ActionScript classes) When selecting a base class for a new ActionScript class:
  - 1 Select File > New > ActionScript class.
  - 2 For the Superclass field, click Browse.
  - 3 (Optional) Type text or select filters to modify the classes visible in the list.
  - 4 Select a base class from the list.

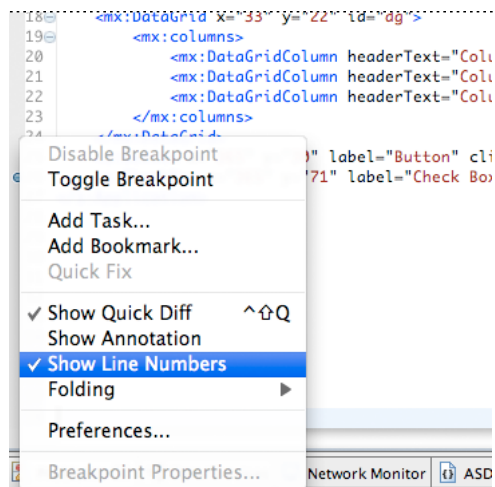
- (New MXML components) When selecting a base component for a new MXML component:
  - 1 Select File > New > MXML Component.
  - 2 From the list of projects in your workspace, select a project for new MXML component and specify a filename.  
The available base components vary, depending on the namespaces configured for a project.
  - 3 For the Based On field, click Browse.  
*Note:* Clear or modify the default base class listed in the Based On field to widen your choices.
  - 4 (Optional) Type text or select filters to modify the classes visible in the list.
  - 5 Select a base component from the list.

## Show line numbers

You can add line numbers in the editor to easily read and navigate your code.

- ❖ From the context menu in the editor margin, select Show Line Numbers.

The editor margin is between the marker bar and the editor.



## Use markers

Markers are shortcuts to lines of code in a document, to a document itself, or to a folder. Markers represent tasks, bookmarks, and problems and they are displayed and managed. Selecting markers opens the associated document in the editor and, optionally, highlights the specific line of code.

With Flash Builder, you must save a file to update problem markers. Only files that are referenced by your application are checked. The syntax in an isolated class that is not used anywhere in your code is not checked.

The workbench generates the following task and problem markers automatically. You can manually add tasks and bookmarks.

**Tasks** Task markers represent a work item. Work items are generated automatically by the workbench. You can add a task manually to a specific line of code in a document or to the document itself. For example, to remind yourself to define a component property, you might create a task called “Define skinning properties.” You can also add general tasks that do not apply directly to resources (for example, “Create a custom component for the employee log-in prompt”). You use the Task view to manage all the task markers. For more information, see “[Add tasks](#)” on page 78.

**Problems** Problem markers are generated by the compiler and indicate invalid states of various sorts. For example, syntax errors and warnings generated by the compiler are displayed as problem markers in the Problem view. For more information, see “[Problems view](#)” on page 9.

**Bookmarks** You can manually add bookmarks to a line of code or a resource (folder or document). You use bookmarks as a convenience, to keep track of, and easily navigate to items in your projects. You use the Bookmarks view to manage all bookmarks. For more information, see “[Add and delete bookmarks](#)” on page 79.

**Note:** The Tasks and Bookmarks views are not displayed by default in the Flash Development perspective. For more information about adding these views, see “[Work with views](#)” on page 17.

## Navigate markers

Markers are descriptions of and links to items in project resources. Markers are generated automatically by the compiler to indicate problems in your code, or added manually to help you keep track of tasks or snippets of code. Markers are displayed and managed in their associated views. You can easily locate markers in your project from the Bookmarks, Problems, and Tasks views, and navigate to the location where the marker was set.

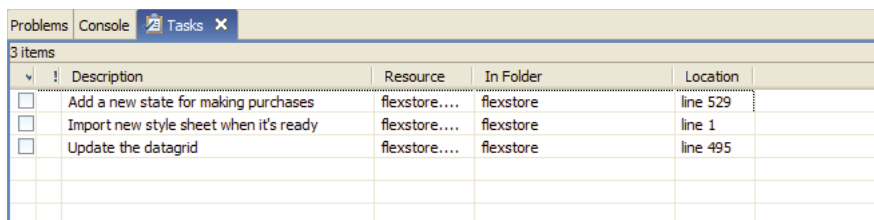
### Go to a marker location

- ❖ Select a marker in the Bookmarks, Problems, or Tasks views.

The file that contains the marker is located and opened in the editor. If a marker is set on a line of code, that line is highlighted.

## Add tasks

Tasks represent automatically or manually generated workspace items. All tasks are displayed and managed in the Tasks view (Window > Other Views > General > Tasks), as the following example shows:



	Description	Resource	In Folder	Location
<input type="checkbox"/>	Add a new state for making purchases	flexstore....	flexstore	line 529
<input type="checkbox"/>	Import new style sheet when it's ready	flexstore....	flexstore	line 1
<input type="checkbox"/>	Update the datagrid	flexstore....	flexstore	line 495

### Add a task to a line of code or a resource

- 1 Open a file in the editor, and then locate and select the line of code where you want to add a task; or in the Flex Package Explorer, select a resource.
- 2 In the Tasks view, click the Add Task button in the toolbar.
- 3 Enter the task name, and select a priority (High, Normal, Low), and click OK.

**Note:** The resource, as shown in the Flex Package Explorer, does not indicate that it was marked. You can view and manage all task markers in the Task view.

## Complete and delete tasks

When a task is complete, you can mark it and then optionally delete it from the Tasks view.

### Mark a task as complete

- ❖ In the Tasks view, select the task in the selection column, as the following example shows:

<input type="checkbox"/>	Update the data grid	CartView.mxml	flexstore	
<input type="checkbox"/>	<mx:Script> - Update script	flexstore.mxml	flexstore	line 32
<input checked="" type="checkbox"/>	task example	testapp.mxml	testapp	line 13

### Delete a task

- ❖ In the Tasks view, open the context menu for a task and select Delete.

### Delete all completed tasks

- ❖ In the Tasks view, open the context menu and select Delete Completed Tasks.

## Add and delete bookmarks

You can use bookmarks to track and easily navigate to items in your projects. All bookmarks are displayed and managed in the Bookmarks view (Window > Other Views > General > Bookmarks), as the following example shows:

Problems Console Tasks Bookmarks X				
5 items				
Description	Resource	In Folder	Location	
<mx:HTTPService	flexstore....	flexstore	line 370	
Compare function	flexstore....	flexstore	line 155	
Login event listener	myFlexAp...	myFlexApp	line 22	
Login panel	myFlexAp...	myFlexApp	line 21	
Script block	flexstore....	flexstore	line 13	

### Add a bookmark to a line of code or a resource

- 1 Open a file in the editor, and then locate and select the line of code to add a bookmark to.
- 2 From the main menu, select Edit > Add Bookmark.
- 3 Enter the bookmark name, and click OK.

A bookmark icon (  ) is added next to the line of code.

**Note:** The resource, as shown in the Flex Package Explorer, does not indicate that it was marked. You can view and manage all bookmarks in the Bookmarks view.

### Delete a bookmark

- 1 In the Bookmarks view, select the bookmark to delete.
- 2 Right-click (Windows) or Control-click (Mac OS) the bookmark and select Delete.

## Organize import statements

When you use Content Assist in the MXML and ActionScript editors, the packages in which classes are located are automatically imported into the document. They are added in the order in which they were entered into code. Imports that are unused or unneeded are automatically removed.

To help organize the code in your ActionScript documents, you can alphabetically sort import statements. To do so, open the Preferences dialog, select Flash Builder> Editors > ActionScript Code, and then select Keep Imports Organized.

### Sort import statements

- ❖ With an ActionScript document that contains import statements open in the editor, from the Source menu, select Organize Imports.

You can also use the keyboard shortcut: Control+Shift+O (Windows) or Command+Shift+O (Mac OS).



# Chapter 4: Using Projects in Flash Builder

Adobe® Flash® Builder™ lets you create, manage, package, and distribute projects for building web, desktop, and mobile applications. When you generate shared component library (SWC) files, you can share components and other resources between applications or with other developers. You can also work with different versions of the Adobe Flex SDK directly in Flash Builder.

## Create projects in Flash Builder

Flash Builder lets you create different types of projects such as Flex, Flex on mobile, ActionScript, and AIR projects, as well as other types of projects. Flash Builder provides you a project-creation wizard that guides you through the steps, prompting you for the type of project to create, the project name, location, application type (web or desktop), SDK versions, and other options.

For information about creating an ActionScript project, see “[ActionScript projects](#)” on page 84.

For information about creating library projects, see “[Use Flex library projects](#)” on page 212.

For information about creating Flash Professional projects, see “[Using Flash Builder with Flash Professional](#)” on page 228.

## Flex projects

You can use Flex projects to create Web applications (run in Flash Player) or Desktop applications (runs in Adobe AIR). There are options to create MX only Flex projects or Spark only projects that use only Spark components.

### Create Flex Projects

Use this procedure to create your basic Web or Desktop applications.

- 1 Select File > New > Flex Project.

- 2 Enter a project name and location.

The default location is the current workspace.

- 3 Select either Web or Desktop for the application type.

- 4 Use either the default Flex SDK, or browse to another installed SDK. Click Next.

- 5 (Optional) Specify Server Settings.

See “[Access data services](#)” on page 200.

- 6 Specify an Output Folder.

If you do not specify an application server, the location is inside your project folder.

If you specify an application server, the output server is outside the project folder. Typically, you place the output folder with your service files.

- 7 Click Finish, or click Next to specify additional configuration options.

- 8 (Optional) Specify Build Paths and other configuration options.

See “[Build paths, native extensions, and other project configuration options](#)” on page 87.

- 9 Click Finish.



Watch the video [Build your First Desktop Application with Flash Builder](#) by Adobe Evangelist [James Ward](#).

### More Help topics

[“Projects in Flash Builder”](#) on page 30

### Create a Flex project that uses only MX components

The MX Only option is useful for creating applications that are similar in design to applications created with the previous release of Flex (that is, Flex 3), but still have access to the latest Flex and Flash Builder features (such as the states syntax, advanced CSS, compiler improvements, and other language features).

If you specify MX Only for a project, then Spark components are not available to applications in the project.

You can convert a Flex project to be an MX Only project. However, Flash Builder does not rewrite any code in the project. Manually update your code to remove any reference to Spark components.

### Create an MX Only Flex project

- 1 Select File > New > Flex Project.
- 2 Specify Project Location and Server Settings, as described in [“Build paths, native extensions, and other project configuration options”](#) on page 87.
- 3 In the Build Paths page of the New Flex Project wizard, specify MX Only.
- 4 Specify other Build Path settings, as described in [“Build paths, native extensions, and other project configuration options”](#) on page 87. Click Finish.

### Convert a Flex project to an MX Only Flex project

- 1 Make the project the active project in Flash Builder:  
Typically, you open a source file in the project to make the project active.
- 2 Select Project > Properties > Flex Build Path.
- 3 For Component Set, select MX Only. Click OK.
- 4 Modify any application code in the project that accesses Spark components.  
You cannot reference Spark components in an MX Only project.

### Create a Flex project that uses only Spark components

The Spark Only option is useful for creating applications that use Flex and Flash Builder features such as the new states syntax, advanced CSS, compiler improvements, and other language features.

If you specify Spark Only for a project, then MX components that were available with Flex 3 are not available to applications in the project.

- 1 Select File > New > Flex Project.
- 2 Specify Project Location and Server Settings, as described in [“Flex projects”](#) on page 81.
- 3 In the Build Paths page of the New Flex Project wizard, specify Spark Only.
- 4 Specify other Build Path settings, as described in [“Build paths, native extensions, and other project configuration options”](#) on page 87. Click Finish.

## Flex mobile projects

With Flash Builder, you can create Flex projects targeted for mobile devices. A Flex mobile project creates an AIR application that can run on Apple iOS, BlackBerry Tablet OS, and Google Android devices.

### Create Flex mobile projects

Use this procedure to create a Flex mobile project for Apple iOS, BlackBerry Tablet OS, and Google Android devices.

1 Select File > New > Flex Mobile Project.

2 Enter a project name and location.

The default location is the current workspace.

3 Use the default Flex 4.6 SDK that supports mobile application development. Click Next.

4 Specify Mobile settings:

- Select the platform that your application must target. For more information, see [Choose target platforms](#).
- Specify an application template.

For more information, see [Choose an application template](#).

- Specify permissions.

Select the target platform, and set the permissions for each platform, as required. You can edit the permissions later in the application descriptor XML file.

For more information, see [Choose mobile application permissions](#).

- Specify platform settings.

Platform settings let you select a target device family. Depending on the platform that you select, you can select the target device or a target device family.

**Note:** There are no platform-specific settings for the Google Android or BlackBerry Tablet OS platform.

For more information, see [Choose platform settings](#).

- Specify application settings.

For more information, see [Choose application settings](#).

5 Click Finish or click Next to specify server settings.

6 (Optional) Specify Server Settings.

For more information, see [“Access data services”](#) on page 200.

7 Specify an Output Folder.

If you do not specify an application server, the location is inside your project folder.

If you specify an application server, the output server is outside the project folder. Typically, you place the output folder with your service files.

8 Click Finish, or click Next to specify additional configuration options.

9 (Optional) Specify Build Paths and other configuration options.

For more information, see [“Build paths, native extensions, and other project configuration options”](#) on page 87.

10 Click Finish.



Flex consultant, Brian Telintelo, blogged about [creating a mobile application for the Android platform](#).

For more information on mobile development using Flex and Flash Builder, see [Developing mobile applications with Flex and Flash Builder](#).

### More Help topics

“[Flex Mobile Projects](#)” on page 31

## ActionScript projects

Use ActionScript projects to create Web or Desktop applications that are based on either the Flash API (not the Flex framework) or AIR APIs.

### Create ActionScript projects

Use this procedure to create an ActionScript application for the Web or Desktop.

- 1 Select File > New > ActionScript Project.
- 2 Enter a project name and location.  
The default location is the current workspace.
- 3 Select either Web or Desktop for the application type.
- 4 Specify Build Paths.

For more information, see “[Build paths, native extensions, and other project configuration options](#)” on page 87.

- 5 Click Finish.

For more information on coding in ActionScript, see the [ActionScript Developer's Guide](#).

### Create ActionScript mobile projects

Use this procedure to create AIR-based ActionScript applications that can run on Apple iOS, BlackBerry Tablet OS, and Google Android devices.

- 1 Select File > New > ActionScript Mobile Project.
- 2 Enter a project name and location.  
The default location is the current workspace.
- 3 Use the default Flex 4.6 SDK that supports mobile application development. Click Next.
- 4 Specify Mobile settings:

- Select the platforms that your application must target.

For more information, see [Choose target platforms](#).

- Specify platform permissions.

Select the target platform, and set the permissions for each platform, as required. You can edit the permissions later in the application descriptor XML file.

For more information, see [Choose mobile application permissions](#).

- Specify platform settings.

For more information, see [Choose platform settings](#).

- Specify application settings.

Select Automatically Reorient if you want your application to rotate when the device rotates.

Select Full Screen if you want your application to display in fullscreen mode on the device.

5 Click Finish, or click Next to specify additional configuration options.

6 (Optional) Specify Build Paths.

See [“Build paths, native extensions, and other project configuration options”](#) on page 87.

7 Click Finish.



Read about [creating a mobile application in ActionScript for the Android platform](#) by, Adobe Community Professional, Randy Troppmann.

### More Help topics

[“ActionScript Mobile Projects”](#) on page 31

[“Manage launch configurations”](#) on page 111

## Flash Catalyst compatible projects

You can create a Flex project that is compatible with Flash Catalyst.

When you create a Flash Catalyst compatible project, the

`xmlns:fc="http://ns.adobe.com/flashcatalyst/2009"` namespace is added. This namespace lets you create components in Flash Builder that can be used in Flash Catalyst. Flash Catalyst CS5.5 supports a subset of components that are available in Flex 4.5 or a higher version.

The Flash Catalyst Compatibility checker is automatically turned on for a Flash Catalyst compatible project (Project > Flash Catalyst > Automatically Check Compatibility). If you use any incompatible assets or introduce any incompatibilities in the project, Flash Builder displays the compatibility errors in the Problems view. For more information on the guidelines to create a Flash Catalyst compatible project, see [Guidelines to create a Flash Catalyst compatible project](#).

A Flash Catalyst compatible project lets designers and developers collaborate and work on the same project. You can share a Flash compatible project between Flash Builder and Flash Catalyst through FXP or FXPL files. You can also edit a Flash Catalyst compatible project directly from within Flash Builder. Use this workflow if you have both Flash Builder and Flash Catalyst installed on the same computer.

For more information, see [“Using Flash Builder with Flash Catalyst”](#) on page 233.

### More Help topics

[Understanding Flash Catalyst and Flash Builder workflows](#)

## Create a Flash Catalyst compatible project

1 Select File > New > Flash Catalyst Compatible Project.

2 Enter a project name and location.

The default location is the current workspace.

3 Use either the default Flex SDK, or browse to another installed SDK. Click Next.

4 Click Finish.

## Check compatibility for Flash Catalyst compatible projects

You can do any of the following to ensure that your project remains compatible with Flash Catalyst.

- Select Project > Flash Catalyst. Enable Automatically Check Compatibility.
- Select Project > Properties > Flash Catalyst. Enable Automatically Run Flash Catalyst Compatibility Checker For This Project.
- Select Project > Flash Catalyst > Run Compatibility Checker.

## Edit a Flash Catalyst compatible project

- 1 With the project selected in the Package Explorer, select Project > Flash Catalyst > Edit Project In Flash Catalyst to launch Flash Catalyst.
- 2 While you edit the project in Flash Catalyst, the project is locked in Flash Builder to ensure that there are no conflicting changes.
- 3 Once you complete editing the project in Flash Catalyst, save the changes and exit Flash Catalyst.

**Note:** When editing a Flash Catalyst compatible project, if you introduce any incompatibilities, Flash Builder displays the compatibility errors in the Problems view. Right-click the problem to know more about the compatibility error. Make sure that you resolve all compatibility errors in Flash Builder before opening the project in Flash Catalyst.

- 4 To open the project for editing in Flash Builder, select Project > Flash Catalyst > Resume Working On Project.

When you select this option, you are prompted to save the changes made in Flash Catalyst. Doing so, brings the newly saved project back into Flash Builder. In the background, the originally exported project version from Flash Builder is deleted after the newly saved project version is successfully imported from Flash Catalyst.

- 5 At any point while editing the project in Flash Catalyst, you can discard the changes and return to Flash Builder. To do so, select Project > Flash Catalyst > Cancel Editing Project in Flash Catalyst.

When you select this option, any changes made to the project in Flash Catalyst are discarded and the original version of the project is opened. The original version of the project is the version that was initially opened in Flash Catalyst. Before deleting the project version open in Flash Catalyst, you have an option of saving the changes as a different project file in a new location.

## Create AIR projects

- 1 Open Flash Builder.
- 2 Select File > New > Flex Project.
- 3 Enter the project name.
- 4 In Flex, AIR applications are considered an application type. You have two application types: a Web application that runs in Adobe Flash Player and a desktop application that runs in Adobe AIR. Select Desktop Application as the application type.
- 5 Select the server technology (if any) that you want to use with your AIR application. If you're not using a server technology, select None and then click Next.
- 6 Select the folder in which you want to place your application. The default is the bin-debug folder. Click Next.
- 7 Modify the source and library paths as needed and then click Finish to create your AIR project.

Watch the video [Build your first desktop application with Flash Builder](#) by Adobe Evangelist [James Ward](#).

## Build paths, native extensions, and other project configuration options

When creating a Flex project, you can customize its configuration. All additional configuration steps are optional.

**Note:** You can also change a project's configuration after the project has been created. When the Flash Builder editor is in Source mode, go to *Project > Properties*.

**Source Path** Use the Source tab to add additional source folders to a project. You can reorder the source folders, edit the location of folders, and remove folders from the source path.

**Main source folder, main application file, output folder URL** By default, Flash Builder places source files in a project's src folder. The default name of the main MXML application file is the name of the project. You can change these defaults when creating the project.

When you create a project, Flash Builder runs application files from a default URL based on your project settings. Specify an output folder URL to override the default settings. The output folder URL is useful when you build and deploy your application on your project server, but you debug your application on a web server. You then specify the web server URL as the output folder URL. For example, if you specify <http://myserver/test.swf> as the output folder URL, a launch configuration is created with that URL.

See “[Set up a project output folder](#)” on page 98 and “[Run and debug applications](#)” on page 110.

**Library Path** Use the Library Path tab to specify the framework linkage and build path libraries

- Component Set

Typically, you make all components available. In some cases, you specify MX components only. See “[Component Set \(MX + Spark, Spark Only, or MX Only\)](#)” on page 97.

- Framework Linkage

By default, application classes for Flex 4.5 or a higher version of the Flex framework use dynamic linking. The following options are also enabled by default:

- Verify RSL Digests (Recommended For Production)
- Remove unused RSLs

**Note:** This option is not available for Flex frameworks older than Flex 4.5.

- Use Local Debug SWF RSL When Debugging
- Automatically Determine Library Order Based on Dependencies

For more information, see “[Application Framework Linkage](#)” on page 97.

- Build Path Libraries

You can add or remove project libraries, SWC library folders, or SWC files to the build path. You can also change the build path order.

Use the Edit button to change the location of added libraries or folders.

Use the Add Flex SDK button to restore the default SDK for a project if you removed the Flex SDK from the build path.

**Native Extensions** Use the Native Extensions tab to include ActionScript Native Extension (ANE) files to provide native platform capabilities in your applications.

**Important:** You can create ActionScript extensions only for mobile and desktop projects that support AIR 3.0.

For more information, see [Add native extensions to a project](#).

## Export and import projects

Flash Builder lets you export and import projects in FXP format, ZIP format, and from open file directories. The FXP format is an archive format that includes project folders, files, and metadata about the project. Flash Builder exports Flex projects in the FXP format and Flex library projects in the FXPL format. ActionScript projects can only be exported as an archive file, typically in the ZIP format.

Importing and exporting projects allow you to transfer projects securely and safely between different computers and users.

**Note:** You can also use the Eclipse Export wizard to export Flex projects and Flex library projects in ZIP format (used in Flex Builder 3) or other archive formats.

### Export projects

Flash Builder can export various projects in addition to Eclipse export functions.

#### Export a Flex project or Flex library project as an FXP file

Some Flex projects require special handling upon import. See [“Projects requiring special handling”](#) on page 92

- 1 In Flash Builder, select File > Export Flash Builder Project.

You can also use the context menu for a project in the Package Explorer. Select Export > Flash Builder > Flash Builder Project.

- 2 In the Export Flex Project wizard, select the project to export.

All available projects for export are listed in the Project pop-up menu.

- 3 Browse to the location on your file system to which you want to export the FXP file.

- 4 (Optional) Select Validate Project Compilation.

Use this option to confirm that your project compiles without errors. If there are errors, you can still export the project.

- 5 Click Finish.

For server projects, absolute paths to server resources are saved as path variables. When you later import the project, you specify values for the path variables.

#### Export an ActionScript project in ZIP format (or other archive format)

- 1 In Flash Builder, select File > Export > Other.

- 2 In the Export wizard, select General > Archive File, and click Next.

- 3 Select the project and files to export:

- In the leftmost panel, expand the project to specify project folders to include
- In the rightmost panel, for each selected folder, specify the files to include.

- 4 Browse to a location to save the exported project and specify a filename.

- 5 Specify archive file options and click Finish.



## Import projects

Flash Builder can import Flex projects, Flex library projects, and ActionScript projects that were previously exported from Flash Builder. You can import multiple versions of the same Flex project or Flex library project.

### Import a Flex project or a Flex library project

You can import a project from an exported FXP file or by navigating to a folder containing the project.

- 1 From the Flash Builder menu, select File > Import.

You can also use the context menu for the Package Explorer to import a project.

- 2 (Project folder) If you are importing from an existing project folder, select Project Folder, and navigate to the folder containing the project.

- 3 (FXP file) If you are importing from an FXP file, select File and navigate to the location of the file.

If an FXP file contains more than one project, you can select individual projects to import.

- 4 (Library project or FXPL project) If you are importing a library project or a Catalyst FXPL project, you have the option to import the contents into an existing project.

- 5 (FXP file) If a project by the same name exists in the workspace, specify the import method:

- Import as New Project: Flash Builder appends a numeric identifier to the project name. Previous versions of the project are preserved.

In the Extract To field, specify a location in which to extract the file. Typically, this location is a directory in your Flash Builder workspace representing a project folder. You can specify a new project folder or overwrite an existing project folder.

- Overwrite existing project: Select the project to overwrite. The previous version of the project is permanently removed.

- 6 (Path variables) If you are importing a project that defines path variables, update path variables for the project.

Projects compiled for ColdFusion, PHP, ADEP Data Services, or other server technologies use path variables to access a web server and server resources. Other projects can have user-defined path variables.

Select each path variable and provide a valid value for the variable.

- 7 (Font references) If you are importing an FXP exported by Catalyst, the project can contain font references. You have the option to resolve references to fonts.

See [“Resolve font references when importing Catalyst projects”](#) on page 91.

- 8 Click Finish.

- 9 (PHP server projects) If you are importing a project of application server type PHP, then install or update your Zend installation.

The Zend dialog guides you through the process.

**Note:** If you cancel the process in the Zend dialog, manually install or update your Zend framework. You cannot access PHP services unless the Zend Framework is properly installed and configured. See *Installing Zend Framework for information on installing, configuring, and troubleshooting your Zend Framework installation.*

- 10 (Server projects) Deploy services.

- a Manually place services under the web root of the server. Use the same directory structure that was used in the original project.

- b In the Data/Services view, from the context menu for a service, select Refresh.

## Import a Flex 3 project

You can import a Flex 3 project into Flash Builder 4.6 using Flex 3 Compatibility Mode. In this case, the namespaces and components are unchanged from Flex 3. However, you can take advantage of the compiler available with Flex 4.6.

New documents created in Flex 3 Compatibility Mode use MX components and the following namespace:

```
mx="http://www.adobe.com/2006/mxml"
```

- 1 In Flash Builder, select File > Import Flex Project.
- 2 Navigate to the previously exported Flex 3 project ZIP file, or browse to the Flex 3 Project folder.
- 3 Click Finish.
- 4 In the Choose Flex SDK Version dialog, make sure that Flex 4 SDK is specified. Select Use Flex 3 Compatibility Mode.
- 5 Select OK.

For more information about using Flash Builder 4 for your existing Flex 3 projects, see the Adobe DevNet article:

[Moving existing Flex projects from Flex Builder 3 to Flash Builder 4](#)

## Import an ActionScript project

Use the Eclipse wizard to import an ActionScript project.

- 1 In Flash Builder, select File > Import > Other > General > Archive File.  
You can also use the context menu for the Package Explorer to import an ActionScript project.
- 2 In the Import Flex Project dialog box, select the ZIP file you want to import.
- 3 Click Finish.

## Import projects exported with the Eclipse Export wizard

If you have a project that was exported using Eclipse's Export wizard, use the Eclipse Import wizard to import the project. Select File > Import > General. Then navigate to the appropriate format for your project.

For more information, see the Eclipse documentation for importing projects. This documentation is available as Help in the Eclipse Import and Export wizards.

If the project contained services created with Flash Builder tools for accessing data services, then you manually have to add the services. Copy the server files in the services folder to an appropriate server. Use the service properties for a server from the Data/Service view to determine the service location.

If you exported a PHP project that uses the Zend Framework, install the Zend Framework on the target server. Modify the `amf-config.ini` file that configures the Zend Framework. For `zend_path`, specify the absolute path to the Zend installation directory.

For information on installing, configuring, and troubleshooting your Zend Framework installation, see [Installing Zend Framework](#).

## Import projects into multiple workspaces

When you import a project, you import it into a Flash Builder workspace. A project can be imported into multiple workspaces. In this scenario, the project files exist on disk in one location, but are referenced by each workspace. Changes you make to a project apply to all workspaces.

## Import source files into a new project

If you have source files and assets on your file system, but are not in a project, you can create a project for these files.

- 1 From the Flash Builder menu, select File > New > *Project*.

Project can be a Flex project, Flex library project, or ActionScript project.

- 2 In the New Project wizard, specify the source and output folder settings to the appropriate location in your file system.

**Note:** You could also accept the default wizard locations and move the source files accordingly.

## Compare changes in a project

If you import multiple versions of a project you can compare, copy, or merge the contents of the versions. You can only compare different versions of the same project.

- 1 In the Package Explorer, select one of the projects you want to compare.

- 2 Open the Package Explorer context menu and select Compare Project With Version.

The Compare Viewer launches, allowing you to compare the project with other versions of the project.

- 3 Select the version for comparison, which opens the Eclipse Compare Editor.

- 4 In the compare editor, navigate to the file you want to compare and from the context menu, select Show Content Comparison.

The Compare Editor displays both versions of the file, with differences highlighted.

You can use Compare Editor options to copy or merge differences in the file. See the Eclipse documentation on the Compare Editor for details.

## Support for Flash Catalyst projects

Flash Builder provides development support to application designers using Adobe® Flash® Catalyst™.

Flash Catalyst exports a project as an FXP file and exports components in an FXPL file. The FXP and FXPL files can then be imported into Flash Builder for development. For FXP files, the resulting project is a Flex web project that runs in Adobe Flash Player. An FXPL file contains a library file. You can import an FXPL file as a Flex library project or you can import the contents into an existing Flex project.

- To import an FXP or FXPL file into Flash Builder, from the Project menu, select Flash Catalyst > Import Flash Catalyst Project.
- To export a Flex project as a Flash Catalyst project, from the Project menu, select Flash Catalyst > Export Flash Catalyst Project. At the time of exporting a Flex project, you can validate the compatibility of the Flex project with Flash Catalyst.

You can create an Adobe AIR project from a Flash Catalyst project. Import the FXP file for the Catalyst project into Flash Builder. Convert the application type for the project from Web (runs in Adobe Flash Player) to Desktop (runs in Adobe AIR). See [“Change a web application project to a desktop application”](#) on page 119.

For more information about the workflows between Flash Builder and Flash Catalyst, see [“Using Flash Builder with Flash Catalyst”](#) on page 233.

## Resolve font references when importing Catalyst projects

When importing an FXP project created with Adobe Catalyst, the imported project can contain references to fonts that are not available on your system.

The Import wizard provides the option to fix font references using CSS. If you select this option, Flash Builder imports the Catalyst style sheet `Main.css`. `Main.css` contains references to the fonts used in the project.

If you get compile errors from the fonts referenced in the style sheet, fix the references in the style sheet with fonts available on your system.

Catalyst FXPL projects do not contain style sheets. Flash Builder attempts to correct any references to fonts when importing an FXPL file. If Flash Builder cannot find a corresponding font on the target system, the original font references are left intact. For FXPL projects, font references that Flash Builder cannot resolve are discovered at runtime. There is either a font substitution or a runtime error for unresolved font references.

**Note:** For FXPL files, Flash Builder modifies the `fontFamily` attribute in MXML files when it attempts to resolve font references.

## Projects requiring special handling

Some Flex projects require special handling upon import and export. For example:

- The project references an earlier version of the Flex SDK
- The project references service files for access to server-side data
- The Zend Framework configuration for access to PHP services needs to be updated
- The project uses ADEP Data Services links to a data model file.

When exporting or importing a Flex project, some contents of the project require special handling.

- Different versions of the Flex SDK

You can import a Flex project that references a version of the Flex SDK that is not installed with your Flash Builder. See [“Installed Flex SDKs”](#) on page 250 for information on downloading and installing additional versions of the Flex SDK. If Flash Builder cannot find a specific version of the Flex SDK, browse to the location of the SDK.

- Service files

Flex server projects that connect to data services, such as ColdFusion or BlazeDS, contain a services folder whose ActionScript class reference deployed server files. When exporting the project, Flash Builder exports the services folder but you have to ensure that a server and the corresponding server-side files exist when importing. Upon import, you might have to manually deploy the server-side files and also update the server addresses in classes in the server-side files. For projects that connect to services using ADEP Data Services or BlazeDS, make sure that service destinations are available on the target server.

- Zend Framework

Flex projects that connect to data services using PHP and the Zend Framework contain two configuration files. Upon import, examine these files to make sure that they are configured properly for your system:

`amf-config.ini`

`gateway.php`

See [Installing Zend Framework](#)

for information on installing, configuring, and troubleshooting your Zend Framework installation.

- Data model files (ADEP Data Services)

A Flex project that uses ADEP Data Services links to a data model file. Upon export and subsequent import, Flash Builder references the actual data model file and not a link to it. If you want to use a linked file, and not the one packaged with the exported project, then change the data model file using project properties. Select Project > Properties > Data Model and make the changes.

## Build projects

Adobe® Flash® Builder™ automatically builds and exports your projects into applications, creating application and library files, placing the output files in the proper location, and alerting you to any errors encountered during compilation.

There are several options for modifying the build settings to control how your projects are built in to applications. For example, you can set build preferences on individual projects or on all the projects in your workspace. You can also modify the build output path, change the build order, and so on. You can also create custom build instructions using third-party build tools such as the Apache Ant utility.

When your applications are ready to be released, you have the option of publishing all or selected parts of the application source code. Users can view your application source code in a web browser, similar to the way they are able to view HTML source code.

## Understand how projects are built and exported

A typical workflow consists of building your Flex and ActionScript projects with the Build Automatically option enabled. During the development process, Flash Builder gives you errors and warnings in the Problems view. When you run your application, a debug version of the SWF file is placed in the project output (bin) folder along with required assets and an HTML wrapper. This build contains debug information and is suitable for developer use only. For more information about exporting projects, see “[Export and import projects](#)” on page 88.

When your application is ready to deploy, you create an optimized, release-quality version of your application using the Export Release Build wizard. This stores the SWF file in the bin-release folder. Since debug information is removed, the file size is smaller. This version is a production build that end users can view. For Adobe AIR projects, AIR applications are exported to an AIR file. You use Export Release Build to create a digitally signed AIR file, which users must install before running an application (similar to an install.exe).

For library projects, you do not have to export. The SWC file built by a Flex library project is suitable for both developer and production use. For more information see “[Use Flex library projects](#)” on page 212.

## Build basics

MXML and ActionScript 3.0 are *compiled* languages. Compiled languages are different from *interpreted* languages, such as JavaScript, that can be executed by their runtime environments. That is, MXML and ActionScript 3.0 must be converted into a compiled format before they can be executed by Flash Player. This process, along with the generation of related output files, is called *building*.

Flash Builder automatically builds your projects whenever a file in your project is changed and saved. You also have the option of building your applications manually. Understanding the build process and the output files that are generated helps you to diagnose and repair project configuration problems, if any.

**Flex projects** Source files and embedded assets (such as images) are compiled into a single output SWF file. The SWF file can be run directly in the stand-alone Flash Player or in a web browser through an *HTML wrapper* file that is also generated by the build. These files are generated into the project's output folder. The output folder is named *bin*, by default, but you can rename it.

**ActionScript 3.0 projects** Like Flex projects, ActionScript 3.0 projects compile source files and embedded assets into a SWF file.

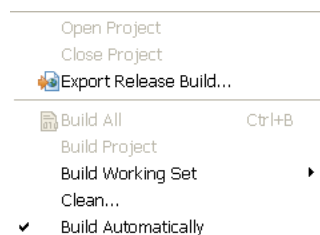
**Flex library projects** For library projects, source files are components and related resources. When library projects are built, a SWC file is generated into the output folder. A SWF file is archived into a SWC file containing components, resources, and a *catalog.xml* file that is the manifest of the elements contained within the SWF file.

## Automatic builds

In the stand-alone configuration of Flash Builder, the option to Build Automatically is selected, by default, and your applications are built automatically. In the plug-in configuration, select the Build Automatically option. Deselecting the Build Automatically option prevents the compiler from identifying syntax errors. The Problems view then does not display warning and error messages as you enter code. Only when you compile the project, the Problems view displays any warning and error messages. It is therefore recommended to set Flash Builder to build automatically.

## Advanced project build options

With the advanced build options, you can control the timing and scope of your builds. For example, you can build a single project, all projects in the workspace, or create a working set (a collection) of projects to build. All build commands are accessible from the Project menu, as shown in the following example.



The Flash Builder compiler is incremental. It builds only those resources that have been added or affected by updates and ignores all others. This saves time and system resources. You have the option, however, to rebuild all the resources in the project. You do this by performing a *clean build*. You might do this if your application is behaving erratically during testing and you want to eliminate all potential sources of the problem by discarding and rebuilding all the files in your project. For more information, see [“Advanced build options”](#) on page 100.

If you create dependencies between separate projects in the workspace, the compiler automatically determines the order in which the projects are built, so these dependencies resolve properly. You can, however, override the default build order and manually set the order in which the projects in your workspace are built.

You can also modify the build path, application list, and compiler settings for each project in the workspace.

## More Help topics

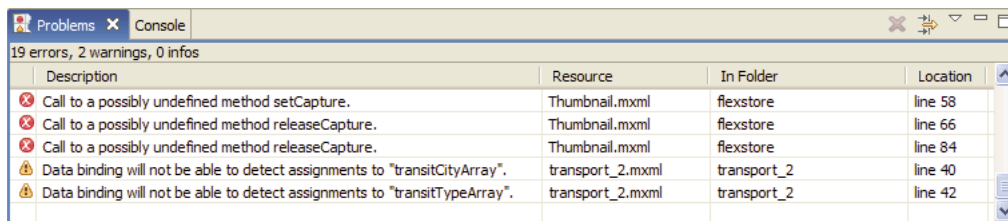
[“Build projects manually”](#) on page 100

[“Manage project application files”](#) on page 35

[“Advanced build options”](#) on page 100

## Build errors displayed in the Problems view

Errors encountered by the compiler during builds appear in the Problems view, which is included in the Development and Debugging perspectives, and in the code editor, where lines of code containing errors are marked with an x, as in the following example:



Description	Resource	In Folder	Location
Call to a possibly undefined method setCapture.	Thumbnail.mxml	flexstore	line 58
Call to a possibly undefined method releaseCapture.	Thumbnail.mxml	flexstore	line 66
Call to a possibly undefined method releaseCapture.	Thumbnail.mxml	flexstore	line 84
Data binding will not be able to detect assignments to "transitCityArray".	transport_2.mxml	transport_2	line 40
Data binding will not be able to detect assignments to "transitTypeArray".	transport_2.mxml	transport_2	line 42

## More Help topics

[“Problems view”](#) on page 9

## Eclipse environment errors in the log file

Sometimes you encounter errors thrown by the Eclipse environment. These errors most often occur when resources such as SWC files are not found at runtime. In these cases, you can see the error messages in the Eclipse Error Log file. The default location of this log file on Windows is c:\Documents and Settings\user\_name\workspace\metadata\log. For Macintosh, the default location is also in the workspace directory, but files and directories that begin with a dot are hidden by default.

## Custom build scripts with Apache Ant

You can modify and extend the standard build process by using Apache Ant, which is an open-source Java-based build tool. For more information about creating custom builders, see [“Customize builds with Apache Ant”](#) on page 101.

## Flex compiler options

You can modify the default Flex compiler settings that Flash builder uses. To view the default settings and modify the defaults, open the Flex Compiler properties page. From the Flash Builder menu, select Project > Properties > Flex Compiler.

## Flex SDK version

The default SDK for Flash Builder is Flex 4.6. However, if your project uses a specific version, such as Flex 3.5, Flash Builder compiles applications in the project using the specified Flex SDK.

You can change the default setting to use a specific Flex SDK, or to compile using Flex 3 compatibility. Specifying backward compatibility affects some behavior such as the layout rules, padding and gaps, skins, and other style settings. In addition, it affects the rules for parsing properties files. Setting the compatibility version does not enforce all differences that exist between the versions.

## Adobe Flash Player options

The default version of Flash Player used by the compiler is the minimum version required by the Flex SDK used for compilation.

You can specify a specific version of Flash Player that you want to target for the application. Features requiring a later version of Flash Player are not compiled into the application.

## Compiler options

Flash Builder provides check boxes for the following compiler options:

- Use Flash Text Engine in MX components

The Flash Text Engine (FTE) is a library that provides text controls with a rich set of formatting options. All Spark components in the `spark.components` package support FTE. See [Embed fonts](#).

Some MX controls provide support for FTE. MX controls that support FTE use the same embedded fonts as Spark components using FTE. See [Using FTE in MX controls](#).

- Copy non-embedded files to output folder
- Generate accessible SWF file

Enables accessibility features when compiling the application or SWC file. For information on using accessibility features with Flex, see [Accessible applications](#).

- Enable strict type checking

When strict type checking is enabled, the compiler prints undefined property and function calls. The compiler also performs compile-time type checking on assignments and options supplied to method calls.

- Enable warnings

This option enables specified warnings. For more information, see [Viewing warnings and errors](#).

You can also specify multi-line compiler arguments that are available with the command-line, `mxmcl` compiler. You can set the values of most options in the Additional Compiler Arguments field by using the same syntax as on the command line. For information about the syntax for setting options in the Flex Compiler dialog box, see [About the command-line compilers](#).

In the Additional Compiler Arguments field, you can substitute a path to the SDK directory by using the `${flexlib}` token, as the following example shows:

```
-include-libraries "${flexlib}/libs/automation.swc" "${flexlib}/libs/automation_agent.swc"
```

## HTML wrapper

In addition to generating SWF files for web applications, the Flash Builder compiler also generates an HTML wrapper that you can use when you deploy the application. The following options are available:

- Generate HTML wrapper file
- Check target player version

When enabled, the compiled application checks for the correct version of Flash Player.

If Express Install is enabled, the application runs a SWF file in the existing Flash Player to upgrade users to the latest version of the player.

- Enable integration with browser navigation



This option enables deep linking. Deep linking lets users navigate their interactions with the application by using the Back and Forward buttons in their browser.

## Command-line access to the Flex framework compilers

You have direct command-line access to use the Flex framework compilers (mxmmlc and compc). For more information, see About the command-line compilers in [Using Adobe Flex 4.6](#).

## Customize project builds

Flash Builder allows you to build your applications automatically using the default project settings. Using the default project settings is the recommended approach to building your applications. You can, however, customize project builds to suit your needs. For example, you can change the default output folder or modify the compiler options.

### Component Set (MX + Spark, Spark Only, or MX Only)

By default Flex projects have the entire component set available to applications in the project. The component set includes Spark components introduced with Flex 4 as well as MX components that were available with Flex 3.

In some scenarios, you may want to use only the MX components that were available with Flex 3, or only the Spark components introduced with Flex 4. For example, suppose you have an existing Flex 3 project and you do not want to introduce the new Spark components. But you do want to take advantage of features introduced with Flex 4 and Flash Builder 4, such as the new states syntax, compiler improvements, and other language features. In this scenario, you select the MX Only component set. When you select the MX Only component set, all Spark related libraries are removed from the build path. If you convert a Flex 4 project to MX Only, Flash Builder does not modify any of the code in the project. You must manually update your code to remove any references to Spark components and libraries.

Similarly, if you select the Spark Only component set, only the Spark components are used, and any MX-related libraries are removed from the build path.

### Application Framework Linkage

By default, application classes for Flex 4 and higher versions of the Flex framework use dynamic linking. Rather than compiling all classes into the application SWF file (static linking), some classes are loaded from the framework runtime shared library (RSL). Applications built with dynamic linking have smaller SWFs, which means they download faster. However, these applications use more memory because all framework classes are loaded, not just the classes you need. For more information, see Runtime Shared Libraries.

You can modify a project's properties to customize this behavior for all applications in a project. After selecting a project, from the Flash Builder menu, select Project > Properties > Flex Build Path > Library Path.

By default, Flash Builder uses the default behavior of the Flex SDK for framework linkage. For Flex 4 and higher versions, the default behavior is dynamic linking of RSLs. For Flex 3, the default behavior is static linking. Use the Framework Linkage drop-down list to override the default behavior.

For Flex 4.5 SDK and higher versions of the Flex framework, the following options are enabled by default:

- Verify RSL Digests (Recommended For Production)

Verifies that the digest of the RSL matches the digest that was stored in the application at compile time when the application was linked to the cross-domain RSL. For more information, see About RSL digests.

- Remove Unused RSLs

Removes RSLs that are not used during compile time. Only the RSLs that are used are copied to the output folder. This option is available only for Flex 4.5 SDK and higher versions of the Flex framework, not older Flex frameworks.

You can, however, force-load an unused RSL during compile time by double-clicking the project library path. Then, select Force Load RSL in the Library Path Item Options dialog box.

You can specify the application domain in which the cross-domain RSLs have to be loaded. You can select Default, Current, Parent, or Top-Level as the application domain.

For example, if a module uses a specific RSL, you can select Current as the application domain. Then, the specified RSL is accessible only to that module and not to the application loading the module. Unloading the module automatically unloads the associated RSL also.

**Note:** *Cross-domain RSLs and application domains are supported only in Flex 4.5 SDK and higher versions of the Flex framework, not older Flex frameworks. If you are importing a project that was built using a Flex framework that used standard RSLs, then Flash Builder automatically converts the standard RSLs into cross-domain RSLs.*

For more information, see Compile with standard or cross-domain RSLs in Flash Builder.

- Use Local Debug SWF RSL When Debugging

Use local RSLs when debugging the application. Using local RSLs allows you to step into debug RSL files. This option is ignored when exporting a release build.

- Automatically Determine Library Order Based On Dependencies

If enabled, Flash Builder determines the library order, based on dependencies in the libraries. To customize the library order, disable this option and use the up and down buttons to specify a library order.

## Enable and disable automatic builds

In the stand-alone configuration of Flash Builder, your projects are built automatically. In the plug-in configuration, select this option yourself. Flash Builder is designed to automatically build your projects; turning this option off prevents the compiler from identifying syntax errors and displaying warning and error messages as you enter code. For more information about building your projects manually, see [“Build projects manually”](#) on page 100.

Do one of the following:

- Select Project > Build Automatically.
- Open the Preferences dialog and select the General > Workspace. Select or deselect the Build Automatically option.

The Build Automatically option affects all projects in the workspace.

## Set up a project output folder

When you create a project in Flash Builder, by default, the build output is generated into the output folder.

You can change the name of this folder when you create the project or after the project is created. You can either create a folder or select an existing folder in the workspace.

- 1 In the Flex Package Explorer, select a project.
- 2 Right-click (Control-click on Macintosh) and select Properties from the context menu.

The Project Properties dialog box appears.

- 3 Select the Flex Build Path properties page.
- 4 Change the existing output folder by entering a new name or by navigating to an existing folder in your project and selecting it.

**Note:** *You cannot change the output folder of an ADEP Data Services ES application in this manner because its location is controlled by the server and is accessible only through the project's Flex-config.xml file.*

- 5 Click OK.

The new output folder replaces the existing output folder.

**Important:** When you change the name of the output folder, the original output folder and all of its contents are deleted. Rebuild the project to regenerate the application SWF file and HTML wrapper files.

## Modify a project build path

Each project has its own build path, which is a combination of the source path and the library path. (Library project build paths are a little more complex. For more information, see [“Flex Library Projects”](#) on page 32.) The source path is the location of the project MXML and ActionScript source files. The library path is the location of the base Flex framework classes and any custom Flex components that you have created, in the form of SWC files.

### Modify the source path

- 1 Select a project in the Flex Package Explorer.
- 2 Right-click (Control-click on Macintosh) and select Properties from the context menu. The Project Properties dialog box appears.
- 3 Select the Flex Build Path properties page. (If you’re working with an ActionScript project, select the ActionScript Build Path properties page.)
- 4 Add a folder to the source path by clicking the Add Folder button.
- 5 Enter a name for the folder or click the Browse button to select the location of the custom classes.

You can also use path variables rather than entering the full path to the file system. You can either enter the name of an existing path variable or create a path variable; for more information, see [“Create a path variable”](#) on page 99.

- 6 Modify the source path as needed, and click OK.

### Modify the library path

- 1 Follow steps 1 through 3 of the previous procedure to access the Flex Build Path properties page.
- 2 Click the Library Path tab.

The library path contains references to the Flex framework classes, which are contained in SWC files. A SWC file is an archive file for Flex components and other assets. For more information, see [“Use SWC files in your projects”](#) on page 215.

You can edit the path to the framework or, if you created custom Flex components, add new folders or SWC files to the library path. You can also remove items from the path.

- 3 Modify the library path as needed, and click OK.

## Create a path variable

Rather than linking to resources by entering the full path to the local or network folder where you store your files, you can define path variables. For example, you can define a path variable called Classes and then set the path to a folder on the file system. You then select Classes as the location of the new linked folder. If the folder location changes, you can update the defined path variable with the new location and all the projects that are linked to Classes continue to access the resources.

### Set or create a path variable

- 1 Select a project in the Flex Package Explorer.

- 2 Right-click (Control-click on Macintosh) and select Properties from the context menu. The Project Properties dialog box appears.
- 3 Select the Flex Build Path properties page. (If you're working with an ActionScript project, select the ActionScript Build Path properties page.)
- 4 You can create a path variable for any item on the path (this includes folders in the source path and SWC folders, projects, and SWC files in the library path). As an example, on the Source Path tab select the Add Folder button. The Add Folder dialog box appears.
- 5 Enter a path variable using the following format: `${pathvariablename}`.

**Note:** If the variable does not exist, the path entry fails. The list of existing resource variables is available by selecting *Window > Preferences from the main menu and then selecting General > Workspace > Linked Resources*. You can also manage linked resource variables on this properties page.

- 6 Click OK to add the path variable to the path.

## Advanced build options

Flash Builder has advanced options for customizing project builds. You can, for example, build projects manually, change the default build order of projects in the workspace, and create custom builders using the Apache Ant utility.

### Build projects manually

When you build projects manually, you can control the timing and scope of the build. For example, you can build a single project, all projects in the workspace, or create a working set of projects or selected project resources and build only those projects and resources. A working set is a collection of workspace resources (projects, files, and folders) that you can select and group together and work with as you see fit. For more information about working sets, see [“Create working sets”](#) on page 36.

#### Build a single project

- 1 In the Flex Package Explorer, select the project you want to build.
- 2 Select Project > Build Project from the main menu.

The selected project is built, and new or updated release and debug application files are added to the project output folder.

**Note:** If any of your project files need to be saved, you are prompted to do so before the build begins. To bypass this save prompt, you can set workspace preferences to save files automatically before a build begins.

#### Build all projects in the workspace

- ❖ Select Project > Build All from the main menu.

All projects in the workspace are built and application files are added to each project output folder. You are then prompted to save files if you have not already chosen to save files automatically before a build begins.

#### Build a working set

Do either of the following:

- Select Project > Build Working Set > Select Working Set from the main menu. Click New to create a working set. For more information about creating a working set, see [“Create working sets”](#) on page 36.
- Choose an existing working set by selecting Project > Build Working Set > Select Working Set from the main menu.

All projects in the working set are built and the application files are added to the project output folder.

## Save project resources automatically

When you build your projects manually, you are prompted to save all resources before the build begins. To bypass this prompt, you can set workspace preferences to automatically save project resources.

- 1 Open the Preferences dialog, select General > Workspace.
- 2 Select the Save Automatically Before Build option.
- 3 (Optional) You can modify how often resources are saved by entering a value (in minutes) for the Workspace Save Interval.

## Perform a clean build

After a project has been built, subsequent builds affect only the resources that have been added or modified. To force the Flash Builder compiler to rebuild all resources in a project, you can perform a clean build. You can perform a clean build if, for example, you want to eliminate all potential sources of a problem you encountered when testing your application.

- 1 Select Project > Clean from the main menu.
- 2 Select the project (or projects) whose build files you want to discard and rebuild from scratch.
- 3 Click OK.

## Change the project build order

Flash Builder lets you create relationships between projects when working with multiple projects in the workspace. For example, you can import ActionScript classes from one project into another. Creating relationships between projects affects the order in which your projects are built.

By default, the compiler builds related projects in the order required to build them all properly. For example, if a project refers to classes contained in another project, the project containing the classes is built first. In most cases, relying on the compiler to build projects in the proper order is sufficient and your applications is generated successfully.

You can, however, change the build order. For example, you can change the build order if you created a custom Ant builder and associated it with a project in your workspace, and you have to build that project before other projects are built. For more information about creating custom builders, see [“Customize builds with Apache Ant”](#) on page 101.

- 1 Open the Preferences dialog and select General > Workspace > Build Order.

The Build Order dialog box displays the following options:

**Use Default Build Order** The default build order is dictated by the dependencies between projects and is handled by the compiler.

**Project Build Order** You can manually set the build order for all the projects in the workspace. You can also remove a project from the build order list; it will still be built, but only after all the projects in the build order list.

**Max Iterations When Building With Cycles** If your projects contain cyclic references (something you should avoid), you can set the number of build attempts so that the compiler can properly build all the projects. The default maximum number of iterations is 10.

- 2 Modify the build order as needed, and click OK.

## Customize builds with Apache Ant

By creating a custom builder, you can modify and extend the standard build process. Flash Builder contains a standard build script that is used to compile your applications. If needed, you can create custom build scripts using Apache Ant, which is an open-source Java-based build tool.

You can apply custom builders to all the Flash Builder project types.

### Create a builder

- 1 In the Flex Package Explorer, select a project and then right-click (Control-click on Macintosh) to display the context menu and select Properties.
- 2 Select the Builders properties page. If you're using other Eclipse plug-ins, there can be more than one builder listed. Flash Builder provides a builder named Flex, which you cannot modify.
- 3 Select New.
- 4 In the Choose Configuration Type dialog box, select the appropriate configuration type. Flash Builder supports the program type. Select it and click OK to continue. From the new builder properties page, you define the builder properties and reference the Ant script (an XML file).
- 5 Click OK to apply it to the project.

Detailed information about working with Ant build scripts can be found in the Eclipse documentation at [help.eclipse.org/help31/index.jsp](http://help.eclipse.org/help31/index.jsp).

### Using multiple SDKs in Flash Builder

Flash Builder lets you change the version of the SDK that you use to compile your projects. You can select the SDK when you first create a project or at any time you are working on a project.

The combination of a framework and the compiler make up the SDK. If you select the Flex 4 SDK, then you are using the 4 version of the Flex framework SWC files, and the 4 version of the Flex compiler. You cannot use, for example, the Flex 4 compiler with the Flex 3 framework SWC files.

Using a different SDK can be useful if you are given a project that was developed using Flex Builder 3 (which uses the Flex 3 SDK), but you are running Flash Builder 4 (which uses the Flex 4 SDK by default). By selecting an older SDK to build with, you can maintain projects that have not been updated to be compatible with the latest version of the SDK. In addition, if you are currently working on a project for the Flex 3 SDK, but want to use the Flash Builder 4 features, you can upgrade your edition of Flash Builder, but then select the older SDK as the default SDK.

If you develop a project and then change the SDK, Flash Builder performs a full rebuild, not an incremental build. As a result, Flash Builder flags any differences that would throw compiler errors as if the project had been developed under the original SDK.

Flash Builder also regenerates all supporting files for the projects. These include the history management and deep linking files used by the HTML wrapper. For Flex 4 SDK projects, Flash Builder creates the Flex 4 SDK-compatible history.swf, history.html, and history.js history management files in the html-templates directory. For Flex 3 SDK projects, Flash Builder creates the Flex 3 SDK-compatible deep-linking history.htm, history.js, and historyFrame.html files in the html-templates/history directory.

In addition, the availability of Flash Builder options change depending on the selected SDK. For example, if you add a module to your project with a project that uses the Flex 3 SDK, Flash Builder does not let you select whether you want to optimize that module or not. You must do this manually.

For more information about the differences between the Flex 4 SDK and the Flex 3 SDK, see Backward compatibility.

When you create a Flex project, Flash Builder uses the default SDK. The default is the latest SDK that shipped with Flash Builder, but you can change it to any SDK that is visible in the list of available SDKs in Flash Builder.

When you create a Flex library project or ActionScript project, you can select which SDK you want that project to use in the New Flex Library Project and New ActionScript Project dialog boxes.

### Add a new Flex SDK to the list of available SDKs

- 1 Open the Preferences dialog and select Flash Builder > Installed Flex SDKs.  
The currently installed SDKs are listed. The default SDK has a check mark next to its name.
- 2 Click Add.
- 3 Enter the location of the SDK in the Flex SDK Location field.
- 4 Enter a name for the SDK in the Flex SDK Name field. Do not use the name of an existing SDK for this field.
- 5 Click OK to save your changes.
- 6 Click OK again to add the new SDK to the list of available SDKs. This list is maintained in the Flash Builder workspace, across Flex projects. The next time you create a project, the list of available SDKs includes this new SDK.

### Change the SDK version for the current project

- 1 Select Project > Properties.
- 2 Select Flex Compiler.
- 3 Click Use a Specific SDK.
- 4 Select the SDK you want to use from the drop-down list. If the SDK you want to use is not in the drop-down list, click the Configure Flex SDKs link.
- 5 Click OK.

Flash Builder applies the new SDK to the current project. Errors and warnings can appear if the project uses code that is not compatible with the new SDK.

### Select a new default SDK

- 1 Open the Preferences dialog, select Flash Builder > Installed Flex SDKs.  
The default SDK has a check mark next to its name.
- 2 Select the check box next to an SDK. This SDK becomes the default SDK. It is also applied to any project that has the Use Default SDK option selected in the Flex Compiler dialog box, including the current project. If the current project is set to use a specific SDK, then it continues to use that specific SDK, even if you change the workspace's default SDK to a different one.
- 3 Click OK to save your changes.
- 4 Click OK again.

## Alternatives to using project references

Project references can affect build order, so Flash Builder provides alternatives to using project references.

**Flex Library projects** The preferred way to create a reusable library. Flash Builder creates a project reference to ensure that the SWC project is built before the main project that includes it on the library path. Also, because Flash Builder adds it to the library path, code hints appear in the main project for the classes in the SWC project.

**Source path** The recommended way to include code in your project that is not under the same folder structure. This enables code hints in the project files and classes in related files, and the compiler knows where to find the source code. You can add any number of source paths to your project and they are displayed as linked folders in the Package Explorer.

## Flash Builder command line build using Apache Ant

Flash Builder provides the Ant task `<fb.exportReleaseBuild>`. Use this task to implement command line builds that synchronize a developer's individual build settings with the nightly build. Alternatively, you can use the `<mxmlc>` task in custom scripts for nightly builds.

**Note:** Command line build support is available only in Flash Builder Premium.

### `<fb.exportReleaseBuild>` task

Using the `<fb.exportReleaseBuild>` task ensures that the nightly build's settings exactly match the settings used by developers during their daily work.

For example, if a developer changes the library path of a Flex project, the new library path is written to the Flash Builder project. When the nightly build machine runs `<fb.exportReleaseBuild>`, that task loads the Flash Builder project and all of its settings.

Another advantage of using `<fb.exportReleaseBuild>` is that it automatically takes care of additional "housekeeping" tasks normally included in a Flash Builder build, such as:

- Automatically compile associated library projects
- Copy assets such as JPEGs, and so on, into the output directory
- Copy the HTML template, including macro substitution based on the compilation results (such as width and height)

**Note:** The `<fb.exportReleaseBuild>` task requires you to install Flash Builder on the nightly build machine.

### `<mxmlc>` task

If you write a custom script that uses the `<mxmlc>` task (for example, an Ant script), then you do not need to install Flash Builder on the build machine. However, the build machine is required to have the Flex SDK available. Thus, the build machine can be on a Linux platform.

However, the disadvantage of this approach is that you have two sets of build settings to synchronize. That is, one in Flash Builder, used by developers during their daily work, and another on your nightly build machine.

### `<fb.exportReleaseBuild>` usage

- 1 Install Flash Builder on a build machine.
- 2 Write `build.xml` with `fb.exportReleaseBuild` as a target. For example:

```
<?xml version="1.0"?>
<project default="main">
  <target name="main">
    <fb.exportReleaseBuild project="MyProject" />
  </target>
</project>
```

`build.xml` specifies to run a command line build of your Flex project, using the settings saved in your project files. See "[Parameters for `fb.exportReleaseBuild` task](#)" on page 106 for details on available parameters.

- 3 Create a nightly build script that tells Eclipse to look for a build file and execute its target.

The following examples specify `build.xml` as a build file, which executes `MyTarget`.

If your nightly build script is on a Macintosh platform, you could run the following script:



```
WORKSPACE="$HOME/Documents/Adobe Flash Builder"

# works with either FlashBuilder.app or Eclipse.app
"/Applications/Adobe Flash Builder/FlashBuilder.app/Contents/MacOS/FlashBuilder" \
  --launcher.suppressErrors \
  -noSplash \
  -application org.eclipse.ant.core.antRunner \
  -data "$WORKSPACE" \
  -file "$(pwd)/build.xml" MyTarget
```

If your nightly build is on a Windows platform, you could run the following batch file:

```
set WORKSPACE=%HOMEPATH%\Adobe Flash Builder

REM works with either FlashBuilderC.exe or eclipse.exe
"C:\Program Files\Adobe\Adobe Flash Builder 4.6\FlashBuilderC.exe" ^
  --launcher.suppressErrors ^
  -noSplash ^
  -application org.eclipse.ant.core.antRunner ^
  -data "%WORKSPACE%" ^
  -file "%cd%\build.xml" MyTarget
```

### **fb.running Ant property**

The `fb.running` Ant property has a value of `true` when Flash Builder is running. You can use this property when running scripts inside Flash Builder. For example:

```
<target name="myFlashBuilderTasks" if="fb.running">
  <fb.exportReleaseBuild ... />
</target>
```

### **Eclipse Ant Tasks**

Eclipse provides several Ant tasks you can incorporate as targets in your build script. For example:

- `eclipse.incrementalBuild`
- `eclipse.refreshLocal`
- `eclipse.convertpath`

For more information on these scripts, see the Eclipse documentation.

### Parameters for fb.exportReleaseBuild task

Attribute	Description	Required?	Default Value
application	The name of the application to compile. You can specify just the application name with no path or extension (for example: app1). To avoid ambiguity in naming, you can specify full path, relative to the project root (for example: src/app1.mxml). To compile all applications, specify '*', or omit this attribute. When running against an AIR project, you can only specify a single application. The '*' value is not allowed.	No	The default application of the project.
basefilename	For mobile and AIR desktop projects only: The name of the package to export. Specify the package name without the filename extension.	Yes	The name of the selected application
certificate	For mobile and AIR projects: When exporting an AIR or native-installer file, the path to the certificate used to sign the package.  Note: If omitted, the value of the certificate attribute in the project's .actionScriptProperties file is used. If the certificate attribute does not exist in the .actionScriptProperties file, an error message displays.	No	n/a
destdir	Specifies the output folder to export the release package. The folder can be a relative path or an absolute path. If you specify a relative path, it is relative to the root of the project.	No	"bin-release" for Web projects and "" ( indicates root of the project) for all other project types
failonerror	Indicates whether compilation errors cause the build to fail.	No	true
locale	Specifies the locale, for example, en-US. This value is passed to the compiler using the compiler's -locale flag. If specified, this locale value overrides any locale that has been specified in Flash Builder's Additional Compiler Arguments field.	No	n/a
packagetype	For mobile and AIR desktop projects: Specifies the package type to use when exporting the release package.  For mobile projects, you can specify "air", "airi", or "platform" package types. When you specify "air", a single platform-independent AIR file is generated. When you specify "airi", an airi file is generated for each platform. When you specify "platform", a platform-specific file is generated for each platform. For example, an APK file is generated for the Android platform and an IPA file is generated for the iOS platform.  For AIR desktop projects, you can specify "air", "airi", or "native" package types. Each option generates only one file.	Yes	n/a
password	For AIR projects only: The password for the certificate that is used to sign the AIR file. If this argument is omitted, an error message displays.  <b>Caution:</b> Specifying a literal value for a password can compromise security.	No	n/a

Attribute	Description	Required?	Default Value
platform	For mobile projects only: Specifies the target platform to export the release build. You can specify the platform as "android", "ios", or "qnx". Use a comma-separated list to specify multiple platforms.  <b>Note:</b> For a mobile project with "air" package type, this attribute is not required.	Yes	n/a
project	The project to build. Specify the name of a project in your Flash Builder workspace, without a path. For example, "MyFlexProject".	Yes	n/a
publishsource	Whether to publish the source of the application, allowing the user to view source files using the context menu View Source.	No	false
timestamp	For AIR projects only: Indicates whether the generated AIR file includes a timestamp.  Note: If omitted, the value of the timestamp attribute in the project's .actionScriptProperties file is used. If the timestamp attribute does not exist in the project's .actionScriptProperties file, the default value "false" is used.	No	false
verbose	The <fb.exportReleaseBuild> task outputs additional information. For example, it lists the files that were packaged into the AIR file and how long each step of the process took.	No	false

### Specify platform-specific attributes for mobile projects

You can export release packages to multiple platforms at the same time by specifying platform-specific attributes. You can use platform-specific attributes by specifying the platform-specific prefix. For example, to use the "certificate" attribute for the Android platform, specify "android.certificate", and for the iOS platform, specify "ios.certificate".

## Android platform

Attribute	Description	Required?
android.airDownloadURL	The URL to download the AIR Runtime if it is not installed on a user's device when launching the application.  <b>Note:</b> If you do not specify this attribute, then the default Android Market URL is used.	No
android.certificate	The path to the certificate that is used to sign the APK file.  You specify this attribute when you set the "packagetype" as "platform" and one of the target platforms is Android.	Yes, if you have not defined the certificate path for the Android platform in the .actionScriptProperties file.
android.password	The password for the certificate that is used to sign the APK file. If this attribute is omitted, an error message appears.	Yes

## iOS platform

Attribute	Description	Required?
ios.certificate	The path to the certificate (.p12 filename extension) that is used to sign the IPA file.  You specify this attribute when you set the "packagetype" as "platform" and one of the target platforms is Apple iOS.	Yes, if you have not defined the certificate path in the .actionScriptProperties file.
ios.packagetype	The package type that is used when exporting an IPA file. Specify the package type as "adhoc" for ad-hoc packaging and "appstore" for packaging to the Apple App store.	Yes, if you have not defined the package type in the .actionScriptProperties file.
ios.password	The password for the certificate that is used to sign the IPA file. If this argument is omitted, an error message displays.	Yes
ios.provisioning	The path to the provisioning file (.mobileprovision filename extension) that is used to sign the IPA file.	Yes, if you have not defined the provisioning file path in the .actionScriptProperties file.

## Export Release Build wizard

When you run the Export Release Build wizard (Project > Export Release Build), the settings you make in the wizard are saved in the .actionScriptProperties file. A command line build that uses fb.exportReleaseBuild task picks up the settings from the wizard. The Export Release Build wizard saves the following settings:

- View Source

The source files you specify for View Source are saved. If you specify the publishsource parameter to fb.exportReleaseBuild, then the wizard includes these files as viewable source files.

**Important:** For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See *Exporting source files with release version of an application*.

- For AIR projects, any additional output files that you specify in the wizard to include with the AIR or AIRI file.

## Run command line builds on Linux and other platforms

The <fb.exportReleaseBuild> task is only supported on Windows and Mac platforms.

However, if you are writing a build script for another platform, use the -dump-config option to the mxmcl or compc compiler to write compiler configuration settings to a file. You can then use the -load-config option to read the configuration options.

Modify the configuration settings in the file as necessary. For example, change `<debug>true</debug>` to `<debug>false</debug>` if your nightly build is supposed to do a release build.

### Run a command line build using Flash Builder compiler settings

- 1 In Flash Builder, select Project > Properties > Flex Compiler
- 2 In Additional Compiler Arguments, specify the following argument:  
`-dump-config pathname`, where *pathname* specifies the absolute path to a file on your system.
- 3 Apply the changes in the Project window.  
The compiler settings are written to the specified file. Remove the `-dump-config` argument after you have verified that the file has been written.
- 4 Modify the configuration settings as necessary.
- 5 In your build script, run the compiler so it includes the saved compiler settings:  
`mxmlc -load-config pathname`

### Limitations to command line builds

There are a few limitations to running command line builds using the `<fb.exportReleaseBuild>` task.

### Run command line builds on 64-bit platforms

Flash Builder runs on platforms that implement 32-bit Java. To run a command line build on platforms that support 64-bit Java (for example, Mac OS X Snow Leopard), add `-d32` to the command-line options that are passed to Java. For example:

```
java -d32 ...
```

## Publish source code

When your applications are ready to be released, Flash Builder lets you choose whether users can view source code and assets in the application. As with HTML, users can access and view the source in a web browser by selecting View Source from the context menu. The source viewer formats and colors the code so that it is easy to read. It is also a convenient way to share code with other Flex and ActionScript 3.0 developers.

### Enable the view source option

- 1 With the completed application project open in the editor, select Project > Export Release Build.
- 2 Select Enable View Source or Include Source for ActionScript projects.
- 3 Click Choose Source Files.
- 4 In the Publish Application Source dialog box, select the application file or files to include in the View Source menu. By default, the main application file is selected.  
***Important:** For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*
- 5 (Optional) Change the source output folder. By default, a source view folder is added to the project output folder.
- 6 Click OK.

When users run your application, they can access the source code by selecting View Source from the context menu. The source code appears in the default web browser as a source tree that reflects the structure of the resources (packages, folders, and files) contained in your application (the ones that you decided to publish). Selecting a source element displays the code in the browser. Users can also download the entire set of source files by selecting the Download.zip file link.

**Note:** *Because of Internet Explorer security restrictions, you are sometimes unable to view the source on your local development computer. If so, deploy the application to a web server to view the source.*

### Add the view source menu to ActionScript projects

In Flex projects, you add the View Source option to your application with the Export Release Build wizard. In ActionScript applications, you add this option manually.

The Flex framework contains the following function that you can use in an ActionScript application's constructor to enable the view source menu:

```
com.adobe.viewsource.ViewSource.addItem(obj:InteractiveObject, url:String,  
hideBuiltins:Boolean = true)
```

You can use the code in your ActionScript applications as shown here:

```
package {  
    import flash.display.MovieClip;  
    import com.adobe.viewsource.ViewSource;  
  
    public class MyASApp extends MovieClip  
    {  
        public function MyASApp()  
        {  
            ViewSource.addItem(this, "srcview/index.html");  
  
            // ... additional application code here  
        }  
    }  
}
```

This example demonstrates adding the view source menu using the default location of the source folder (srcview). If you change the location of the source folder, make sure that your code uses the correct location.

## Run and debug applications

Your applications are run (and debugged) based on a launch configuration. When you create new Flex and ActionScript applications, a launch configuration specifies the location of the built applications files and the main application file. You can modify the launch configuration or create custom launch configurations. For more information, see [“Create custom launch configurations”](#) on page 113.

You can run your projects in a number of ways in Flash Builder. For example, you can use the Run command, which is available from the workbench main menu and toolbar, from the Flex Package Explorer, and code editor pop-up menus.

**Note:** The Run button has two elements: the main action button, and a pop-up menu that shows the application files in the project that can be run or debugged. When you click the main action button, the default application file is run. Alternatively, you can click the pop-up menu and select any of the application files in the project and create or edit a launch configuration in the Create, Manage, and Run Configurations dialog box.

## Manage launch configurations

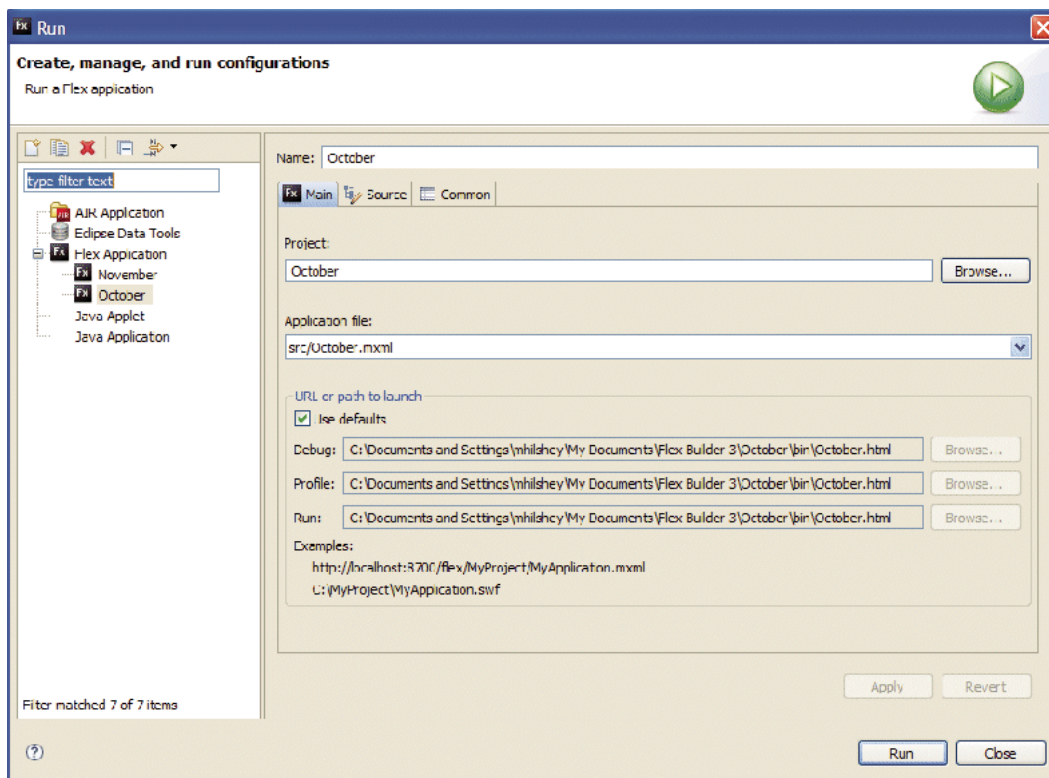
Launch configurations are used both to run and to debug applications. Flash Builder provides a default launch configuration for Flex and ActionScript applications.

A launch configuration defines the project name, main application file, and the path to the run and debug versions of the application. Flash Builder contains a default application launch configuration that is used to create launch configurations automatically for each of your projects.

### Create or edit launch configurations for web applications

When you create and build a project, it is ready to be run or debugged. Both running and debugging of the applications in your project are controlled by a launch configuration. By default, Flash Builder creates a launch configuration for each of the application files in your project the first time you run or debug them. The configurations are based on the default application configuration, and you can edit them as necessary.

Launch configurations are managed in the Create, Manage, and Run Configurations dialog box.



### Access and edit a launch configuration

- 1 In the Flex Package Explorer, select a project.
- 2 With a project file open in the code editor, select Run > Run Configurations.

- 3 Select the launch configuration to edit.
- 4 Modify the configuration preferences as needed, and click Run or Debug.

Running your web projects, opens the main application SWF file in your default web browser or directly in the stand-alone Flash Player.

### More Help topics

[“Change the default web browser”](#) on page 116

[“Run the application SWF file in the stand-alone Flash Player”](#) on page 116

### Create or edit launch configurations for desktop applications

You can create launch configurations for desktop applications to override the default configurations provided by Flash Builder.

When you create or edit a launch configuration for a desktop application, you specify command line arguments, a Publisher ID, a profile, and screen size parameters. For desktop applications, Profile can be either desktop or extendedDesktop.

To create a launch configuration:

- 1 Select Run > Run As  
For Debug launch configurations, specify Run > Debug As. For Profile launch configurations, specify Run > Profile As.
- 2 Select the desktop project. Click the New Launch Configuration button.  
Specify the configuration settings in the Main, Source, or Common tabs.
- 3 Select Run, Debug, or Profile to run the application with those settings.  
Click Close to save the settings.

### Create or edit launch configurations for mobile applications

Before you first run or debug a mobile application, open the Launch Configurations window to define a launch configuration. You can create one or more launch configurations for an application. The set of launch configurations you define is shared between running and debugging the application.

- 1 Select Run > Run Configurations to open the Run Configurations dialog.  
To open the Debug Configurations dialog, select Run > Debug Configurations See Run and debug a mobile application on a device.  
You can also access the Run or Debug Configurations menu from the drop-down list of the Run button or Debug button in the Flash Builder toolbar.
- 2 Expand the Mobile Application node. Click the New Launch Configuration button from the dialog toolbar.
- 3 Select a Target Platform from the drop-down list.



#### 4 Specify a Launch Method:

- On Desktop

Runs the application on your desktop using the AIR Debug Launcher, according to a specified device configuration. This launch method is not a true emulation of running the application on a device. However, it does allow you to view the application layout and interact with the application.

See Preview applications with ADL.

Click Configure to edit device configurations. See Set device configurations.

- On Device

Install and run the application on your device.

Flash Builder installs the application on your device and launches the application. Flash Builder accesses the device connected to your computer's USB port. See Run and debug a mobile application on a device for more information.

Windows platforms require a USB driver to connect an Android device to your computer. For more information, see Install USB device drivers for Android devices (Windows).

#### Check for connected devices

When you run or debug a mobile application on a device, Flash Builder checks for connected devices. If Flash Builder finds a single connected device online, Flash Builder deploys and launches the application. Otherwise, Flash Builder launches the Choose Device dialog for these scenarios:

- No connected device found
- Single connected device found that is offline
- Multiple connected devices found

The Choose Device dialog lists the devices and their state (online or offline). Select the device to launch.

#### Debug mobile applications

You can debug a mobile application from your development desktop or from a device. Before you can debug a mobile application, create a launch configuration. See [“Manage launch configurations”](#) on page 111.

Debugging on an Android device requires Android 2.2 or a later Android version installed on it. When you run an application on a device from Flash Builder, Flash Builder installs a debug version of the application on the device.

For more information, see Debug an application on a Google Android device.

Debugging on an Apple iOS device requires you to package the application as a debug iOS package format (IPA) file, and manually install the application on the connected device.

For more information, see Debug an application on an Apple iOS device.

**Note:** If you export a release build to a device, you install a non-debug version of the application. The non-debug version, however, is not suitable for debugging.

## Create custom launch configurations

Launch configurations differ, depending on whether you are running a web application, desktop application, or mobile application. You can customize the launch configurations that Flash Builder creates automatically for you.

- 1 From the Flash Builder menu, select Run > Run configurations.

- 2 In the Create, Manage, and Run Configurations dialog box, select the type of application you want to configure. Click the New button in the toolbar.
- 3 Specify a name, project, and application file for the launch configuration.
- 4 Modify the configuration properties as needed for the type of application you are configuring.
- 5 Click Apply to save the settings or click Run to run the application.

#### **Customize the launch configuration to launch the browser manually**

When you run the application, Flash Builder automatically launches the browser. You can, however, choose to manually launch the browser instead of Flash Builder launching it. To do so:

- 1 From the list of configurations, select the project's launch configuration.
- 2 On the Main tab, deselect the Use Default option.
- 3 Change the default URL or path to launch to about:blank.

#### **Run the last launched configuration**

- ❖ Click the Run button on the main toolbar.

If a launch configuration has to be created, Flash Builder opens the Launch configurations dialog.

#### **More Help topics**

[“Create or edit launch configurations for web applications”](#) on page 111

[“Create or edit launch configurations for desktop applications”](#) on page 112

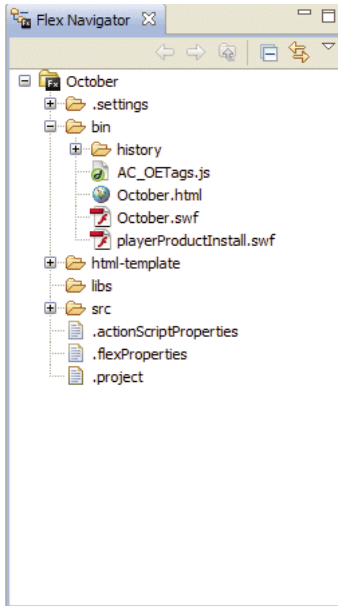
[“Create or edit launch configurations for mobile applications”](#) on page 112

## **Debug version of the application**

The debug version of your application contains debugging information that is used when you debug your application. The Export Release Build version does not include the additional debugging information and is therefore smaller in size than the debug version. An HTML wrapper file contains a link to the application SWF file and is used to run or debug your application in a web browser.

**Note:** Both the Run and Debug commands launch the development build in the bin-debug folder (not the exported release build folder, bin-release.)

In a standard application, a typical output folder resembles the following example:



You can run or debug your Flex and ActionScript applications in a browser, in AIR, or on your mobile device. You control how your applications are run or debugged by modifying the project's launch configuration (see [“Manage launch configurations”](#) on page 111).

### More Help topics

[“How to debug your application”](#) on page 127

## Use debugger version of Flash Player

By default, Flash Builder runs the debugger version of Flash Player. This version is available as a browser plug-in or ActiveX control, or as a stand-alone version. This version is installed with Flash Builder, but it is also available as a download from the Adobe website.

The installers for the debugger version of Flash Player are located in the *Adobe Flash Builder Installation/player* directory on your computer.

You can programmatically determine which version of Flash Player you are running by using the `Capabilities.isDebugger()` method. For more information, see [Determining Flash Player version in Flex](#).

When you launch or debug an application from Flash Builder, you can specify the Flash Player version to use.

### More Help topics

[“Stand-alone Flash Player version management”](#) on page 244

## Run and debug applications in a browser

By default, the launch configuration for web applications specifies that the run and debug paths point to the HTML wrapper files in the output folder of your project; therefore, your applications are run and debugged in Flash Player running in a web browser. Alternatively, you can run and debug your applications in the stand-alone Flash Player (see [“Run the application SWF file in the stand-alone Flash Player”](#) on page 116). You can also override the system default web browser setting and run your applications in any browser you have installed (see [“Change the default web browser”](#) on page 116).

### Change the default web browser

Flash Builder uses the system default web browser when running and debugging applications. While you cannot set each launch configuration to use a specific web browser, you can change the workbench web browser setting, which affects how all of your applications are run and debugged.

- 1 Open the Preferences dialog and select General > Web Browser.
- 2 Select a web browser from the list of web browsers installed on your system.

**Note:** *The Use Internal Web Browser option does not apply to running and debugging applications. Applications are always run and debugged in an external web browser.*

You can also add, edit, and remove browsers from the list.

- 3 Click OK to apply your changes.

## Run the application SWF file in the stand-alone Flash Player

You can choose to run and debug your applications in the stand-alone Flash Player by making a simple change to the launch configuration. You can specify the Flash Player version to use when you run and debug your applications. For more information, see [“Stand-alone Flash Player version management”](#) on page 244.

### Run and debug applications in the stand-alone Flash Player

- 1 From the Create, Manage, and Run Configurations dialog box, select the launch configuration you want to modify.
- 2 In the Main tab, deselect Use Defaults.
- 3 In either or both of the Run and Debug paths, click Browse.

The file selection dialog box appears and lists the contents of the build output folder.

- 4 Select the application SWF file in the bin-debug directory. Do not select the SWF file in the bin-release directory, if there is one. This SWF file does not contain debug information.
- 5 Click Open to select the file and return to the configuration dialog box.
- 6 Apply your changes and use the modified configuration to run or debug the application.

## Compare debug and non-debug versions of your application

By default, Flash Builder generates debug versions of your application’s SWF file and stores them in your project’s bin-debug directory. This application is larger than the non-debug version because it includes additional code and metadata that the debugger uses.

To generate a non-debug version of your application, you can do one of the following:

- Select Project > Export Release Build. Doing so, creates a non-debug SWF file or AIR file in the bin-release directory.

- Add `-debug=false` to the Additional Compiler Arguments field. Doing so, generates a non-debug SWF file no matter where you export it to.

## Export a release version of an application

After you have finished creating your application and want to publicly release it, you can export a release version of your application. Flash Builder's Export Release Build wizard creates an optimized release-quality version (non-debug SWF file or AIR file) of your application.

Depending on the type of application, the wizard guides you through steps to customize and optimize packaging.

After running the wizard, additional steps are required to deploy your application on a server.

### Web application (runs in Adobe Flash player)

- 1 Select Project > Export Release Build to open the Export Release Build wizard.
- 2 Select the project and application you want to export.
- 3 (Optional) Select Enable View Source to make application source files available from the exported application.  
Click View Source Files to specify which source files to include. In addition to the specified source files, the wizard generates a ZIP archive file containing the source files.

**Important:** For server projects, you can select the services folder when exporting source files. Exporting files that implement services have security implications. These files can expose access to your database, including user names and passwords. See *Exporting source files with release version of an application*.

- 4 Click Finish.
- 5 Copy the folder containing the exported release to the web root of the server hosting the application.
- 6 (Server projects) If exporting a release from a project that specified an application server type, deploy the services and other server-side files to the web root of the target server.

Maintain the same directory structure used during development.

This step applies to ColdFusion, PHP, BlazeDS, and Data Services. You specify the application server type when creating a project in Flash Builder.

If the application server is on a different host than the deployed application, a cross-domain policy file is necessary to access these services. This applies to projects accessing static XML service files or local files for HTTP services or web services. See *Using cross-domain policy files*.

- 7 (PHP server projects only) For PHP projects, perform these additional steps:
  - a Install the Zend framework on the server. See *Installing Zend Framework*.
  - b Modify the `amf-config.ini` file, which is in the output folder of the exported release:  
For `zend_path`, specify the absolute path to the Zend installation directory.  
Set `amf.production` to **true**.  
Update `webroot` with the absolute path to the web root on the server.

## Desktop application (runs in Adobe AIR)

- 1 (Optional) Change the server settings in the project properties.

An exported desktop application can only access services used during development. If you want to change the server for the exported desktop application, modify the project settings.

- 2 Select Project > Export Release Build to open the Export Release Build wizard.

- 3 Select the project and application you want to export.

**Note:** For AIR applications, Flash Builder cannot make application source files available from the exported application.

- 4 Specify the location to export the project. The default location is inside your project folder.

- 5 To export your AIR application, you must digitally sign your build by selecting one of the following:

- Signed AIR Package to export a digitally signed AIR package
- Signed Native Installer to create a digitally signed installer for the target platform, either Windows or Mac OS X
- Signed Application With Runtime Bundled to export the application with the AIR runtime bundled within the application.
- Intermediate AIRI package to export a file that can be signed later.

Click Next.

For more information about digital signage, see the Adobe Flash Builder documentation at “[Digitally sign your AIR applications](#)” on page 123

- 6 On the Digital Signature page:

Specify the digital certificate that represents the application publisher’s identity. To generate a self-signed certificate, click Create and enter data in the required fields.

- 7 On the Package Contents page, select the output files to include in the AIR or AIRI file.

- 8 Click Finish.

- 9 Copy the .air project to the target desktop.

- 10 (Server projects) If exporting a release from a project that specified an application server type, deploy the services on the target server.

This step applies to ColdFusion, PHP, BlazeDS, and LCDS services. You specify the application server type when creating a project in Flash Builder.

- 11 (PHP server projects only) For PHP projects, perform these additional steps:

- a Install the Zend framework on the server. See Installing Zend Framework.

- b Modify the `amf-config.ini` file, which is in the output folder of the exported release:

For `zend_path`, specify the absolute path to the Zend installation directory.

Set `amf.production` to `true`.

Update `webroot` with the absolute path to the web root on the server.

## Create an intermediate AIR file

You can create an intermediate AIR file that can be signed later. Use this option for testing only.

- 1 Select Project > Export Release Build.

Select the AIR project to export and the file to export the project to. Click Next.

- 2 Select Export an Intermediate AIR File That Will Be Signed Later option.

- 3 (Optional) Click Next. Select output files to include in the exported AIR file.

By default, all the files are included.

- 4 Click Finish.

After you have generated an intermediate AIR file, it can be signed using the AIR Developer Tool (ADT). For information on the ADT command line tool, see [Signing an AIR intermediate file with ADT](#) in the Adobe AIR documentation.

For more information about using the AIR Developer Tool (ADT), see the Adobe AIR documentation at [www.adobe.com/go/learn\\_fbairdevelopertool\\_en](http://www.adobe.com/go/learn_fbairdevelopertool_en).

## Running an application with captive runtime

Depending on whether you are exporting the package with the AIR runtime bundled to Windows or Mac, you have different procedures to run the application after final deployment.

### On Windows

- ❖ The application package contains the required application files and the runtime to run the application. Users can run the application immediately after packaging. You can also choose to use third-party tools to create a Windows Installer (MSI) from the exported application package.

### On Mac

- ❖ To run the application, drag the application package into the Applications folder.

## Change Flex projects to Adobe AIR projects

You can change the application type of a Flex project from Web (runs in Flash Player) to Desktop (runs in Adobe AIR). The following changes are made during the conversion:

- An AIR descriptor file is created for each application in the project.
- The launch configurations for the project are updated to properly launch in Adobe AIR.
- Settings for HTML wrapper are removed.
- Custom Flash Player settings are removed.
- The library path is modified to include airglobal.swc instead of playerglobal.swc.

During conversion, you can specify whether to change the base Application tags to WindowedApplication tags for each application in the project. If you choose to convert these tags, this is the only change to application code that occurs during conversion. After the conversion, inspect the attributes to the base tags to make sure that the application runs as intended in Adobe AIR.

## Change a web application project to a desktop application

- 1 Select the project that you want to convert.

The project should be a Flex project with the Web application type (runs in Flash Player)

- 2 From the context menu for the project, select Add/Change Project Type > Convert to Desktop/Adobe AIR project.
- 3 In the Convert to Desktop/Adobe AIR Project dialog, specify whether to rewrite code:

- Convert Application Tags to WindowedApplication Tags

For existing applications in the project, all Application tags are rewritten to WindowedApplication tags. No other change to your code occurs. Inspect attributes to the base tags to make sure that the application runs as intended in Adobe AIR.

New applications you create in the project are desktop applications and can run in Adobe AIR.

- Do Not Rewrite Any Code

No changes are made to your code. Edit any applications in the project before they can run in Adobe AIR.

New applications you create in the project are desktop applications and can run in Adobe AIR.

*Note: This procedure cannot be undone.*

## Mobile devices (runs in Adobe AIR)

- 1 Select Project > Export Release Build to open the Export Release Build wizard.
- 2 Select the project and application that you want to export.
- 3 Select the target platforms.
- 4 Specify the location to export the project. You can export the project to a target desktop or to a connected device.
- 5 The base filename is the name of the project, by default. You can change the filename, if necessary.
- 6 To package your application with a digital signature for each target platform, select Signed Packages For Each Target Platform.

To package your application as a digitally signed AIR application for the desktop, select Signed AIR Package For Installation On Desktop.

If you want to export a file that can be signed later, export an intermediate AIRI file.

To generate a self-signed certificate, click Create to enter data in the required fields.

Click Next.

For more information about digital signage, see the Adobe Flash Builder documentation “[Digitally sign your AIR applications](#)” on page 123

- 7 On the Packaging Settings page, you can specify the digital certificate and the package contents. Depending on the target platform, the settings vary. These settings are used when packaging an application using the Run/Debug or Export Release Build workflows.

In the Digital Signature page, specify the P12 digital certificate that represents the application publisher’s identity. You can also specify a password for the selected certificate.

For the Apple iOS platform, convert the Apple iPhone certificate to the P12 file format and select a provisioning file. To do so, you must first obtain an iPhone developer certificate and a mobile provisioning profile from Apple.

For more information, see Prepare to build, debug, or deploy an iOS application.

You can then choose to deploy the final release package to the Apple App store or as an ad-hoc package for limited distribution.

In the Package Contents page, select the output files that you want to include in the final version of your application.



**Note:** You cannot exclude the SWF file and the application descriptor file as they are required for exporting a valid package.

The following files are not needed and hence are automatically deselected or hidden:

- .ipa files
- certificate files (.p12, .pfx, .provisioning)
- .air, .airi files

For the Google Android platform, Select Install And Launch Application On Any Connected Devices to export the project to a connected device. The exported application installs on the device.

If AIR is not already installed on a user's device, you can select or specify a URL to download AIR for the application package. The default URL points to a location on the Android App store. You can, however, override the default URL and select the URL that points to a location on the Amazon Appstore, or enter your own URL.

#### 8 Click Finish.

For more information, see Package and export a mobile application.

## Package AIR applications

### Package AIR applications

When your application is complete and ready to be distributed (or tested running from the desktop), you package it into an AIR file. Packaging consists of the following steps:

- Selecting the AIR application you want to publish
- Optionally allowing users to view the source code and then selecting which of your application files to include
- Selecting a generic AIR file or a native installer to install the application
- Digitally signing your AIR application using a commercial code signing certificate or by creating and applying a self-signed signature
- Optionally choosing to create an intermediate AIR file, which can be signed at a later time

### Package an AIR application

- 1 Open the project and ensure that the application has no compilation errors and runs as expected.
- 2 Select Project > Export Release Build.
- 3 If you have multiple projects and applications open in Flash Builder, select the specific AIR project you want to package.
- 4 Optionally select Enable View Source if you want users to be able to see the source code when they run the application. You can select individual files to exclude by selecting Choose Source Files. By default all the source files are selected. For more information about publishing source files in Flash Builder, see the Flash Builder Help.

**Important:** For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See *Exporting source files with release version of an application*.

- 5 Select Export to AIR File or Export to Native Installer. Click Next.

Export to AIR File creates a generic installer file that can be used to install the application on either Windows or Mac OS X platforms.

Export to Native Installer creates an installer for the target platform, either Windows or Mac OS X.

- 6 You can also optionally change the name of the AIR file that is generated. When you're ready to continue, click Next to digitally sign your application.

## Project properties for packaged desktop projects

You can manage the package's digital certificate and contents by selecting a project and viewing its properties.

- 1 In the Package Explorer, select a project.
- 2 Select Project > Properties from the main menu or select Properties from the context menu.
- 3 In the Project Properties dialog box, select Flex Build Packaging.
- 4 On the Digital Signature page:  
Select the digital certificate that you want to use to package and sign the AIR application. To generate a self-signed certificate, click Create, and enter data in the required fields.
- 5 On the Package Contents page:  
Deselect the files that you don't want to include in the final version of your application.
- 6 Click OK.

## Project properties for packaged mobile projects

You can manage the package's digital certificate and contents by selecting a project and viewing its properties.

- 1 In the Package Explorer, select a project.
- 2 Select Project > Properties from the main menu or select Properties from the context menu.
- 3 In the Project Properties dialog box, select Flex Build Packaging, and select the target platform.
- 4 In the Digital Signature page:  
Select the digital certificate that you want to use to package and sign the mobile application.  
  
For the Apple iOS platform, select a provisioning file. For more information, see Apple iOS development process using Flash Builder.  
  
For the Google Android platform, you can select a digital certificate or generate a self-signed certificate. To generate a self-signed certificate, click Create, and enter data in the required fields.
- 5 In the Package Contents page:  
Deselect the files that you don't want to include in the final version of your application.
- 6 In the Permissions page:  
To change mobile application permissions for Android, edit the application descriptor file (*app\_name*-app.xml).  
For more information, see Choose mobile application permissions.
- 7 Click OK.

## Digitally sign your AIR applications

When exporting your release version of your application, you will need to digitally sign your AIR application. You have the following options:

- You can sign the application using a commercial code signing certificate.
- You can create and use a self-signed digital certificate.
- You can package the application now and sign it later.

Digital certificates issued by certification authorities such as VeriSign, Thawte, GlobalSign, and ChosenSecurity assure your users of your identity as a publisher. The digital certificates also verify that the installation file has not been altered since you signed it. Self-signed digital certificates serve the same purpose but they do not provide validation by a third party.

You also have the option of packaging your AIR application without a digital signature by creating an intermediate AIR file (.air). An intermediate AIR file is not valid in that it cannot be installed. It is instead used for testing (by the developer) and can be launched using the AIR ADT command line tool. AIR provides this capability because in some development environments a particular developer or team handles signing. This practice insures an additional level of security in managing digital certificates.

For more information about signing applications, see [Signing AIR applications](#) in Adobe AIR documentation.

### Digitally sign your AIR application

You can digitally sign your AIR application by selecting an existing digital certificate or by creating a new self-signed certificate.

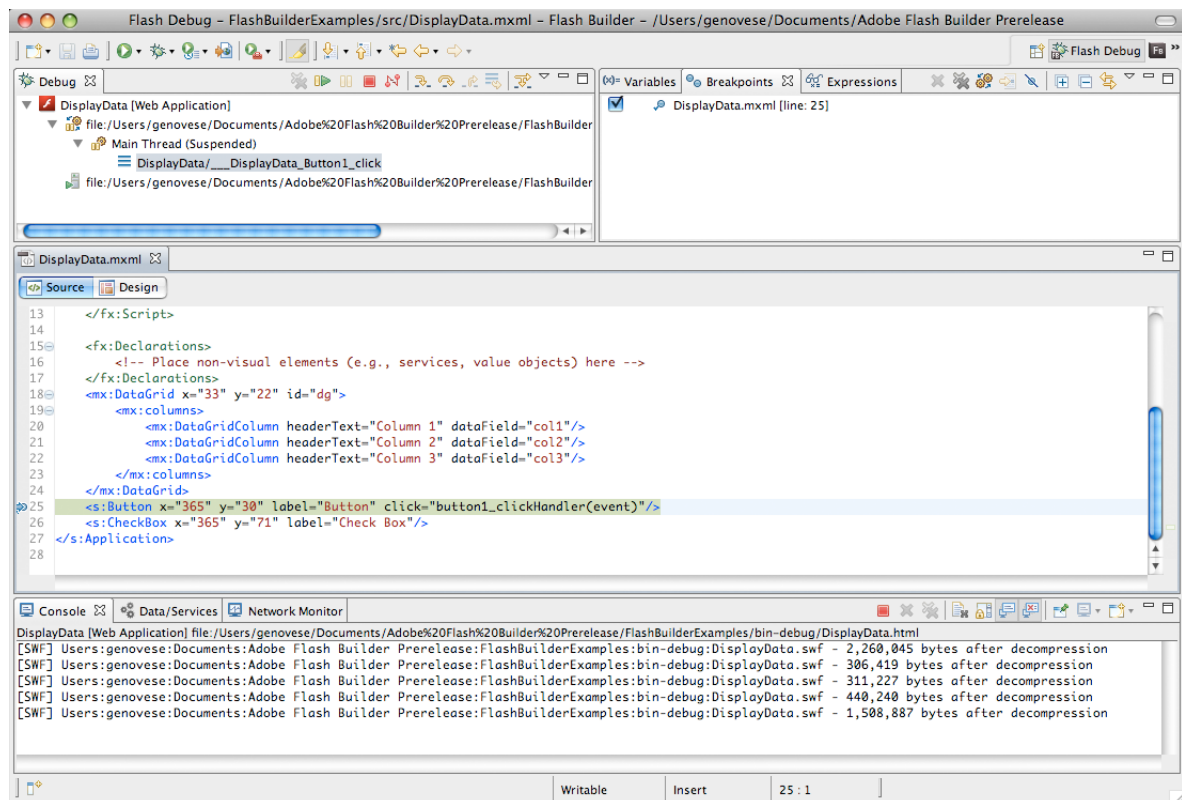
- 1 Select Project > Export Release Build.  
Select the AIR project to export and the file to export the project to. Click Next.
- 2 Select the Export and Sign an AIR File with a Digital Certificate option.
- 3 If you have an existing digital certificate, click Browse to locate and select it.
- 4 To create a new self-signed digital certificate, select Create.
- 5 Enter the required information and click OK.
- 6 (Optional) Click Next. Select output files to include in the exported AIR file.  
By default, all the files are included.
- 7 Click Finish to generate the AIR file.

# Chapter 5: Debugging Tools in Flash Builder

## The Flash Debug perspective

The Flash Debug perspective provides an extensive set of debugging tools. The debugging tools allow you to:

- Set and manage breakpoints
- Determine how to suspend, resume, and terminate the application
- Step into and over code
- Watch variables
- Evaluate expressions



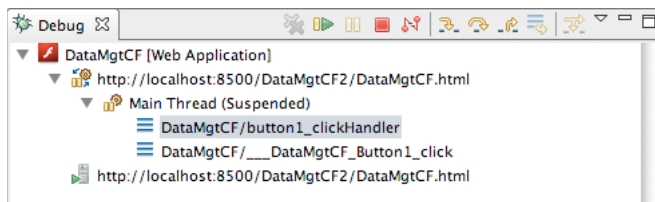
You add breakpoints to executable lines of code in the code editor. The Debug perspective appears automatically when the first breakpoint is reached. The Flash Debug perspective is then activated and you can inspect the state of and manage the application by using the debugging tools. For more information, see [“Start a debugging session”](#) on page 127.

You can also switch to the Debug perspective manually by selecting it from the Perspective bar. The Perspective bar is at the right edge of the main workbench toolbar. The Debug perspective contains Debug, Breakpoints, Console, Variables, and Expressions views.

## The Debug view

The Debug view is the control center of the Flash Debug perspective. You use it to control the execution of the application, to suspend, resume, or terminate the application, or to step into or over code.

The Debug view (in other debuggers it is sometimes referred to as the *callstack*) displays the stack frame of the suspended thread of the application you are debugging.

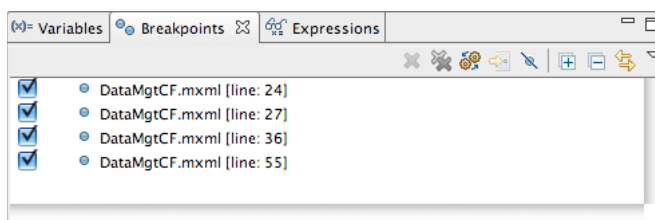


Applications built with Flex are single-threaded (not multi-threaded like Java, for example) and you can debug only one application at a time. Therefore, when you debug an application, you see only the processes and Debug view for a single thread of execution.

The Debug view shows a list of all the functions called to that point, in the order called. For example, the first function called is at the bottom of the list. You can double-click a function to move to it in the script; Flash Builder updates the information in the Variables view to reflect the new location in the script.

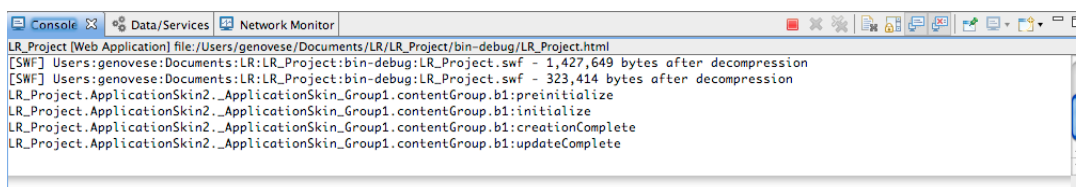
## Breakpoints view

The Breakpoints view lists all of the breakpoints you set in your project. You can double-click a breakpoint and display its location in the editor. You can also disable, skip, and remove breakpoints.



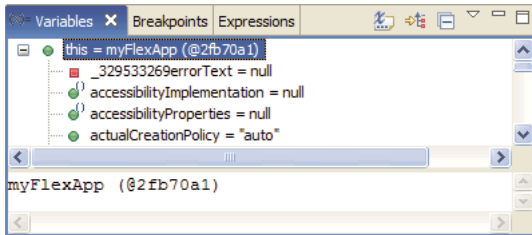
## Console view

The Console view displays the output from trace statements placed in your ActionScript code and also feedback from the debugger like, status, warnings, errors, and so on.

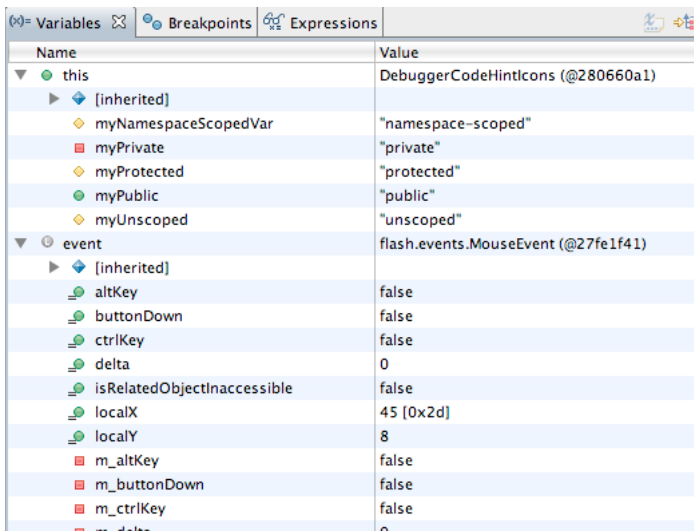


## Variables view

The Variables view displays information about the variables in a selected stack frame. You can select variables to monitor (in the Expressions view) and also change variable values during the debugging session. During the debug session, you can see the changes in the currently running SWF file and experiment with fixes for the problem you have to resolve.



The Variables view uses icons and overlays to provide visual cues about the type of variable.



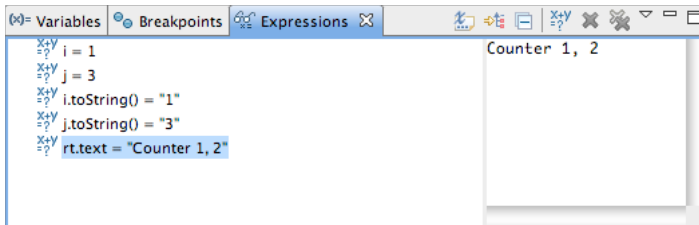
The Variables view

Complex variables can be expanded to display their members. You use the Variables view to watch variables by adding them to the Expressions view and to modify the value of variables during the debugging session. You can also set watchpoints in the Variables view, as described in [“Use watchpoints”](#) on page 132.

All superclass members are grouped in a separate tree node; by default you see only the members of the current class. This grouping helps reduce excess numbers of variables that are visible at one time in Variables view.

## Expressions view

The Expressions view is used to monitor a set of critical variables. You can choose the variables you consider critical in the Variables view and add them to the Expressions view for monitoring. You can also add and evaluate watch expressions. Watch expressions are code expressions that are evaluated whenever debugging is suspended.



### More Help topics

[“Run and debug applications”](#) on page 110

[“How to debug your application”](#) on page 127

## How to debug your application

After your projects are built in to applications (see [“Build projects”](#) on page 93), you can debug them in Flash Builder.

Debugging is similar to running your applications. However, when you debug you control when the application stops at specific points in the code. You can also control whether you want it to monitor important variables, and you can test fixes to your code.

Both running and debugging use a configuration to control how applications are launched. When you debug your application, you run the debug version of the application file. See [“Debug version of the application”](#) on page 114.

You can edit the launch configuration to change the default main application file. You can also modify the default launch path to run or debug in the stand-alone Flash Player rather than in a web browser. See [“Run the application SWF file in the stand-alone Flash Player”](#) on page 116.

While debugging mobile applications, you use the launch configuration to specify whether to launch the application on the desktop or on a device connected to your computer.

### More Help topics

[“The Flash Debug perspective”](#) on page 124


[“Run and debug applications”](#) on page 110

[“Eclipse environment errors in the log file”](#) on page 95

## Start a debugging session

To begin a debugging session, you run the application launch configuration in the Flash Debug perspective.

### Debug an application

- 1 In the Flex Package Explorer, select the project to debug.
- 2 Click  on the main workbench toolbar.

**Note:** The Debug button has two elements: the main action button and a drop-down list. The drop-down list shows the application files in the project that can be run or debugged. When you click the main action button, the project's default application file is debugged. You can alternatively click the drop-down list and select any of the application files in the project to debug. You can also access the launch configuration dialog box and create or edit a launch configuration by selecting the Debug command.

You can also select Run > Debug.

If your project has not been built yet, Adobe® Flash® Builder™ builds and runs it in debug mode.


- 3 Your application appears in your default web browser or the stand-alone Flash Player and you can then use the Flash Builder debugger to interact with it.
- 4 When a breakpoint is reached, the Flash Debug perspective is activated in the workbench.

### Start a debugging session in the plug-in configuration

The Debug command works differently in the plug-in configuration of Flash Builder. Instead of running the selected project, it debugs the most recently launched configuration. You can also select from a list of recently launched configurations.

### Debug an AIR application

Flash Builder provides full debugging support for AIR applications.

- 1 Open a source file for the application (such as an MXML file) in Flash Builder.
- 2 Click  on the main workbench toolbar.

The application launches and runs in the ADL application (the AIR Debugger Launcher). The Flash Builder debugger catches any breakpoints or runtime errors and you can debug the application like any other application that is built in Flex.

You can also debug an application from the command line, using the AIR Debug Launcher command-line tool. For more information, see *Using the AIR Debug Launcher (ADL)* in the AIR documentation.

## Add and remove breakpoints

You use breakpoints to suspend the execution of your application. By doing so, you can inspect your code and use the Flash Builder debugging tools to explore options to fix any errors. You add breakpoints in the code editor and then manage them in the Breakpoints view when you debug your applications.

You add breakpoints to executable lines of code. The debugger stops only at breakpoints set on lines that contain the following:

- MXML tags that contain an ActionScript event handler, such as `<mx:Button click="dofunction()" ...>`
- ActionScript lines such as those enclosed in an `<mx:Script>` tag or in an ActionScript file
- Any executable line of code in an ActionScript file

You can set breakpoints as you write code or while you debug.

### Set a breakpoint in the code editor

- 1 Open a project file that contains ActionScript code.
- 2 Locate the line of code on which you want to set a breakpoint, and double-click in the marker bar to add a breakpoint.

The marker bar is along the left edge of the code editor.



A breakpoint marker is added to the marker bar and to the list of breakpoints in the Breakpoints view of the Flash Debug perspective.

When the debugger encounters a breakpoint, the application is suspended, the Flash Debug perspective is displayed, and the line of code is marked with a breakpoint. The line of code is highlighted in the code editor. You then use the debugging commands in the Breakpoints view toolbar to interact with the code. See [“Breakpoints view”](#) on page 125.

### Remove a breakpoint in the code editor

- ❖ In the marker bar, double-click an existing breakpoint.

The breakpoint is removed from the marker bar and the Breakpoints view of the Flash Debug perspective.

### Remove breakpoints in the Breakpoints view

You can remove one, a few, or all of the breakpoints in the Breakpoints view from the Breakpoints toolbar.

- Select one or more breakpoints from the list of breakpoints, and then click Remove Selected Breakpoints.
- To remove all the breakpoints in a single action, click Remove All Breakpoints.

You can also disable the breakpoints and re-enable them at a later time.

## Set conditional breakpoints

You can specify conditions for breakpoints to stop the debugger from executing when specific conditions are met. When you set a conditional breakpoint, you specify an ActionScript expression that is evaluated during the debugging session. You configure the conditional breakpoint to stop execution for any of the following conditions:

- The expression evaluates to true.
- The value of the expression changes.
- A specified Hit Count has been reached.

### How to set a conditional breakpoint

- 1 From the context menu for a breakpoint, select Breakpoint Properties.
- 2 In the Breakpoint Properties dialog, specify any of the following:

- Enabled

Toggle to enable or disable the breakpoint.

- Hit Count

Select Hit Count to enable a counter for the breakpoint. Specify a number for the Hit Count.

If you specify both Hit Count and Enable Condition, the Hit Count is the number of times that the specified condition is met (evaluates to true or the value of the condition changes).

If you specify Hit Count only, then Hit Count is the number of times the breakpoint has been reached.

- Enable Condition

Select Enable Condition and enter an ActionScript expression to evaluate. See [“Examples of expressions”](#) on page 130 for information on types of expressions supported for evaluation.

**Note:** Flash Builder checks the syntax of the expression and notifies you of syntax errors. If you have an assignment operator in the expression, Flash Builder displays a warning.

- Suspend when:

Specify when to stop execution, either when the expression for the condition evaluates to true or the value of the expression changes.

## Manage variables in the Variables view

### Change the value of a variable

- 1 Select the variable to modify.
- 2 Right-click (Control-click on Macintosh) to display the context menu and select Change Value.
- 3 Enter the new value and click OK.

The variable contains the new value.

Modified variables are displayed in red.

### Find variables

- ❖ To locate a variable or variable member in the Variables view, with the Variables view selected, begin entering the name of the variable you're looking for. You can also use the wildcard character (\*) to search for words that occur anywhere within a variable name (for example, "\*color").

## Use the Expressions view

While debugging, you can inspect and modify the value of the variables that you selected to watch. You can also add watch expressions, which are code expressions that are evaluated whenever debugging is suspended. Watch expressions are useful for watching variables that may go out of scope when you step into a different function and are therefore not visible in the view.

You can also hover the pointer over an expression or variable in the source editor to see the value of that expression or variable as a tooltip. You can add the expression to the Expressions view by right-clicking and selecting Watch from the menu.

### Examples of expressions

The Flash Builder Debugger supports a wide range of simple and complex expressions. The following table lists examples of expressions that can be evaluated during a debugging session. This is not the complete list of expressions supported, but just a sampling of what you can do.

## Examples of supported expressions

Expression	Description
<code>myString.length</code>	Returns the length of a string.
<code>myString.indexOf('@')</code>	Tracks the index of the '@' character.
<code>"constant string".charAt(0)</code>	Tracks the character at a specific position in a string. String constants are supported.
<code>employees.employee.@name</code>	<code>employees</code> is an XML variable. This type of expression is useful for debugging E4X applications.
<code>x == null</code>	Reserved words representing values in expressions.
<code>user1 === user2</code>	Most ActionScript operators are supported.
<code>MyClass.myStaticFunc()</code>	Functions resolved to a class.
<code>this.myMemberFunc()</code>	Functions resolved using the keyword <code>this</code> .
<code>String.fromCharCode(33)</code>	<code>String</code> is actually a function, not a class, and <code>String.fromCharCode</code> is actually a dynamic member of that function.
<code>myStaticFunc()</code>	Can be valued only if <code>myStaticFunc</code> is visible from the current scope chain
<code>myMemberFunc()</code>	Can be valued only if <code>myMemberFunc</code> is visible from the current scope chain.
<code>Math.max(1,2,3)</code>	Math functions are supported.
<code>mystring.search(/myregex/i)</code>	Regular expressions are supported.
<code>["my", "literal", "array"]</code>	Creation of arrays.
<code>new MyClass()</code>	Instantiation of classes.
<code>"string" + 3</code>	Correctly handles string plus Integer.
<code>x &gt;&gt;&gt; 2</code>	Logical shift operations supported.
<code>3.5 + 2</code>	Performs arithmetic operations correctly.

## Limitations of expression evaluation

There are some limitations to expression evaluation.

- Namespaces are not supported.
- Inline objects are not supported.
- The keyword `super` is not supported.
- Fully qualified class names are not supported.

For example, you cannot evaluate `mx.controls.Button`.

You can refer to the unqualified class name. For example, you can specify `Button` to refer to `mx.controls.Button`.

If a class name is ambiguous (two classes with the same name in different packages,) then you cannot control which class will be evaluated. However, you can specify:

```
getDefinitionByName("mx.controls.Button")
```

- Most E4X expressions can be evaluated, but E4X filter expressions are not supported.

For example, you cannot evaluate `myxml.(@id=='3')`.

- You cannot call functions that are defined as a variable.

## Use watchpoints

When debugging an application, you can set watchpoints on specific instances of variables to stop execution when the watched variable changes value. Because watchpoints are set on a specific instance of a variable, you cannot set the watchpoint in the code editor. Instead, you set a watchpoint from the Variables view during a debugging session.

When setting watchpoints, keep in mind the following:

- When a debugging session ends, all watchpoints are removed.
- You cannot set watchpoints on getters, but you can set them on the field of a getter.

For example, you cannot set a watchpoint on `width`, but you can set a watchpoint on `_width`.

- You cannot set watchpoints on local variables, but you can set watchpoints on members of local variables, as illustrated in the following code fragment.

```
public class MyClass
{
    // These are fields of a class, so you can set a watchpoint on
    // 'memberInt', and on 'memberButton', and on 'memberButton._width':
    private var memberInt:int = 0;
    private var memberButton:Button = new Button();

    public function myFunction():void {
        // You CANNOT set a watchpoint on 'i', because it is local:
        var i:int = 0;

        // You CANNOT set a watchpoint on 'someButton', because it is local;
        // but you CAN set a watchpoint on 'someButton._width':
        var someButton:Button = new Button();

        ...
    }
}
```

- Execution stops for a watchpoint when the original value of an object instance changes.

This differs from using an expression in a conditional breakpoint to stop execution whenever a variable changes value.

### Set watchpoints

❖ In a debugging session, there are two ways to set a watchpoint:

- In the Variables view, open the context menu for a variable, and select Toggle Watchpoint
- From the Flash Builder Run menu, select Add Watchpoint.

From the Add Watchpoint dialog, select the variable you want to watch.

The Variables view displays a “pencil icon” to indicate that a watchpoint has been set on that variable.

**Note:** If you attempt to set a watchpoint on a getter, Flash Builder opens a dialog suggesting a valid variable for the watchpoint. If you delete the suggested variable, the dialog lists all valid variables for the object.

## Use Run to Line

Flash Builder provides the Run to Line command to break out of a loop during a debugging session.

While debugging, you sometimes find that your code is executing a loop that repeats many times. To break out of this loop, use the Run to Line command, available from the Run menu.

# Chapter 6: Profiling Tools in Flash Builder

Use the profiler in Adobe Flash® Builder™ to identify performance bottlenecks and memory leaks in your applications. The profiler is available only in Flash Builder Premium.

## The Flash Profiling perspective

Flash Builder Premium contains an additional perspective - the Flash Profiling perspective. The Profiling perspective displays several panels (or views) that present profiling data in different ways. As you interact with your application, the profiler records data about the state of the application, including the number of objects, the size of those objects, the number of method calls, and the time spent in those method calls.

### Profiler views

The profiler is made up of several views (or panels) that present profiling data in different ways. The following table describes each of these views:

View	Description
Profile	Displays the currently connected applications, their status, and all the memory and performance snapshots that are associated with them.  Initially, profiling sessions start with no recorded performance or memory snapshots.
Saved Profiling Data	Displays a list of saved snapshots, organized by application. You can load saved profiling data by double-clicking the saved snapshot in this list.  For more information, see <a href="#">“Save and load profiling data”</a> on page 152.
Live Objects	Displays information about the classes used by the current application. This view shows which classes are instantiated, how many were created, how many are in the heap, and how much memory the active objects are taking up.  For more information, see <a href="#">“View information in the Live Objects view”</a> on page 134.
Memory Snapshot	Displays the state of the application at a single moment in time. Contrast this with the Live Objects view, which is updated continuously. The Memory Snapshot view shows how many objects were referenced and used in the application and how much memory each type of objects used at that time.  You typically compare two memory snapshots taken at different times to determine the memory leaks that exist between the two points in time.  You view the Memory Snapshot view by clicking the Take Memory Snapshot button and then double-clicking the memory snapshot in the Profile view.  For more information, see <a href="#">“Memory Snapshot view”</a> on page 135.
Loitering Objects	Displays the objects that were created between two memory snapshots and still exist in memory or were not garbage collected. You can double-click a class name in the table to open the Object References view. This lets you examine the relationship between the selected objects and the other objects.  You view the Loitering Objects view by selecting two memory snapshots and clicking the Loitering Objects button.  For more information, see <a href="#">“Loitering Objects view”</a> on page 144.

View	Description
Allocation Trace	<p>Displays method statistics when comparing two memory snapshots.</p> <p>You view the Allocation Trace view by selecting two memory snapshots and then clicking the View Allocation Trace button.</p> <p>For more information, see <a href="#">“Allocation Trace view”</a> on page 139.</p>
Object References	<p>Displays objects and the objects that reference them.</p> <p>You view the Object References view by double-clicking a class name in the Memory Snapshot or Loitering Objects views.</p> <p>For more information, see <a href="#">“Object References view”</a> on page 137.</p>
Object Statistics	<p>Displays details about the caller and callee of the selected group of objects.</p> <p>You view the Object Statistics view by double-clicking an entry in the Allocation Trace view.</p> <p>For more information, see <a href="#">“Object Statistics view”</a> on page 140.</p>
Performance Profile	<p>Displays how the methods in the application performed during a given time interval. You then click a method name in the table to open the Method Statistics view, which lets you locate performance bottlenecks.</p> <p>You view the Performance Profile view by double-clicking one of the performance snapshots in the Profile view.</p> <p>For more information, see <a href="#">“Performance Profile view”</a> on page 141.</p>
Method Statistics	<p>Displays the performance statistics of the selected group of methods.</p> <p>You view the Method Statistics view by double-clicking a row in the Performance Profile view or selecting a method in the Performance Profile and clicking the Open Method Statistics button.</p> <p>For more information, see <a href="#">“Identify method performance characteristics”</a> on page 143.</p>
Memory Usage	<p>Graphically displays peak memory usage and current memory usage over time.</p> <p>For more information, see <a href="#">“Memory Usage graph”</a> on page 145.</p>

## View information in the Live Objects view

The Live Objects view displays information about the classes that the current application uses. This view shows the classes that are instantiated, how many were created, how many are in memory, and how much memory the active objects are taking up.

The profiler updates the data in the Live Objects view continually while you profile the application. You do not have to refresh the view or keep focus on it to update the data.

To use the Live Objects view, enable memory profiling when you start the profiler. This is the default setting. If you close the Live Objects view and want to reopen it, open the drop-down list in the Profile view and select Watch Live Objects.

The following example shows the Live Objects view:

Class	Package (Filtered)	Cumulative Instances	Instances	Cumulative Memory
Employee	valueObjects	1006 (32.3%)	1000 (48.57%)	244000 (78.38%)
_EmployeeEntityMetadata	valueObjects	1006 (32.3%)	1000 (48.57%)	60000 (19.27%)
DataMgtCF		1 (0.03%)	1 (0.05%)	1476 (0.47%)
MethodQueueElement	UIComponent.as\$168	62 (1.99%)	0 (0.0%)	848 (0.27%)
_DataMgtCF_mx_managers_SystemManager		1 (0.03%)	1 (0.05%)	728 (0.23%)
_mx_managers_CursorManagerStyle__embed_css_Assets_swf_r		1 (0.03%)	1 (0.05%)	424 (0.14%)
_e5e0a8b749c58154626094e667bd3c593a478f9a8720b4c6f5		1 (0.03%)	1 (0.05%)	404 (0.13%)
_Se891232bd6d81b9d6974dfdba52e58bf92fe717751ed14a87b		1 (0.03%)	1 (0.05%)	404 (0.13%)
_12bb6e99c417300f479d05e018fdc231181d5e4fbd4451087bc		1 (0.03%)	1 (0.05%)	404 (0.13%)
_0dc790747641a559f6e86f8aa121d894dc3470a66614c311433		1 (0.03%)	1 (0.05%)	404 (0.13%)
ListCollectionViewCursor	ListCollectionView.as\$85	22 (0.71%)	2 (0.1%)	304 (0.1%)
LineSegment	Path.as\$138	12 (0.39%)	12 (0.58%)	288 (0.09%)
RPCDataServiceRequest	RPCDataServiceAdapter.as\$141	1 (0.03%)	1 (0.05%)	260 (0.08%)
Vector.<*>	__AS3__vec	5 (0.16%)	5 (0.24%)	220 (0.07%)
Vector.<int>	__AS3__vec	8 (0.26%)	4 (0.19%)	160 (0.05%)
Vector.<Number>	AS3 .vec	8 (0.26%)	4 (0.19%)	160 (0.05%)

The following table describes the columns in the Live Objects view:

Column	Description
Class	The classes that have instances in the currently running application.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Cumulative Instances	The total number of instances of each class that have been created since the application started.
Instances	The number of instances of each class that are currently in memory. This value is always smaller than or equal to the value in the Cumulative Instances column.
Cumulative Memory	The total amount of memory, in bytes, that all instances of each class used, including classes that are no longer in memory.
Memory	The total amount of memory, in bytes, that all instances of each class currently use. This value is always smaller than or equal to the value in the Cumulative Memory column.

You typically use the data in the Live Objects view to see how much memory the objects use. As objects are garbage collected, the number of instances and memory use decrease, but the cumulative instances and cumulative memory use increase. This view also tells you how memory is used while the application is running.

For more information on running and analyzing the results of garbage collection, see [“Garbage collection”](#) on page 154.

You limit the data in the Live Objects view by using the profiler filters. For more information, see [“Profiler filters”](#) on page 158.

## Memory Snapshot view

The Memory Snapshot view displays information about the application’s objects and memory usage at a particular time. Unlike the Live Objects view, the data in the Memory Snapshot view is not continually updated.

To use the Memory Snapshot view, enable memory profiling when you start the profiler. This setting is the default.

### Create and view a memory snapshot

- 1 Start a profiling session.

- 2 Interact with your application until you reach a point in the application's state where you want to take a memory snapshot.
- 3 Select the application in the Profile view.
- 4 Click the Take Memory Snapshot button.

The profiler creates a memory snapshot and marks it with a timestamp. The profiler also implicitly triggers garbage collection before the memory snapshot is recorded.

- 5 To view the data in the memory snapshot, double-click the memory snapshot in the Profile view.

The following example shows the Memory Snapshot view:

Class	Package (Filtered)	Instances	Memory
Employee	valueObjects	1000 (48.85%)	244000 (78.76%)
_EmployeeEntityMetadata	valueObjects	1000 (48.85%)	60000 (19.37%)
DataMgtCF		1 (0.05%)	1476 (0.48%)
_DataMgtCF_mx_managers_SystemManager		1 (0.05%)	728 (0.23%)
_mx_managers_CursorManagerStyle__embed_css_Assets_swf_mx_skins_cursor_BusyCurs		1 (0.05%)	424 (0.14%)
_e5e0a8b749c58154626094e667bd3c593a478f9fa8720b4c6f5a9dbdec0d6f39_flash_displ		1 (0.05%)	404 (0.13%)
_5e891232bd6d81b9d6974dfdba52e58bf92fe717751ed14a87b1b6a6e8ec76bc_flash_disp		1 (0.05%)	404 (0.13%)
_12bb6e99c417300f479d05e018f8dc231181d5e4fbd4451087bd39336fac89240_flash_disp		1 (0.05%)	404 (0.13%)
_0dc790747641a559f6e86f8aa121d894dc3470a66614c3114331e5d3a910bc47_flash_disp		1 (0.05%)	404 (0.13%)
RPCDataServiceRequest	RPCDataServiceAdapter.as\$141	1 (0.05%)	260 (0.08%)
Vector.<*>	__AS3__vec	5 (0.24%)	220 (0.07%)
ListCollectionViewCursor	ListCollectionView.as\$85	2 (0.1%)	152 (0.05%)
EmployeeService	services.employeeservice	1 (0.05%)	144 (0.05%)
LineSegment	Path.as\$138	6 (0.29%)	144 (0.05%)
Vector.<int>	__AS3__vec	2 (0.1%)	80 (0.03%)
Vector.<Number>	__AS3__vec	2 (0.1%)	80 (0.03%)

The following table describes the columns in the Memory Snapshot view:

Column	Description
Class	The classes that had instances in memory at the time that you recorded the memory snapshot.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Instances	The number of instances in memory of each class at the time that you recorded the memory snapshot.
Memory	The amount of memory, in bytes, that all instances of each class used at the time that you recorded the memory snapshot.

You typically use a memory snapshot as a starting point to determine which classes to focus on for memory optimizations or to find memory leaks. To do so, create multiple memory snapshots at different points in time and then compare the differences in the Loitering Objects or Allocation Trace views.

You can save memory snapshots to compare an application's state during a different profiling session. For more information, see [“Save and load profiling data”](#) on page 152.

When you double-click a row in the Memory Snapshot view, the profiler displays the Object References view. This view displays the stack traces for the current class's instances. You view the stack traces for the current class's instances in the Object References view. For more information about the Object References view, see [“Object References view”](#) on page 137.

You can also specify which data to display in the Memory Snapshot view by using profiler filters. For more information, see [“Profiler filters”](#) on page 158.



## Object References view

The Object References view displays stack traces for classes that were instantiated in the application.

The Object References view displays information about the selected class's instances in a tree view format. The Object References view also displays paths to the object's back-references leading up to the GC root. If there is more than one instance of a path to an object reference, the path is not repeated.

To open the Object References view, double-click a class name in the Memory Snapshot or Loitering Objects views.

The Object References view displays data in two tables: the Instances table and the Allocation Trace table.

The Instances table lists all objects that hold references to the current object. The number in parentheses after each class name is represented as a node in the tree view. The number denotes the number of paths leading to the GC root from that node.

You cannot view the number of forward references for an object. An object without references is not listed in this table. This would not normally happen because that object should have been garbage collected if it had no references.

The following example shows the Instances table in the Object References view:

Instance	Property	ID	GC R...
en_US\$styles_properties (10 Paths)		53217	
Object (10)	styles	52518	
Object (10)	en_US	52475	
mx.resources:ResourceManagerImpl (10)	localeMap	52474	
productsView:ProductCart	_resourceManager	70291	
Object	target	70297	YES
productsView:ProductCatalogPanel	_resourceManager	72426	
Object	target	72433	YES
productsView:Grip	_resourceManager	72041	
Object	target	72042	YES
productsView:ProductDetails	_resourceManager	73421	
Object	target	73427	YES
productsView:ProductCatalogThumbnail	_resourceManager	559794	
Object	target	559795	YES
SupportView	_resourceManager	78577	
Object	target	78582	YES
mx.skins.spark:PanelBorderSkin	_resourceManager	570590	
Object	target	570591	YES
productsView:ProductSupport	_resourceManager	74914	
Object	target	74921	YES
productsView:CatalogTitleButtons	_resourceManager	77569	
Object	target	77577	YES
productsView:ProductListItem	_resourceManager	407100	
Object	target	407101	YES

The following table describes the columns in the Instances table:

Column	Description
Instance	The class of the object that holds a reference to the specified object. Expand an instance of a class to view the paths to the object. The number of paths displayed is configurable from the Filters option in the Memory snapshot view.
Property	The property of the object that holds a reference to the specified object. For example, if you have object <code>o</code> with a property <code>i</code> , and assign that property to point to your button's label:  <pre>o.i = myButton.label;</pre> That creates a reference to <code>myButton.label</code> from property <code>i</code> .
ID	The reference ID of the object that holds the reference to the selected object.

Column	Description
GC Root	Indicates whether an object has a back-reference to GC Root. Expand an object node in the tree view to view whether there is a back-reference to GC Root.
Shallow memory	Specifies the size of the object.
Retained memory	Specifies the size of the object and the size of all the other objects that it references.

The properties of an object are displayed in the Object Properties view. Double-click a property to see the property-hierarchy of that object.

The Allocation Trace table shows the stack trace for the selected instance in the Instances table. When you select an instance in the Instances table, the profiler displays the call stack for that instance in the Allocation Trace table.

The following example shows the Allocation Trace table in the Object References view:

Method	Location	Line
BasicChartEvent:BasicChartEvent_PlotSeries2_c()	BasicChartEvent.mxml	41
<anonymous>()	BasicChartEvent.mxml	
Function:http://adobe.com/AS3/2006/builtin:call()		
mx.core:ComponentDescriptor:getProperties()	ComponentDescriptor.as	239
mx.core:Container:createComponentFromDescriptor()	Container.as	3592
mx.core:Container:createComponentsFromDescriptors()	Container.as	3528
mx.containers:Panel:createComponentsFromDescriptors()	Panel.as	1510
mx.core:Container:createChildren()	Container.as	2630
mx.containers:Panel:createChildren()	Panel.as	1050
mx.core:UIComponent:initialize()	UIComponent.as	5172
mx.core:Container:initialize()	Container.as	2567
mx.core:UIComponent:http://www.adobe.com/2006/flex/mx/internal:child	UIComponent.as	5069
mx.core:Container:http://www.adobe.com/2006/flex/mx/internal:childAdd	Container.as	3340
mx.core:Container:addChildAt()	Container.as	2258
mx.core:Container:addChild()	Container.as	2188
mx.core:Container:createComponentFromDescriptor()	Container.as	3716
mx.core:Container:createComponentsFromDescriptors()	Container.as	3528
mx.core:Container:createChildren()	Container.as	2630
mx.core:UIComponent:initialize()	UIComponent.as	5172
mx.core:Container:initialize()	Container.as	2567
mx.core:Application:initialize()	Application.as	841
BasicChartEvent:initialize()	BasicChartEvent.mxml	
mx.managers:SystemManager:http://www.adobe.com/2006/flex/mx/inter	SystemManager.as	1634
mx.managers:SystemManager:initializeTopLevelWindow()	SystemManager.as	2505
mx.managers:SystemManager:doFrameHandler()	SystemManager.as	2356
[execute-queued]()		

The following table describes the columns in the Allocation Trace table:

Column	Description
Method	The top-level method in this table is the method that created the instance of the class that is listed in the Instances table.  You can expand the method to show the stack trace of the method. This can help you determine where the call stack began.
Location	The file where the method is defined.
Line	The line number in the file.

You can only view data in this table when you enable allocation traces when you start the profiler.

You can open the source code of the selected class by double-clicking a class in this table.

## Allocation Trace view

The Allocation Trace view shows which methods were called between two memory snapshots and how much memory was consumed during those method calls. To open the Allocation Trace view, you select two memory snapshots, and then click the View Allocation Trace button. For information on recording a memory snapshot, see [“Memory Snapshot view”](#) on page 135.

The result of the memory snapshot comparison is a list of methods that Flash Player executed between the two memory snapshots. For each of these methods, the profiler reports the number of objects created in that method.

You can use this information to optimize performance. After you identify methods that create an excessive number of objects, you can optimize those hotspots.

To use the Allocation Trace view, enable allocation traces when you start the profiler. The default is disabled.

The following example shows the Allocation Trace view:

Method	Package (Filtered)	Cumulative Instances	Self Instances	Cumulative Memory	Self Memory
[newclass]		49 (1.85%)	44 (7.86%)	65244 (19.5%)	65028 (84.44%)
global.flash.utils.describeType		478 (18.01%)	478 (85.36%)	5844 (1.75%)	5844 (7.59%)
_DataMgtCF_mx_managers_SystemManager.create		9 (0.34%)	3 (0.54%)	3868 (1.16%)	3708 (4.81%)
global.flash.utils.setInterval		21 (0.79%)	7 (1.25%)	1596 (0.48%)	1148 (1.49%)
global.flash.utils.getQualifiedClassName		7 (0.26%)	7 (1.25%)	422 (0.13%)	422 (0.55%)
DataMgtCF.button1_clickHandler		108 (4.07%)	3 (0.54%)	20989 (6.27%)	248 (0.32%)
SetIntervalTimer.SetIntervalTimer		14 (0.53%)	7 (1.25%)	448 (0.13%)	224 (0.29%)
_Super_Employee.valueObjects._Super_Employee	valueObjects	5 (0.19%)	5 (0.89%)	204 (0.06%)	204 (0.26%)
[enterFrameEvent]		247 (9.31%)	3 (0.54%)	15631 (4.67%)	96 (0.12%)
[mouseEvent]		1032 (38.88%)	2 (0.36%)	79793 (23.85%)	64 (0.08%)
<anonymous>		7 (0.26%)	1 (0.18%)	21265 (6.36%)	28 (0.04%)
_Super_Employee.set last_name	valueObjects	76 (2.86%)	0 (0.0%)	4021 (1.2%)	0 (0.0%)
_Super_Employee.get _model	valueObjects	4 (0.15%)	0 (0.0%)	1007 (0.3%)	0 (0.0%)
Employee.valueObjects:Employee	valueObjects	5 (0.19%)	0 (0.0%)	204 (0.06%)	0 (0.0%)
global\$init.global\$init		49 (1.85%)	0 (0.0%)	65244 (19.5%)	0 (0.0%)
[textEvent]		4 (0.15%)	0 (0.0%)	144 (0.04%)	0 (0.0%)
[keyboardEvent]		4 (0.15%)	0 (0.0%)	2792 (0.83%)	0 (0.0%)

The following table describes the columns in the Allocation Trace view:

Column	Description
Method	The method that was called during the snapshot interval. This column also contains the class whose instance called this method.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Cumulative Instances	The number of objects instantiated in this method and all methods called from this method.
Self Instances	The number of objects instantiated in this method. This does not include objects that were instantiated in subsequent method calls from this method.
Cumulative Memory	The amount of memory, in bytes, used by the objects instantiated in this method and all methods called from this method.
Self Memory	The amount of memory, in bytes, used by the objects instantiated in this method. This does not include the memory used by objects that were instantiated in subsequent method calls from this method.

When recording methods during sampling intervals, the profiler also records internal Flash Player actions. These actions show up in the method list in brackets and appear as [mouseEvent] or [newclass] or with similar names. For more information about internal Flash Player actions, see [“How the profiler works”](#) on page 147.

To open the Object Statistics view, click a row in the Allocation Trace table. This view provides details about the objects that were created in the method that you selected. It also lets you drill down into the objects that were created in methods that were called from this method. For more information, see “[Object Statistics view](#)” on page 140.

You limit the data in the Allocation Trace view by using the profiler filters. For more information, see “[Profiler filters](#)” on page 158.

## Object Statistics view

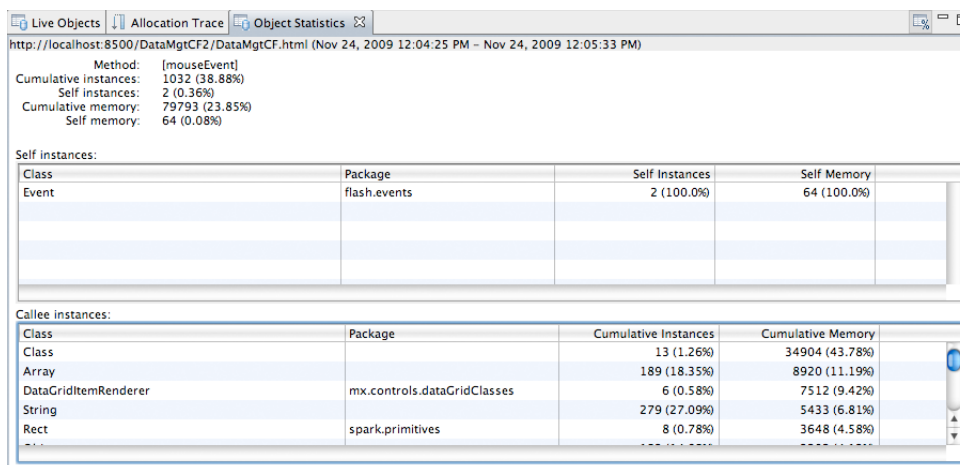
The Object Statistics view shows the performance statistics of the selected group of objects. This view helps you identify which methods call a disproportionate number of other methods. It also shows you how much memory the objects instantiated in those method calls consume. You use the Object Statistics view to identify potential memory leaks and other sources of performance problems in your application.

To access the Object Statistics view, you select two memory snapshots in the Profile view and view the comparison in the Allocation Trace view. Then you double-click a row to view the details in the Object Statistics view.

There are three sections in the view:

- A summary of the selected object’s statistics, including the number of instances and amount of memory used.
- Self Instances table: A list of objects that were instantiated in the method that you selected in the Allocation Trace view. This does not include objects that were instantiated in subsequent method calls from this method. The number of objects in this list matches the number of objects specified in the Self Instances column in the Allocation Trace view.
- Callee Instances table: A list of objects that were instantiated in methods that were called by the method that you selected in the Allocation Trace view. This does not include objects that were instantiated in the method itself. The number of objects in this list matches the number of objects specified in the Cumulative Instances column in the Allocation Trace view.

The following example shows the method summary and the Self Instances and Callee Instances tables of the Object Statistics view:



Method: MouseEvent  
Cumulative instances: 1032 (38.88%)  
Self instances: 2 (0.36%)  
Cumulative memory: 79793 (23.85%)  
Self memory: 64 (0.08%)

Self instances:

Class	Package	Self Instances	Self Memory
Event	flash.events	2 (100.0%)	64 (100.0%)

Callee instances:

Class	Package	Cumulative Instances	Cumulative Memory
Class		13 (1.26%)	34904 (43.78%)
Array		189 (18.35%)	8920 (11.19%)
DataGridItemRenderer	mx.controls.dataGridClasses	6 (0.58%)	7512 (9.42%)
String		279 (27.09%)	5433 (6.81%)
Rect	spark.primitives	8 (0.78%)	3648 (4.58%)

The following table describes the fields in the Self Instances table in the Object Statistics view:

Column	Description
Class	The classes that were instantiated only in the selected method. This does not include classes that were instantiated in subsequent calls from this method.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Self Instances	The number of instances of this class that were created only in the selected method. This does not include instances that were created in subsequent calls from this method.
Self Memory	The amount of memory, in bytes, that is used by instances that were created only in the selected method. This does not include the memory used by instances that were created in subsequent calls from this method.

The following example shows the Callee Instances table of the Object Statistics view:

Callee instances:

Class	Package	Cumulative Instances	Cumulative Memory
Class		13 (1.26%)	34904 (43.78%)
Array		189 (18.35%)	8920 (11.19%)
DataGridItemRenderer	mx.controls.dataGridClasses	6 (0.58%)	7512 (9.42%)
String		279 (27.09%)	5433 (6.81%)
Rect	spark.primitives	8 (0.78%)	3648 (4.58%)
Object		153 (14.85%)	3292 (4.13%)
Point	flash.geom	130 (12.62%)	3120 (3.91%)
SpriteAsset	mx.core	4 (0.39%)	1952 (2.45%)
DataGridListData	mx.controls.dataGridClasses	48 (4.66%)	1536 (1.93%)
Function		41 (3.98%)	1312 (1.65%)
SolidColor	mx.graphics	8 (0.78%)	1056 (1.32%)
EdgeMetrics	mx.core	23 (2.23%)	920 (1.15%)
Dictionary	flash.utils	13 (1.26%)	752 (0.94%)
TextFormat	flash.text	6 (0.58%)	600 (0.75%)
ListEvent	mx.events	12 (1.17%)	576 (0.72%)
PropertyChangeEvent	mx.events	10 (0.97%)	520 (0.65%)
Matrix	flash.geom	9 (0.87%)	504 (0.63%)

The following table describes the fields in the Callee Instances table of the Object Statistics view:

Column	Description
Class	The classes that were instantiated in the selected method. This includes classes that were instantiated in subsequent calls from this method.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Cumulative Instances	The number of instances of this class that were created in the selected method and in subsequent calls from this method.
Cumulative Memory	The amount of memory, in bytes, that is used by instances that were created in the selected method and in subsequent calls from this method.

## Performance Profile view

The Performance Profile view is the primary view to use when doing performance profiling. It shows statistics such as number of calls, self-time, and cumulative time for the methods that are called during a particular sampling interval. You use this data to identify performance bottlenecks.

The process of performance profiling stores a list of methods and information about those methods that were called between the time you clear the performance data and the time you capture new data. This time difference is known as the *interaction period*.

To use the Performance Profile view, enable performance profiling when you start the profiler. This is the default setting.

### Generate a performance profile

- 1 Start a profiling session with performance profiling enabled.
- 2 Interact with your application until you reach the point where you want to start profiling.
- 3 Click the Reset Performance Data button.
- 4 Interact with your application and perform the actions to profile.
- 5 Click the Capture Performance Profile button.

The time difference between when you clicked Reset Performance Data and the time you clicked Capture Performance Profile is the interaction period. If you do not click the Reset Performance Data button at all, then the performance profile includes all data captured from the time the application first started.

- 6 Double-click the performance profile in the Profile view.

The following example shows the Performance Profile view:

Method	Package (Filtered)	Calls	Cumulative Time (ms)	Self Time (ms)	Avg. Cumulative Time (ms)	Avg. Self Time (ms)
[tincan]		1 (0.01%)	4203 (61.77%)	4203 (61.77%)	4203.0	4203.0
[mouseEvent]		1 (0.01%)	1387 (20.39%)	1384 (20.34%)	1387.0	1384.0
NetConnectionMessageResponder.resultHandler		0 (0.0%)	952 (13.99%)	0 (0.0%)	0.0	0.0
[reap]		1 (0.01%)	173 (2.54%)	173 (2.54%)	173.0	173.0
[enterFrameEvent]		1 (0.01%)	138 (2.03%)	1 (0.01%)	138.0	1.0
TypeUtility.convertResultHandler	com.adobe.serializers.utility	0 (0.0%)	129 (1.9%)	0 (0.0%)	0.0	0.0
TypeUtility.convertListToStrongType	com.adobe.serializers.utility	0 (0.0%)	128 (1.88%)	1 (0.01%)	0.0	0.0
[mark]		1 (0.01%)	108 (1.59%)	104 (1.53%)	108.0	104.0
TypeUtility.convertToStrongType	com.adobe.serializers.utility	0 (0.0%)	88 (1.29%)	7 (0.1%)	0.0	0.0
_Super_Employee.set uid	valueObjects	0 (0.0%)	56 (0.82%)	1 (0.01%)	0.0	0.0
[verify]		1 (0.01%)	46 (0.68%)	33 (0.49%)	46.0	33.0
TypeUtility.assignProperty	com.adobe.serializers.utility	0 (0.0%)	44 (0.65%)	0 (0.0%)	0.0	0.0
[generate]		1 (0.01%)	33 (0.49%)	33 (0.49%)	33.0	33.0
_Super_Employee.valueObjects._Super_Employee	valueObjects	0 (0.0%)	32 (0.47%)	5 (0.07%)	0.0	0.0

The following table describes the columns in the Performance Profile view:

Column	Description
Method	The name of the method and the class to which the method belongs.  Internal actions executed by Flash Player appear as entries in brackets; for example, [mark] and [sweep]. You cannot change the behavior of these internal actions, but you can use the information about them to aid your profiling and optimization efforts. For more information on these actions, see <a href="#">“How the profiler works”</a> on page 147.
Package	The package that the class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Calls	The number of times the method was called during the interaction period. If one method is called a disproportionately large number of times compared to other methods, then you can look to optimizing that method so that the execution time is reduced.
Cumulative Time	The total amount of time, in milliseconds, that all calls to this method, and all calls to subsequent methods, took to execute during the interaction period.

Column	Description
Self Time	The amount of time, in milliseconds, that all calls to this method took to execute during the interaction period.
Avg. Cumulative Time	The average amount of time, in milliseconds, that all calls to this method, and calls to subsequent methods, took to execute during the interaction period.
Avg. Self Time	The average amount of time, in milliseconds, that this method took to execute during the profiling period.

If you double-click a method in the Performance Profile view, Flex displays information about that method in the Method Statistics view. This lets you drill down into the call stack of a particular method. For more information, see [“Identify method performance characteristics”](#) on page 143.

You limit the data in the Performance Profile view by using the profiler filters. For more information, see [“Profiler filters”](#) on page 158.

You can save performance profiles for later use. For more information, see [“Save and load profiling data”](#) on page 152.

## Identify method performance characteristics

The Method Statistics view shows the performance characteristics of the selected method. You typically use the Method Statistics view to identify performance bottlenecks in your application. By viewing, for example, the execution times of a method, you can see which methods take a disproportionate amount of time to run. Then you can selectively optimize those methods.

For more information, see [“Performance Profile view”](#) on page 141.

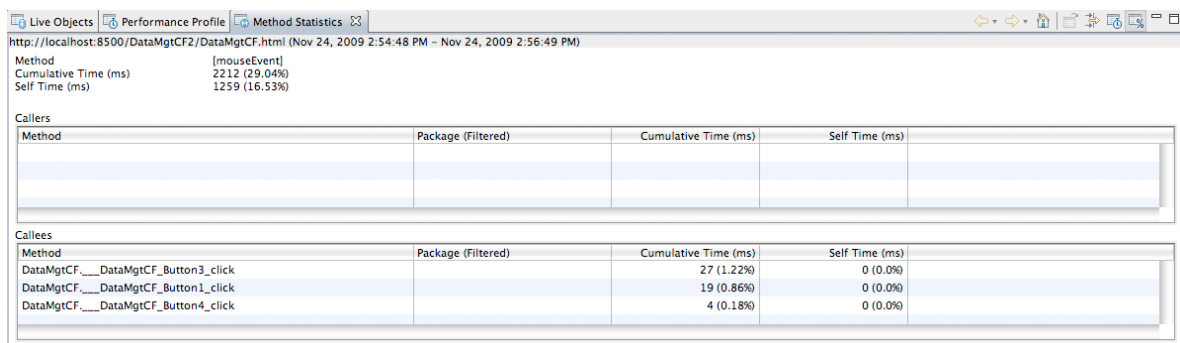
### View method details in the Method Statistics view

- 1 Double-click a row in the Performance Profile view or select a method in the Performance Profile view.
- 2 Click the Open Method Statistics button.

There are three sections in the view:

- A summary of the selected method’s performance, including the number of calls, cumulative time, and self-time.
- Callers table: Details about the methods that called the selected method. In some situations, it is important to know if the selected method is being called excessively, how it is being used, and whether it is being used correctly.
- Calleees table: Details about the methods that were called by the selected method.

The following example shows the method summary and the Callers and Calleees tables of the Method Statistics view:



http://localhost:8500/DataMgtCF2/DataMgtCF.html (Nov 24, 2009 2:54:48 PM – Nov 24, 2009 2:56:49 PM)			
Method	[mouseEvent]		
Cumulative Time (ms)	2212 (29.04%)		
Self Time (ms)	1259 (16.53%)		
Callers			
Method	Package (Filtered)	Cumulative Time (ms)	Self Time (ms)
Calleees			
Method	Package (Filtered)	Cumulative Time (ms)	Self Time (ms)
DataMgtCF___DataMgtCF_Button3_click		27 (1.22%)	0 (0.0%)
DataMgtCF___DataMgtCF_Button1_click		19 (0.86%)	0 (0.0%)
DataMgtCF___DataMgtCF_Button4_click		4 (0.18%)	0 (0.0%)

The following table describes the fields in the Callers table of the Method Statistics view:

Column	Description
Method	The methods that called the method that appears in the summary at the top of this view. If this list is empty, the target method was not called by any methods that are not filtered out.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Cumulative Time	The amount of time, in milliseconds, that each calling method, and all subsequent methods, spent executing.
Self Time	The amount of time, in milliseconds, that each calling method spent executing. This does not include methods called by subsequent methods.

The following table describes the fields in the Callees table of the Method Statistics view:

Column	Description
Method	The methods that were called by the method that is shown in the summary at the top of this view. If this list is empty, then the target method was not called by any methods that are not filtered out.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that the class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the class is in the global package or the unnamed package.
Cumulative Time	The amount of time, in milliseconds, that each called method, and all subsequent methods, spent executing.
Self Time	The amount of time, in milliseconds, that each called method spent executing. This does not include methods called by subsequent methods.

You can navigate the call stack while you attempt to find the performance bottlenecks by clicking the methods in either the Callers or Callees tables. If you double-click a method in these tables, the profiler displays that method's summary at the top of the Method Statistics view and then shows the callers and callees for the newly selected method in the two tables.

**Note:** *The cumulative time minus the self-time in this view will not always equal the cumulative time of the callers. That is because if you drill up to a caller, then the cumulative time will be the self-time of that caller plus all chains from which the original method was called, but not other callees.*

You can also use the Back, Forward, and Home profiler buttons to navigate the call stack.

You can limit the data in the Method Statistics view by using the profiler filters. For more information, see [“Profiler filters”](#) on page 158.

## Loitering Objects view

The Loitering Objects view shows you the differences between two memory snapshots of the application that you are profiling. The differences that this view shows are the number of instances of objects in memory and the amount of memory that those objects use. This is useful in identifying memory leaks. The time between two memory snapshots is known as the *snapshot interval*.

To open the Loitering Objects view, select two memory snapshots and click the Loitering Objects button. For information on recording a memory snapshot, see [“Memory Snapshot view”](#) on page 135.



The following example shows the Loitering Objects view:

Class	Package	Instances	Memory
Array		102 (23.67%)	4748 (4.41%)
String		72 (16.71%)	4932 (4.58%)
Object		62 (14.39%)	2208 (2.05%)
DataGridListData	mx.controls.dataGridClasses	54 (12.53%)	1728 (1.61%)
WeakMethodClosure	flash.events	26 (6.03%)	416 (0.39%)
Class		18 (4.18%)	84081 (78.15%)
Date		16 (3.71%)	512 (0.48%)
EmbeddedFont	mx.core	13 (3.02%)	312 (0.29%)
Function		9 (2.09%)	288 (0.27%)
_EmployeeEntityMetadata	valueObjects	7 (1.62%)	420 (0.39%)
Employee	valueObjects	7 (1.62%)	1288 (1.2%)
ListRowInfo	mx.controls.listClasses	6 (1.39%)	288 (0.27%)
Graphics	flash.display	5 (1.16%)	60 (0.06%)

The following table describes the columns in the Loitering Objects view:

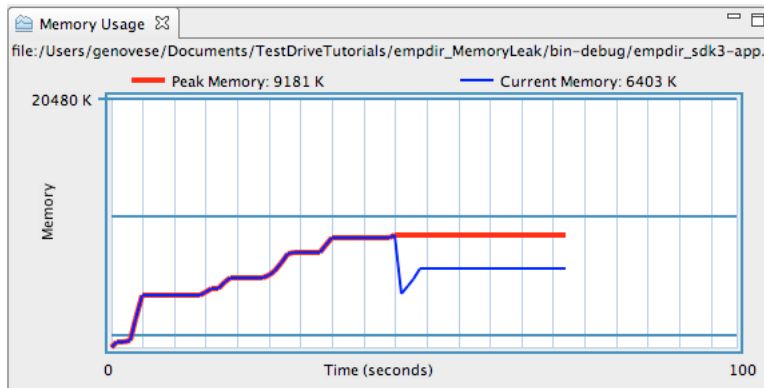
Column	Description
Class	The classes that were created but not destroyed during the snapshot interval.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that this class is in. The number following the dollar sign is a unique ID for that class.  If the Package field is empty, the object is in the global package or the unnamed package.
Instances	The number of instances created during the snapshot interval. This is the difference between the number of instances of each class that existed in the first snapshot and the number of instances of each class in the second snapshot.  For example, if there were 22,567 strings in the first snapshot, and 22,861 strings in the second snapshot, the value of this field would be 294.
Memory	The amount of memory allocated during the snapshot interval. This is the difference between the amount of memory that the instances of each class used at the time the first snapshot was taken and the amount of memory that the instances of each class used at the time the second snapshot was taken.  For example, if Strings took up 2,031,053 bytes in the first snapshot and 2,029,899 bytes in the second snapshot, the value of this field would be 1154 bytes.

For more information on identifying memory leaks, see “[Locate memory leaks](#)” on page 155.

## Memory Usage graph

The Memory Usage graph shows you the memory used by the application that you are profiling. This value is different from the memory usage of Flash Player and its browser. That is because this value does not include the profiler agent or the browser’s memory usage. It consists only of the sum of the profiled application’s live objects. As a result, if you compare the value of memory usage in this graph against the amount of memory the browser uses as shown in, for example, the Windows Task Manager, you get different results.

The following image shows the graph in the Memory Usage view:



The value for Current Memory is the same as the sum of the totals in the Live Objects view's Memory column, assuming that all filters are disabled.

The value for Peak Memory is the highest amount of memory that this application has used during the current profiling session.

The Memory Usage graph shows the application's memory for the last 100 seconds. You cannot configure this number, and you cannot save historical data for the chart.

If you close the Memory Usage graph and want to reopen it, click the drop-down list in the Profile view and select Memory Usage.

## Profiling an application

The profiler helps you identify performance bottlenecks and memory leaks in your applications. You launch it from within Adobe Flash Builder, and as you interact with your application, the profiler records data about the state of the application. The recorded data includes the number of objects, the size of those objects, the number of method calls, and the time spent in those method calls.

Profiling an application can help you understand the following about your application:

**Call frequency** In some cases, computationally expensive methods are called more than once when multiple calls are not necessary. By identifying the most commonly called methods, you can focus your performance-tuning time on a smaller area of the application, where it can have the most impact on performance.

**Method duration** The profiler can tell you how much time was spent in a particular method. Or, if the method is called multiple times, what the average amount of time spent in that method was during a profiling section. If you discover that some methods cause a performance bottleneck, you can try to optimize those methods.

**Call stacks** By tracing the call stack of a method, you can see the entire path that the application takes as it calls successive methods. You then sometimes discover that methods are being called unnecessarily.

**Number of instances (object allocation)** You sometimes discover that the same object is being created many times, when only a specific number of instances are required. In these cases, you can consider implementing a Singleton pattern if you really require only one of those objects. Or you can apply other techniques that reduce excessive object allocation. If the number of objects is large, but necessary, you can consider optimizing the object itself to reduce its aggregate resource and memory usage.

**Object size** If you notice that some objects are disproportionately large, you can try to optimize those objects to reduce their memory footprint. It is especially helpful if you optimize objects that are created many times in the application.

**Garbage collection** When comparing profiling snapshots, you sometimes discover that some objects that the application no longer requires are still "loitering", or are still stored in memory. To avoid these memory leaks, you add logic that removes any remaining references to those objects.

Profiling is an iterative process and a part of each step of application development. To quickly identify problem areas and gain the most benefit by profiling, it is recommended that you profile the application early and as often as possible in the development cycle.

## Types of profiling

Before you use the profiler, decide on what profiling you are going to do: performance profiling or memory profiling.

*Performance profiling* is the process of looking for methods in your application that run slowly and can be improved. Once identified, these hotspots can be optimized to speed up execution times so that your application runs faster and responds more quickly to user interaction. You generally look for two things when doing performance profiling: a method that is called only once but takes more time to run than similar methods, or a method that may not take much time to run but is called many times. You use the performance profiling data to identify the methods that you then optimize. You might find that reducing the number of calls to a method is more effective than refactoring the code within the method.

*Memory profiling* is the process of examining how much memory each object or type of object is using in the application. You use the memory profiling data in several ways: to see if there are objects that are larger than necessary, to see if there are too many objects of a single type, and to identify objects that are not garbage collected (memory leaks). By using the memory profiling data, you can try to reduce the size of objects, reduce the number of objects that are created, or allow objects to be garbage collected by removing references to them.

Memory profiling can slow performance of your application because it uses much more memory than performance profiling. Do memory profiling only when necessary.

You often do both performance profiling and memory profiling to locate the source of performance problems. You use performance profiling to identify the methods that result in excessive memory allocation and long execution times. Then, you use memory profiling to identify the memory leaks in those methods.

When you know what profiling you are going to do, you can start the profiler.

## Additional resources

The profiler alone does not improve the size, speed, and perceived performance of your application. After you use the profiler to identify the problem methods and classes, see the following resources in the Flex documentation for help in improving your application:

- [Optimizing applications](#)
- [Improving startup performance](#)

## How the profiler works

The profiler is an agent that communicates with the application that is running in Flash Player. It connects to your application with a local socket connection. As a result, you might have to disable anti-virus software to use it if your antivirus software prevents socket communication.

When the profiler is running, it takes a snapshot of data at short intervals, and records what Adobe Flash Player is doing at the time. This is called *sampling*. For example, if your application is executing a method at the time of the snapshot, the profiler records the method. If, by the next snapshot, the application is still executing that same method, the profiler continues to record the time. When the profiler takes the next snapshot, and the application has moved on to the next operation, the profiler can report the amount of time it took for the method to execute.

Sampling lets you profile without noticeably slowing down the application. The interval is called the *sampling rate*, and it occurs every 1 ms or so during the profiling period. This means that not every operation is recorded and that not every snapshot is accurate to fractions of a millisecond. But it does give you a much clearer idea of what operations take longer than others.

By parsing the data from sampling, the profiler can show every operation in your application, and the profiler records the execution time of those operations. The profiler also records memory usage and stack traces and displays the data in a series of views, or panels. Method calls are organized by execution time and number of calls, as well as number of objects created in the method.

The profiler also computes cumulative values of data for you. For example, if you are viewing method statistics, the cumulative data includes the time and memory allocated during that method, plus the time and memory allocated during all methods that were called from that method. You can drill down into subsequent method calls until you find the source of performance problems.

## Profiling APIs

The profiler uses the ActionScript APIs defined in the `flash.sampler.*` package. This package includes the `Sample`, `StackFrame`, `NewObjectSample`, and `DeleteObjectSample` classes. You can use the methods and classes in this package to write your own profiler application or to include a subset of profiling functionality in your applications.

In addition to the classes in the `flash.sampler.*` package, the profiler also uses methods of the `System` class.

## Internal player actions

Typically, the profiler records data about methods of a particular class that Flash Player was executing during the sampling snapshot. However, sometimes the profiler also records internal player actions. These actions are denoted with brackets and include `[keyboardEvent]`, `[mark]`, and `[sweep]`.

For example, if `[keyboardEvent]` is in the method list with a value of 100, you know that the player was doing some internal action related to that event at least 100 times during your interaction period.

The following table describes the internal Flash Player actions that appear in profiling data:

Action	Description
[generate]	The just-in-time (JIT) compiler generates AS3 machine code.
[mark]	Flash Player marks live objects for garbage collection.
[newclass]	Flash Player is defining a class. Usually, this occurs at startup but a new class can be loaded at any time.
[pre-render]	Flash Player prepares to render objects (including the geometry calculations and display list traversal that happens before rendering).
[reap]	Flash Player reclaims DRC (deferred reference counting) objects.
[render]	Flash Player renders objects in the display list (pixel by pixel).

Action	Description
[sweep]	Flash Player reclaims memory of unmarked objects.
[verify]	The JIT compiler performs ActionScript 3.0 bytecode verification.
[event_typeEvent]	Flash Player dispatches the specified event.

You can use this information to help you identify performance issues. For example, if you see many entries for `[mark]` and `[sweep]`, you can assume that there are many objects being created and then marked for garbage collection. By comparing these numbers across different performance profiles, you can see whether changes that you make are effective.

To view data about these internal actions, you view a performance profile in the Performance Profile view or a memory profile in the Allocation Trace view. For more information, see [“Performance Profile view”](#) on page 141 and [“Allocation Trace view”](#) on page 139.





## Use the profiler







The profiler requires Flash Player version 9.0.124 or later. You can profile applications that were compiled for Flex 2, Flex 2.0.1, and Flex 3. You can use the profiler to profile ActionScript 3.0 applications that were built with Flash Authoring, as well as desktop applications that run on Adobe® AIR®.

The profiler requires debugging information in the application that you are profiling. When you compile an application and launch the profiler, Flash Builder includes the debugging information in the application by default. You can explicitly include debugging information in an application by setting the `debug` compiler option to `true`. If you export an application by using the Export Release Build option, the application does not contain debugging information in it.







### Profiler toolbar

The following table describes the buttons in the profiler toolbar:

Button	Name	Description
	Resume	Resumes the profiling session. This option is enabled only when an application name is selected and is currently suspended.
	Suspend	Suspends the profiling session. This option is enabled only when an application name is selected and is currently running.
	Terminate	Terminates the profiling session. This option is enabled only when an application name is selected and it has not been already terminated.
	Run Garbage Collector	Instructs Flash Player to run garbage collection. This option is enabled only when an application name is selected and the application is currently running.  For more information about garbage collection, see <a href="#">“Garbage collection”</a> on page 154.

Button	Name	Description
	Take Memory Snapshot	<p>Stores the memory usage of an application so that you can examine it or compare it to other snapshots.</p> <p>This option is enabled only when an application name is selected and that application is currently running and when you select Enable Memory Profiling in the launch dialog box. The profiler adds new memory snapshots as children of the selected application in the Profile view.</p> <p>To open the new memory snapshot in the Memory Snapshot view, double-click the memory snapshot entry.</p> <p>Garbage collection occurs implicitly before memory snapshots are recorded. In other words, clicking the Take Memory Snapshot button is the equivalent of clicking the Run Garbage Collection button and then clicking the Take Memory Snapshot button.</p> <p>For more information about memory snapshots, see <a href="#">“Memory Snapshot view”</a> on page 135.</p>
	Find Loitering Objects	<p>Compares two memory snapshots in the Loitering Objects view.</p> <p>This option is enabled only when two memory snapshots are selected and when you selected Enable Memory Profiling in the launch dialog box.</p> <p>For more information about the Loitering Objects view, see <a href="#">“Loitering Objects view”</a> on page 144.</p>
	View Allocation Trace	<p>Compares the methods between two memory snapshots in the Allocation Trace view.</p> <p>This option is enabled only when two memory snapshots are selected, and when you select Enable Memory Profiling in the launch dialog box.</p> <p>For more information about the Allocation Trace view, see <a href="#">“Allocation Trace view”</a> on page 139.</p>
	Reset Performance Data	<p>Clears the performance profiling data.</p> <p>This option is enabled only when an application name is selected and the application is running and when you select Enable Performance Profiling in the launch dialog box.</p> <p>You typically click this button, interact with your application, and then click the Capture Performance Profile button to get a performance snapshot of the application from the time you reset the data.</p> <p>For more information about the Performance Profile view, see <a href="#">“Performance Profile view”</a> on page 141.</p>
	Capture Performance Profile	<p>Stores a new performance snapshot as a child of the selected application.</p> <p>This option is enabled only when an application name is selected and the application is running and when you select Enable Performance Profiling in the launch dialog box.</p> <p>To open the Performance Profile view, double-click the performance snapshot entry.</p> <p>For more information about the Performance Profile view, see <a href="#">“Performance Profile view”</a> on page 141.</p>
	Delete	<p>Removes the selected snapshot’s data from memory. Clicking this button also removes the application from the profile view, if the application has been terminated.</p> <p>This option is enabled only when a performance or memory snapshot is selected.</p>
n/a	Save	<p>Saves profiling data to disk.</p> <p>The Save option is available from the Profiler view menu. This option is enabled only when an application name is selected.</p>

Some views in the profiler, such as Method Statistics and Object Statistics, have navigation buttons that you use to traverse the stack or change the view. The following table describes the navigation buttons in these profiler views:

Button	Name	Description
	Back	Shows all the methods that you traversed from the first selected method to the currently displaying method.
	Forward	Shows the currently displayed method and the methods that lead to the currently selected method. This item is enabled after you move backward.
	Home	Displays the first selected method.
	Open Source File	Opens a source editor that shows the source code of the selected method.
	Filters	Lets you control which methods you want to include in the table. For more information, see <a href="#">“Profiler filters”</a> on page 158.
	Show/Hide Zero Time Methods	Shows or hides methods that have a time of 0.00 in the average time column, which is a result of not showing up in any samples.

## Start, stop, and resume the profiler

You can profile applications that you are currently developing in Flash Builder. Flash Builder includes debugging information when it compiles and runs an application during a profiling session. You can also profile external applications that you are not currently developing in Flash Builder but whose SWF file is available with a URL or on the file system. To profile an application, the application’s SWF file must include the debugging information. For more information, see [“Profile external applications”](#) on page 153.

### Start profiling an application

- 1 Close all instances of your browser.
- 2 Open your application in Flash Builder.
- 3 Click the *Profile application\_name* button in the main toolbar. Flash Builder prompts you to close all instances of your browsers, if you have not already done so.
- 4 Click OK. Flash Builder compiles the application and launches it in a separate browser window. Flash Builder also displays the Configure Profiler dialog box.
- 5 Select the options in the Configure Profiler dialog box and click Resume. To profile an application, select the Enable Memory Profiling option or the Enable Performance Profiling option. You can select both options when profiling an application.

The following table describes the options:

Setting	Description
Connected From	Shows you the server that you are launching the application from. If the application is running on the same computer as the profiler, this value is localhost. You cannot change this value. However, you can profile an application that is running on a separate computer. See <a href="#">“Profile external applications”</a> on page 153.
Application	Shows you which application you are about to profile. You cannot change this value.
Enable Memory Profiling	Instructs the profiler to collect memory data. Use this option to detect memory leaks or find excessive object creation.  If you are doing performance profiling, you can deselect this option.  By default, Enable Memory Profiling is selected.

Setting	Description
Watch Live Memory Data	<p>Instructs the profiler to display memory data in the Live Objects view while profiling. This is not required for doing either memory or performance profiling. You can select this option only if you selected Enable Memory Profiling.</p> <p>By default, Watch Live Memory Data is selected.</p>
Generate Object Allocation Stack Traces	<p>Instructs the profiler to capture a stack trace each time a new object is created. Enabling this option slows down the profiling experience. Typically, you select this option only when necessary. This option is only available when you select Enable Memory Profiling.</p> <p>By default, Generate Object Allocation Stack Traces is not selected.</p> <p>If you do not select this option, you cannot view allocation trace information in the Object References view or in the Allocation Trace view.</p>
Enable Performance Profiling	<p>Instructs the profiler to collect stack trace data at the sampling intervals. Use these samples to determine where your application spends the bulk of the execution time.</p> <p>If you are doing memory profiling, you can deselect this option.</p> <p>By default, Enable Performance Profiling is selected.</p>

You can change the default values of these options by changing the profiling preferences. For more information, see [“Set profiler preferences”](#) on page 153.

6 You can now start interacting with your application and examining the profiler data.

## Pause and resume profiling an application

After you have started the profiler, you can pause and restart applications in the Profile view. You select an application and then select the action you want to perform on that application. The following example shows you the Profile view with multiple applications. One application is currently running while other applications have been terminated.

### Stop profiling an application

- 1 Select the application in the Profile view.
- 2 Click the Terminate button to end the profiling session manually. Doing so does not close the browser or kill the Player process.
- 3 To return to the Flex Development perspective, select Flex Development from the perspective drop-down list. You can also change perspectives by selecting Control+F8 on Windows.

## Save and load profiling data

After you run the profiler, you can save the data to compare a snapshot from the current profiling session with a snapshot you take after you change your code. Doing so helps you determine if you identified the right problem areas and if your changes are improving the performance and memory usage of your application.

When you save profiling data, you save all the application data in that profile. This data includes all performance profiles, memory snapshots, and allocation traces. Flash Builder writes this information to a group of binary files in the location that you specify.

### Save profiling data

- 1 Select the application in the Profile view.
- 2 Open the drop-down list in the Profile view and select Save. The Browser for Folder dialog box appears.



- 3 Choose a location to save the profile data and click OK. Create a different folder for each set of profiling data that you want to save. If you choose the same folder, the new data overwrites the old data.

### Retrieve saved profiling data

- 1 Select the Saved Profiling Data view.
- 2 Click the Open button. The Browser for Folder dialog box appears.
- 3 Navigate to the folder that contains your application's profile data and click OK. Flash Builder displays the available profiling data in the Saved Profiling Data view. You cannot resume the application in this view, but you can view the memory snapshots, performance profile, or other data that you saved.

You cannot delete saved application data from within Flash Builder.

### Delete profiling data

- 1 Select the snapshot from the application in the Profile view.
- 2 Click the Delete button.

## Set profiler preferences

You can set some profiler preferences so that your settings are applied to all profiling sessions. You can use these settings to define the Flash Player/browser that you use to profile the application in, as well as define the default filters and the port number that the application is available on, if the profiled application is running on a server.

### Set Flash Builder preferences for multiple profiling sessions

- ❖ Open the Preferences dialog and select Flash Builder > Profiler.

Select the options under the Profiler menu to navigate to the various options. The following table describes the preferences you can set:

Menu Selection	Description
Profiler	Lets you select the default profiling method. Select the options to enable or disable memory profiling and performance profiling.
Connections	Lets you define the port number that Flash Builder listens to the profiled application on.  The default port number is 9999. You cannot change the port to 7935, because that port is used by the debugger.
Exclusion Filters	Lets you define the default packages that are excluded from the profiler views. For more information on using filters, see " <a href="#">Profiler filters</a> " on page 158.
Inclusion Filters	Lets you defines the default packages that are included in the profiler views. All other packages are excluded. For more information on using filters, see " <a href="#">Profiler filters</a> " on page 158.
Player/Browser	Lets you define the location of the Flash Player executable and browser executable that Flash Builder uses to run your profiled application.

## Profile external applications

In addition to profiling applications that you are developing in Flash Builder, you can profile external applications. External applications can be SWF files located anywhere that is accessible. This includes applications that are located on a remote web server or an application that is on your local file system.

For the SWF file, you can specify either a URL or a file system location. If you specify a URL, Flash Builder launches the application's SWF file within the default browser. The browser must be using the debugger version of Flash Player to successfully profile the application.

If you specify a file system location for the SWF file, Flash Builder opens the application within the debugger version of the stand-alone Flash Player. In general, use a URL to request the file. Running applications in the stand-alone version of Flash Player can produce unexpected results, especially if your application uses remote services or network calls.

### Profile an external application

- 1 Change to the Flash Profiling perspective.
- 2 Select Profile > Profile External Application. The Profile External Application dialog box appears.
- 3 Select the Launch the Selected Application option (the default) and click the New button. The Add an Application dialog box appears.

You can also manually launch the application by selecting the Launch the Application Manually Outside Flash Builder option.

- 4 Enter the location of the SWF file and click OK, or click the Browse button and locate your application on your file system.
- 5 Click the Launch button. If you specified a URL for the location of the application, Flash Builder launches the application within the default browser. If you specified a file system location for the application, Flash Builder opens the application in the stand-alone version of Flash Player.

If you specified a SWF file that was not compiled with debugging information, Flash Builder returns an error. Recompile the application with the `debug compiler` option set to `true` and launch it again.

## Garbage collection

*Garbage collection* is the act of removing objects that are no longer needed from memory. Memory used by instances that no longer have any references to them should be deallocated during this process.

Flash Player performs garbage collection as necessary during an application's life cycle. Unreferencing an object does not trigger garbage collection. So, when you remove all references to an object, the garbage collector does not necessarily deallocate the memory for that object. That object becomes a candidate for garbage collection.

Clicking the Run Garbage Collector button does not guarantee that all objects that are eligible for garbage collection are garbage collected. Garbage collection is typically triggered by the allocation of memory for new resources. When new resources require memory that is not available in the current allocation, the garbage collector runs and frees up memory that has been marked for deallocation. As a result, even if you remove all references to it, it might not be immediately garbage collected, but likely will be garbage collected when other instances are created and used. If an application is idle, you can watch its memory allocation. Even though there can be objects that are marked for collection, an idle application's memory usage typically does not change until you start interacting with it.

Flash Player allocates memory in blocks of many bytes, and not one byte at a time. If part of a block has been marked for garbage collection, but other parts of the block have not been marked, the block is not freed. The garbage collector attempts to combine unused portions of memory blocks into larger blocks that can be freed, but this is not guaranteed to occur in every pass of the garbage collector.

Garbage collection occurs implicitly before memory snapshots are recorded. In other words, clicking the Take Memory Snapshot button is the equivalent of clicking the Run Garbage Collector button and then clicking the Take Memory Snapshot button.

### Run garbage collection while profiling your application

- ❖ Select the application in the Profile view, and click the Run Garbage Collector button.

You analyze the garbage collector's effectiveness by comparing two memory snapshots before and after garbage collection occurs.

## Identify problem areas

You can use various techniques to identify problem areas in your applications by using the profiler.

### Locate memory leaks

One of the most common problems you face in application development is memory leaks. Memory leaks often take the form of objects that were created within a period but not garbage collected.

One way to identify memory leaks is to look at the number of references to an object in the Instances table in the Object References view. You can generally ignore references from document properties and bindings, and look for unexpected or unusual references, especially objects that are not children of the object. For more information, see [“Object References view”](#) on page 137.

You can also examine paths to instances of an object to determine if a path has a back-reference to the garbage collector (GC Root). An instance that was expected to be released, but has references to GC Root, indicates a memory leak. In these cases, modify your application code so references to GC Root are removed. An instance that has no references to GC Root is ready for garbage collection. Flash Player eventually frees this memory.

Another way to locate memory leaks is to compare two memory snapshots in the Loitering Objects view to determine which objects are still in memory after a particular series of events.

Common ways to clean up memory links are to use the `disconnect()`, `clearInterval()`, and `removeEventListener()` methods.

### Find a back-reference to GC Root for an instance

- 1 Create a memory snapshot.

See [“Create and view a memory snapshot”](#) on page 135.

- 2 Specify the number of back-reference paths to find.

From the Memory Snapshot view, specify the maximum number of paths to find, or select Show All Back-Reference Paths.

- 3 Double-click a class in the memory snapshot to open the Object Reference view.

- 4 Expand the listed paths to and examine if there is a back-reference to GC Root.

### Find loitering objects

- 1 Create two memory snapshots.

See [“Create and view a memory snapshot”](#) on page 135.

- 2 Select the two memory snapshots to compare.

**Note:** *If you have more than two memory snapshots, you cannot select a third one. You can compare only two memory snapshots at one time.*

- 3 Click the Find Loitering Objects button.

Loitering Objects view shows you potential memory leaks. The Instances and Memory columns show the differences between the number of instances of a class and the amount of memory consumed by those instances during the interval between one snapshot and the next. If you see a class that you did not expect to be created during that time, or a class that you expected to be destroyed during that time, investigate your application to see if it is the source of a memory leak.

- 4 To determine how an object in the Find Loitering Objects view was instantiated, double-click the object in the view. The Object References view shows you the stack trace for each instance that you selected.

One approach to identifying a memory leak is to first find a discrete set of steps that you can do over and over again with your application, where memory usage continues to grow. It is important to do that set of steps at least once in your application before taking the initial memory snapshot so that any cached objects or other instances are included in that snapshot.

Then you perform that set of steps in your application a particular number of times—3, 7, or some other prime number—and take the second memory snapshot to compare with the initial snapshot. In the Find Loitering Objects view, you might find loitering objects that have a multiple of 3 or 7 instances. Those objects are probably leaked objects. You double-click the classes to see the stack traces for each of the instances.

Another approach is to repeat the sequence of steps over a long period and wait until the memory usage reaches a maximum. If it does not increase after that, there is no memory leak for that set of steps.

Common sources of memory leaks include lingering event listeners. You can use the `removeEventListener()` method to remove event listeners that are no longer used. For more information, see Object creation and destruction in [Using Flex 4.6](#).

## Analyze execution times

By analyzing the execution times of methods and the amount of memory allocated during those method calls, you can determine where performance bottlenecks occur.

This is especially useful if you can identify the execution time of methods that are called many times, rather than methods that are rarely called.

### Determine frequency of method calls

- 1 Start a profiling session and ensure that you enable performance profiling when configuring the profiler on the startup screen.
- 2 Select your application in the Profile view.
- 3 Interact with your application until you reach the point where you want to start analyzing the number of method calls. To see how many times a method was called from when the application started up, do not interact with the application.
- 4 Click the Reset Performance Data button. This clears all performance data so that the next performance profile includes any data from only this point forward.
- 5 Interact with your application until you reach the point where you check the number of method calls since you reset the performance data.
- 6 Click the Capture Performance Profile button.

7 Double-click the performance profile in the Profile view.

8 In the Performance Profile view, sort the data by the Method column and find your method in the list.

The value in the Calls column is the number of times that method was called during this sampling interval. This is the time between when you clicked the Reset Performance Data button and when you clicked the Capture Performance Profile button.

Examine the values in the Cumulative Time, Self Time, Avg. Cumulative Time, and Avg. Self Time columns of the Performance Profile view. These show you the execution time of the methods.

Compare the time each method takes to execute against the time that all the methods that are called by a particular method take to execute. In general, if a method's self-time or average self-time are high, or high compared to other methods, look more closely at how the method is implemented and try to reduce the execution time.

Similarly, if a method's self-time or average self-time are low, but the cumulative time or average cumulative time are high, look at the methods that this method calls to find the bottlenecks.

## Locate excessive object allocation

One way to identify trouble areas in an application is to find out where you might be creating an excessive number of objects. Creating an instance of an object can be an expensive operation, especially if that object is in the display list. Adding an object to the display list can result in many calls to style and layout methods, which can slow down an application. In some cases, you can refactor your code to reduce the number of objects created.

After you determine whether there are objects that are being created unnecessarily, decide whether it is reasonable or worthwhile to reduce the number of instances of that class. For example, you could find out how large the objects are, because larger objects generally have the greatest potential to be optimized.

To find out which objects are being created in large numbers, you compare memory snapshots of the application at two points in time.

### View the number of instances of a specific class

- 1 Start a profiling session and ensure that you enable memory profiling when configuring the profiler on the startup screen.
- 2 Interact with your application until you reach the place to take a memory snapshot.
- 3 Click the Take Memory Snapshot button. The profiler adds a new memory snapshot to the application list in the Profile view.
- 4 Open the memory snapshot by double-clicking it in the Profile view.
- 5 To view the number of instances of a particular class, and how much memory those instances use, sort by the Class column and find your class in that column. You can also sort by the other columns to quickly identify the objects that take up the most memory or the objects with the most instances. In most cases, Strings are the class with the most instances and the most memory usage.

For more information about the Memory Snapshot view, see "[Memory Snapshot view](#)" on page 135.

### Locate instances of excessive object allocation

- 1 Start a profiling session and ensure that you enable memory profiling when configuring the profiler on the startup screen.
- 2 Interact with your application until you reach the first place to take a memory snapshot.
- 3 Click the Take Memory Snapshot button.

The profiler saves the memory snapshot in the Profile view, and marks the snapshot with a timestamp.

4 Interact with your application until you reach the second place to take a memory snapshot.

5 Click the Take Memory Snapshot button again.

The profiler saves the second memory snapshot in the Profile view, and marks the snapshot with a timestamp.

6 Select the two memory snapshots to compare.

***Note:** If you have more than two memory snapshots, you cannot select a third one. You can compare only two at a time.*

7 Click the View Allocation Trace button.

The Allocation Trace view shows which methods were called between the two snapshots and how much memory was consumed during those method calls. See [“Allocation Trace view”](#) on page 139.

## Profiler filters

The amount of data in a profiler view can sometimes be overwhelming and the level of detail can be too great. The internal actions of Flash Player might obscure the data that you are truly interested in, such as your own methods and classes. Also, Flash Player creates and destroys many objects without your direct interaction. Thus, you could see that thousands of strings or arrays are being used in your application.

You can set filters in the following views:

- Live Objects
- Memory Snapshot
- Performance Profile
- Method Statistics
- Allocation Trace

You can define which packages appear in the profiler views by using the profiler filters. There are two types of filters:

**Exclusion filters** The exclusion filters instruct the profiler to exclude from the profiler views packages that match the patterns in its pattern list. If you use charting controls, for example, but do not want to profile them, you can add the `mx.charts.*` pattern to the exclusion filter. You can also exclude global built-in items. These include global classes such as `String` and `Array`.

**Inclusion filters** The inclusion filters instruct the profiler to include in the profiler views only those packages that match the patterns in its pattern list. If you have a custom package named `com.mycompany.*`, for example, you can view details about only classes in this package by adding it to the inclusion filter.

The default exclusions are `flash.*`, `spark.*`, `mx.*`, and the Flex framework classes in the global or unnamed package. These include global classes such as `String` and `Array`. This means that the default inclusions are user-defined classes in the unnamed package and user-defined classes in nonframework packages (such as `com.mycompany.MyClass`).

You can exclude user-defined classes that are in the unnamed package from the profiling data. To do this, add `“*”` to the exclusion list.

**Maximum visible rows** Maximum visible rows sets the number of rows of data that can be displayed in a view. Increase this value if the data you are looking for is not displayed in the view. Decrease this value to improve performance of the profiler. Use other filters to ensure that you are displaying the data of interest to you.

**Maximum back-reference paths to find:** Maximum back-reference paths sets the number of paths to a referenced object to display when examining object references. The paths are displayed according to the shortest path. By default,

the ten shortest paths are displayed. Increase this value to display additional paths or select Show All Back-Reference Paths. Showing additional paths can help you locate objects that are no longer referenced. See [“Locate memory leaks”](#) on page 155.

### Set default filter preferences

- ❖ Open the Preferences dialog and select Flash Builder > Profiler > Inclusion Filters or Exclusion Filters.

When displaying profiler data, the profiler applies the exclusion filters first; then it applies the inclusion filters. For example, suppose you set the exclusion filter to `mx.controls.*`, but set the inclusion filter to `mx.*.*`; the profiler does not show details about any classes in the `mx.controls` package because that package was excluded, even though their pattern matches the inclusion pattern list. Similarly, suppose you set the exclusion filter to `mx.*.*` and the inclusion filter to `mx.controls.*`; the profiler does not show details about any classes in `mx.controls.*` package because they were excluded before it was included.

When you filter out certain data points, the percentage values of columns are adjusted to reflect only the percentage of non-filtered data.

The profiler maintains filters from one profiling session to the next for the same application.

The filter settings are not inherited by subviews. For example, if you apply a filter to the data in the Memory Snapshot view, and then navigate to the Object References view by double-clicking a method, the Object References view does not apply the same filter.

### Determine whether data is being filtered

- 1 Click the Filter button or look at the titles of the data tables. If there are filters applied, the Package column's heading is Package (Filtered).
- 2 (Optional) Reset the filters to the default by clicking the Restore Defaults button.

# Chapter 7: Unit Testing Tools in Flash Builder

## FlexUnit test environment

The FlexUnit test environment allows you to generate and edit repeatable tests that can be run from scripts or directly within Flash Builder. Flash Builder supports both FlexUnit 4 and Flex Unit 1 open source frameworks.

From Flash Builder, you can do the following:

- Create unit test cases and unit test suites

Flash Builder wizards guide you through the creation of test cases and test suites, generating stub code for the tests.

- Run the test cases and test suites

You can run test cases and test suites various ways from within Flash Builder or outside the Flash Builder environment. The results of the tests are displayed in a test application. Flash Builder opens a Flex Unit Results View for analysis of the test run.

- Navigate to source code from the Flex Unit Results View

In the Test Results panel, double-click a test to open the test implementation.

The Test Failure Details panel lists the source and line number of the failure. If the listed source is in the current workspace, double-click the source to go directly to the failure.



To understand the basics of Test Driven Development (TDD) using FlexUnit and Flash Builder, see this [Adobe Dev Center article](#) by Adobe Community Professional, Elad Elrom.

## Create FlexUnit tests

You can create FlexUnit test case classes and test case suites for the following types of projects:

- Flex project
- Flex Mobile project
- Flex Library project
- ActionScript project
- AIR project

When creating a test case class, you can specify the following options:

- A `src` folder in a Flex project for the class
- A package for the class
- The classes to test
- Methods to test for each specified class

A FlexUnit test case suite is a series of tests based on previously created test case classes, specified methods in those classes, and other test case suites.



## Create a FlexUnit test case class

When you create a FlexUnit test case class, Flash Builder generates an ActionScript file for the test case class, which it places in a package for test cases.

The following procedure assumes that you have created a project in Flash Builder in which you want to create and run Flex Unit tests.

- 1 Select the Flex project and then from the context menu, select New > Test Case Class.

If you select an ActionScript class file in project, then that class is automatically selected for the FlexUnit test case in the New Test Case Class wizard.

- 2 In the New Test Case Class wizard, specify whether to create a class in the FlexUnit 4 style or FlexUnit 1 style.
- 3 Specify a name for the test case class.
- 4 (Optional) Specify a source folder and package for the test case class, or accept the defaults.

The default source folder is the `src` folder of the current project. The default package is `flexUnitTests`, which is at the top level of the default package structure for the project.

- 5 (Optional) Enable the Select Class to Test toggle, and browse to a specific class. Click Next.
- 6 (Optional) Select the methods in the selected class that you want to test.
- 7 Click Finish.

Code the test case you created. Use the generated code stubs as a starting point.

## Create a FlexUnit test case suite

This procedure assumes that you have previously created test case classes.

- 1 Select the Flex project and then create a test case suite from the context menus by selecting New > Test Suite Class.
- 2 In the New Test Suite Class wizard, specify whether to create a class in the FlexUnit 4 style or FlexUnit 1 style.
- 3 Provide a name for the test suite.
- 4 Navigate in the test suites and test cases to select the classes and methods to include in the test suite. Click Finish.

## Customize default FlexUnit test case classes and test case suite classes

You can customize the default FlexUnit test case classes and test case suite classes that Flash Builder creates. Flash Builder uses file templates to create the default versions of these files.

The file templates for FlexUnit are available from the Preferences dialog at Flash builder > File Templates > FlexUnit. There are separate templates for FlexUnit1 and FlexUnit4 test case classes and test suite classes.

See “[Customize file templates](#)” on page 57 for information on how to modify the default file templates.

**Note:** *FlexUnitCompilerApplication.mxml and FlexUnitApplication.mxml derive from the template for MXML Web Application or MXML Desktop Application. The template that is used depends on whether the Flex project is configured for a web application (runs in Adobe® Flash® Player) or a desktop application (runs in Adobe AIR®).*

### More Help topics

[Open source language reference for FlexUnit](#)

[Open source documentation for FlexUnit](#)

## Run FlexUnit tests

FlexUnit tests can be run from within Flash Builder or from outside Flash Builder using SWFs generated for the FlexUnit test. In either case, the results of the tests are displayed in the FlexUnit Results View.

You can also configure and save a FlexUnit test before running it.

By default, FlexUnit tests run in the Flash Debug perspective. You can launch tests from the Flash Development or Flash Profile perspectives, but Flash Builder switches to the Flash Debug perspective when running the test.

You can modify the default perspective for FlexUnit tests. Open the Preferences window, and navigate to Flash Builder > FlexUnit.

### FlexUnit compiler application and FlexUnit application

When you create a FlexUnit test case, Flash Builder creates the following FlexUnit compiler application and a FlexUnit application:

- FlexUnitCompilerApplication.mxml
- FlexUnitApplication.mxml

Flash Builder uses these applications when compiling and running FlexUnit tests. Flash Builder places the applications, in the `src` directory of the project.

This application contains references to all FlexUnit test cases and test suites generated by Flash Builder. Flash Builder places all FlexUnit tests in the `<fx:Declarations>` tag of this application. You typically do not edit or modify this file directly.

Refresh the FlexUnit compiler application for the following circumstances:

- You manually add a test case.

If you create a test case class without using the New Test Case wizard, refresh `FlexUnitCompilerApplication.mxml`. Place the new test case in the package with the other test cases.

- You rename a test case
- You delete a test case

Refresh `FlexUnitCompilerApplication.mxml`:

- 1 If the FlexUnit Results view is not open, select **Windows > Show View > Other > Flash Builder > FlexUnit Results**. Click OK.
- 2 In the FlexUnit Results view, click the Refresh button.

### Run a FlexUnit test within Flash Builder

You can run FlexUnit tests within Flash Builder on a project level or for individual test cases.

You typically run FlexUnit tests from the context menu for a project or from the context menus for an individual test case.

However, you can also run FlexUnit tests from the Flash Builder Run menu, the Flash Builder Run button, or from the Execute FlexUnit Test button in the FlexUnit Results view.

If you run tests from the Flash Builder Run menu, a Test Configuration dialog opens, allowing you to select which test classes and methods to run. Test cases for library projects cannot be run using the Flash Builder Run menu.

Flash Builder provides the following keyboard shortcuts to quickly launch FlexUnit tests:

- **Alt+Shift+A, F**  
Runs all FlexUnit tests in the project.
- **Alt+Shift+E, F**  
Runs the selected FlexUnit test.

Run FlexUnit tests from the current selection in the editor. See “[Configure FlexUnit tests](#)” on page 164.

**1** Select a project and run the test:

From the context menu for a project, select **Execute FlexUnit Tests**.

From the Flash Builder Run menu or Run button, select **Run > FlexUnit Tests**.

**2** (Flash Builder Run menu) In the Run FlexUnit Tests configuration dialog, select the test cases and methods to run in the test. Click **OK** to run the tests.

**3** View the test results.

Flash Builder generates a SWF file in the bin-debug folder of the project.

An application opens displaying information about the test and indicates when the test is complete.

The FlexUnit Results View panel opens displaying the results of the test. See “[View results of a FlexUnit test run](#)” on page 164.

Run individual FlexUnit tests:

**1** In the Project Explorer, navigate to the `flexUnitTest` package:

From the context menu for a FlexUnit test file, select **Execute FlexUnit Tests**.

**2** View the test results.

Flash Builder generates a SWF file in the bin-debug folder of the project.

An application opens displaying information about the test and indicates when the test is complete.

The FlexUnit Results View panel opens displaying the results of the test. See “[View results of a FlexUnit test run](#)” on page 164.

## Run a FlexUnit test outside the Flash Builder environment

This procedure assumes that you have previously run a FlexUnit test within Flash Builder and that Flash Builder is running.

**1** Copy the generated SWF file for a test from the bin-debug folder in your project to a folder outside your development environment.

You can copy the automatically generated SWF file or a SWF file from a FlexUnit test that you have previously saved and configured.

**2** Run the copy of the SWF file.

Flash Player opens displaying information about the test and indicates when the test is complete.

The FlexUnit Results View panel opens in Flash Builder displaying the results of the test.

## Configure FlexUnit tests

- 1 Open the Run FlexUnit Tests configuration dialog:

You can open the Run FlexUnit Tests configuration dialog from the Run menu or from the FlexUnit Results view.

- Select a project. From the Flash Builder menu, select Run > Run > Execute FlexUnit Test.
- From the FlexUnit Results view, select the Execute FlexUnit Tests button.

If the FlexUnit Results view is not open, select Window > Show View > Other > Flash Builder > FlexUnit Results.

- 2 In the Test Configuration dialog, select the test cases and methods to save as a test configuration.

**Note:** *The Test Configuration dialog is not available when you run a test from the context menu in the Package Explorer.*

- 3 (Optional) Select Load to import previous configurations saved as XML files.

- 4 Click Save.

Flash Builder writes an XML file and an MXML file in the `.FlexUnitSettings` folder of your project.

You can use the XML file in build scripts to execute the test.

You can generate a SWF file from the MXML file. This SWF file can be used for testing outside the Flash Builder environment. Typically, you copy the generated MXML file to the `src` folder in your project to generate the SWF file.

## View results of a FlexUnit test run

The FlexUnit Results view displays results of a FlexUnit test, detailing any failures. You can navigate through the results, filter the display, write the results to a file, and load the results from a file.

You can also rerun tests, cancel a running test, and clear the results from the view.

If the FlexUnit Results view is not open, select Window > Show View > Other > Flash Builder > FlexUnit Results.

### Test Results Panel

This panel lists all tests within the test run, indicating whether the test passed or failed.

Double-click a test in the list to go to the test in the ActionScript editor.

### Test Failure Details Panel

Select a test in the Test Results panel to display failure details.

Each failure detail lists the source file and method, including the line number of the failure.

If the source file is local to the working space, double-click the listing to go to the failure in the ActionScript editor.

## FlexUnit Results view menu

From the FlexUnit Results view menu, you can do the following:

- Filter the results displayed
  - Hide the Test Failure Details panel.
  - Display only test runs that failed.
- Scroll through the test runs displayed in the Test Results panel.
- Cancel a running test.
- Save the results of a test run or the configuration of a test run.
- Load results previously saved to a file.
- Clear the results from the panel.
- Rerun the current test. You can choose from:
  - Run all tests.
  - Run failures only.
  - Run the selected test.
- Refresh the FlexUnit configuration.

If you modify a test or add or remove tests, click refresh to load the new FlexUnit configuration.
- Configure and run FlexUnit tests.

Use the Execute FlexUnit Tests button to configure and run FlexUnit tests.

## FlexUnit support for mobile projects

You can run a FlexUnit test on a mobile device. Or, if you don't have a mobile device, you can simulate the device on your desktop and run the FlexUnit test.

### Run a FlexUnit on a mobile device

- 1 Select a mobile project and run the test by selecting Run > FlexUnit Tests or select Execute FlexUnit Tests from the context menu.
- 2 Specify a launch method:
  - **On Desktop** Runs the FlexUnit application on your desktop using the AIR Debug Launcher (ADL), according to a device configuration that you select.
  - **On Device** Runs the FlexUnit application on the device

Flash Builder can access the device connected to your computer's USB port or over the network via Wi-Fi. A socket connection and the computer's IP address is used to communicate between Flash Builder and the FlexUnit application running on the device.

**Note:** You can run the FlexUnit test on a device connected to your computer's USB port. However, you can view the test results only if Wi-Fi is enabled on the device.

The default port number that is used to establish a socket connection is 8765. You can change the default port number by going to the Preferences dialog and selecting Flash Builder > FlexUnit.

If the FlexUnit application cannot establish a connection with the device, Flash Builder sometimes displays a dialog requesting the IP address of the computer. Ensure that the device is properly connected to the network, and that the computer running Flash Builder is accessible from that network.

- 3 In the Run FlexUnit Tests configuration dialog, select the test cases and methods to run in the test. Click OK to run the tests.
- 4 View the test results.

Flash Builder generates a SWF file in the bin-debug folder of the project.

An application opens displaying information about the test and indicates when the test is complete.

The FlexUnit Results View panel opens displaying the results of the test. See “[View results of a FlexUnit test run](#)” on page 164.

# Chapter 8: Developing Applications with Flex

## Basic workflow to develop an application with Flex

### 1 (Optional) Create or import a project.

To create a project, in Flash Builder, select File > New and the type of project you want to open. The New Project wizard differs for each type of project.

In some cases, your project accesses server data, such as from a PHP, ColdFusion, or Java application server. Specify information to access the server when creating the project. For more information, see [“Create projects in Flash Builder”](#) on page 81.

You can import projects that have been previously been created or exported from Flash Builder. For more information, see [“Export and import projects”](#) on page 88.

You can also import projects that have been authored in Adobe Flash Catalyst. For more information, see [“Using Flash Builder with Flash Catalyst”](#) on page 233.

### 2 Create or edit the source files for a project.

Use the Flash Builder editors to edit source files for a project. Flash Builder includes Source mode and Design mode. Many developers lay out user interface elements in Design mode and enter code in Source mode.

For more information, see [“Code Development Tools in Flash Builder”](#) on page 44.

### 3 Build the project.

By default, Flash Builder builds a project every time you save a file in the project. When building the project, Flash Builder places the output files in the appropriate location for the project. Flash Builder alerts you of any errors or warnings encountered when building the project.

For more information, see [“Build projects”](#) on page 93.

### 4 Run applications in the project.

Each project, except library projects, can contain application files that you can launch in run, debug, or profile perspectives of Flash Builder. Flash Builder defines a launch perspective that it uses to launch the application. You can modify the default launch perspective, or create additional launch perspectives.

Mobile AIR Projects, require you to configure a launch perspective before launching the application.

For more information, see [“Run and debug applications”](#) on page 110.

### 5 Test the project.

Use the Flash Builder debugging, profiling, and monitoring tools to test and optimize your projects. You can also use the FlexUnit test environment to generate and edit repeatable tests that can be run from scripts or directly within Flash Builder.

For more information, see [“Debugging Tools in Flash Builder”](#) on page 124 and [“Unit Testing Tools in Flash Builder”](#) on page 160.

If your application accesses remote data, use the Network Monitor to examine the data that flows between an application and a data service. For more information, see [“Monitor applications that access data services”](#) on page 208.

Use the Adobe Flex Profiler to identify performance bottleneck and memory leaks in an application. For more information, see [“Profiling Tools in Flash Builder”](#) on page 133.

## 6 Deploy an application.

Use Flash Builder to export a release version of an application, which you can use to distribute and deploy the application. The process for exporting a release version of an application varies, depending on the type of project.

For more information, see [“Export a release version of an application”](#) on page 117.

# Build user interfaces

Although you can define your user interface completely in Source mode, you typically start by adding user interface elements in design mode.

For more information, see [“Work with components visually in MXML Design Mode”](#) on page 20 and Building the user interface.

**Note:** You can also add assets created with Adobe® Flash® Professional to your application. See [“Create and edit Flash components”](#) on page 229.

## Use forms

You can create rich form experiences using the Spark Form. The Spark Form is a highly customizable container that supports multiple form designs. You can also create and edit skins for the form and form items.

For more information about using the Spark Form, Spark FormHeading, and Spark FormItem containers, see [Using Flex 4.6](#).

## Create a Spark Form (Design mode)

- 1 Create an MXML component. For more information, see [“Create custom MXML components”](#) on page 216.
- 2 In Design mode of the editor, add a Spark Form to the component. Select the Form component from the Layout section of the Component panel, and drag-and-drop into the design area.
- 3 Specify the width and height for the form. Select an MXML or Spark skin for the Form Items.

When you select a skin, an `<fx:Style>` block is created. For example, when you select the `StackedFormItemSkin` Spark skin, the following style block is created.

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    s|Form#form1 s|FormItem
    {
        skinClass: ClassReference("spark.skins.spark.StackedFormItemSkin");
    }
    s|Form#form1 s|FormHeading
    {
        skinClass: ClassReference("spark.skins.spark.StackedFormHeadingSkin");
    }
</fx:Style>
```



If you do not specify a skin, the default Skin classes that are present in `spark.skins.spark` package are used. The default Skin classes are used as follows:

- Form uses `FormSkin` class
- `FormItem` component uses `FormItemSkin` class
- `FormHeading` component uses `FormHeadingSkin` class

- 4 Drag-and-drop the `FormHeading` component into the Spark form, if necessary. When you do so, code like the following is generated:

```
</s:Form>
    <s:FormHeading label="Heading"/>
</s:Form>
```

In Design mode, double-click the `FormHeading` component to edit the heading label.

- 5 Drag-and-drop the required form components into the Spark form. For example, a `TextInput` component. Each component that you drop into the Form creates a `FormItem`.

By default, a `Label` string is added to every `FormItem`. For example, if you drag-and-drop, a `TextInput` component, code like the following is generated:

```
<s:Form>
    <s:FormItem label="Label">
        <s:TextInput/>
    </s:FormItem>
</s:Form>
```

To add a `helpContent` string to the `TextInput` component, select `HelpContent` in the Property Inspector. Then, code like the following is generated:

```
<s:Form>
    <s:FormItem label="Label">
        <s:helpContent>
            <s:Label text="Help String"/></s:Label>
        </s:helpContent>
        <s:TextInput/>
    </s:FormItem>
</s:Form>
```

- 6 You can use the Properties view to configure the component properties. For properties that also exist in the corresponding `mx:Form` component, the user experience and functionality are similar.

Use the Style panel in the Properties view to edit the skins. For more information about applying styles to components, see [“Apply styles”](#) on page 177.

You can also create a custom MXML skin based on a template. For more information, see [“Create a custom MXML skin for a form based on a template \(Design mode\)”](#) on page 172.

## Generate and edit skins for Spark Form Items

The examples below show you how to create custom MXML skins for Spark Form Items. Some changes can be made in Design mode of the editor, while others require you to edit the MXML file in Source mode. It is assumed that you are working in a Flex project using the default Spark theme.

### Create a custom MXML skin for form items based on a template (Design mode)

- 1 In Design mode of the editor, select Create Skin from the context menu for the Form Item. The New MXML Skin dialog box opens.

You can also open the New MXML Skin dialog box in other ways. In the Design mode of the editor, select the Form Item, and do one of the following:

- In the Style section of the Properties view, select the icon next to the Skin field, and select Create Skin.
- From the File menu, select New > MXML Skin.
- From the Design menu, select Create Skin.

- 2 Specify the following in the New MXML Skin dialog box.

- Source folder and package for the generated skin.
- The name for the Skin class that you are creating.
- The host component is selected, by default.
- (Recommended) Select Create As Copy Of and do not remove ActionScript styling code. If you are new to skinning, use a copy to get started with creating a Skin class.

(Advanced Users) Do either of the following if you are familiar with creating Skin classes:

Remove ActionScript styling code or do not create a copy of an existing class. If you do not create a copy of an existing class, Flash Builder generates a blank Skin class file with some comments to guide you.

The remaining steps of this procedure assume that you followed the recommended option for generating a Skin class. For more information, see [“Generate and edit skins for Spark components”](#) on page 188.

- 3 Click Finish. Flash Builder generates a Skin class file and opens it in Design mode of the MXML editor. You can edit the skin using the editing tools in the Style section of the Properties view.
- 4 You are prompted to apply the new skin to all the Form Items. When you do so, a `<fx:Style>` block is created specifying the new skin.

For example, if you apply the `SkinnableContainerSkin` skin to all the `FormItem`s, code like the following is generated:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    s|Form#form1 s|FormItem
    {
        skinClass: ClassReference("spark.skins.spark.SkinnableContainerSkin");
    }
</fx:Style>
<s:Form id="form1" x="145" y="81">
    <s:FormHeading label="Heading"/>
    <s:FormItem label="Label">
        <s:helpContent>
            <s:Label text="Help String"></s:Label>
        </s:helpContent>
        <s:TextInput/>
    </s:FormItem>
    <s:FormItem label="Label">
        <s:helpContent>
            <s:Label text="Help String"></s:Label>
        </s:helpContent>
        <s:TextInput/>
    </s:FormItem>
</s:Form>
```

If you, however, choose to apply the skin to only the selected Form Item. Then, it is mandatory that all the Form Items with that skin have the same number of columns.

For example, if you apply the SkinnableContainerSkin to a single FormItem, code like the following is generated:

```
<s:Form x="186" y="127">
    <s:FormHeading label="Heading"/>
    <s:FormItem label="Label" skinClass="spark.skins.spark.SkinnableContainerSkin">
        <s:helpContent>
            <s:Label text="Help String"></s:Label>
        </s:helpContent>
        <s:TextInput/>
    </s:FormItem>
    <s:FormItem label="Label">
        <s:helpContent>
            <s:Label text="Help String"></s:Label>
        </s:helpContent>
        <s:TextInput/>
    </s:FormItem>
</s:Form>
```

### Define multiple skins for different form items in a single form (Design mode)

Before you define multiple Skin classes for Form Items in a Form, here are a couple of points to remember.

- If you have a form with multiple Form Items, it is recommended that you use the same Skin class for all Form Items. However, you can choose to override this recommendation.
- It is mandatory that Form Items in a Form have the same number of columns.

To define a Skin class, do the following:

- 1 In Design mode of the editor, select the Form Item.
- 2 In the Style section of the Properties view, click Skin and select a Skin class. A warning message appears about the recommended usage. Select Apply Only To Selected Items.

3 Repeat this sequence of steps for each Form Item that you want to apply the skin to.

## Generate and edit skins for Spark Forms

The examples below show you how to create custom MXML skins for Spark Forms. It is assumed that you are working in a Flex project using the default Spark theme.

### Create a custom MXML skin for a form based on a template (Design mode)

To create a custom MXML skin for the form, you can open the New MXML Skin dialog box in any of the following ways.

In the Design mode of the editor, select the Form, and do one of the following:

- Select Create Skin from the context menu for the Form.
- In the Style section of the Properties view, select the icon next to the Skin field, and select Create Skin.
- From the File menu, select New > MXML Skin.
- From the Design menu, select Create Skin.

The remaining steps of this procedure are similar to creating a Skin for Form Items. For more information, see [“Create a custom MXML skin for form items based on a template \(Design mode\)”](#) on page 170.

### Define a single skin for all the form items in a form (Design Mode)

To define a single Skin class for all FormItems in a form, define the FormItem skin in the Form. To do so:

- 1 In Design mode of the editor, select the Form.
- 2 In the Common section of the Properties view, select a FormItem skin.

The selected skin is applied to all the FormItems, and an `<fx:Style>` block is created. For example, if you select the `StackedFormItemSkin` for the FormItems in a form, the `<fx:Style>` block is generated as follows:

```
<fx:Style>
@namespace s "library://ns.adobe.com/flex/spark";
s|Form#form1 s|FormItem
{
    skinClass: ClassReference("spark.skins.spark.StackedFormItemSkin");
}
s|Form#form1 s|FormHeading
{
    skinClass: ClassReference("spark.skins.spark.StackedFormHeadingSkin");
}
</fx:Style>
```

## Use charts

You can use Flash Builder to add charting components to display data in your user interface. The Flex Charting components let you create some of the most common chart types, and also give you extensive control over the appearance of your charts. For an overview of the different charts available, see [Chart types](#).

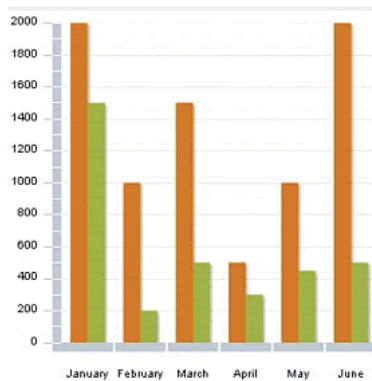
For information on defining chart data, formatting chart elements, and manipulating other aspects of charts, see [Introduction to charts](#).

### Add a charting component

- 1 In the MXML editor's Design mode, drag a charting component from the Components view into the design area.
- 2 Enter an ID for the chart.

- 3 To display more than one series of data in your chart, click the Add button and enter the new series name in the dialog box that appears.

For example, the following ColumnChart control has two data series. The bar on the left represents the gross profit for six months, and next one is the net profit during the same period.



Remove a data series by selecting it in the list and clicking the Remove button.

- 4 (Optional) Select the Include Legend option.

The Include Legend option lets you add a Legend control to the chart that displays the label for each data series in the chart and a key showing the chart element for the series.

- 5 Click OK to insert the chart.

## Apply themes

Themes allow you to implement a more personalized appearance to your applications. Flash Builder provides several themes from which you can choose. You can import additional themes or create your own themes.

Themes provided by Flash Builder include a set of Spark themes and a set of Halo themes. The default theme for Flex 4.6 and Flex 4 components is Spark. Halo is the default theme for Flex 3.

For more information on theme support in Flex see [About themes](#).

## Specify a theme

Specify themes on a project basis. After specifying a theme for a project, all applications in the project share theme.

- 1 Open the Select Project Theme dialog from either Design View or Source View of the MXML Editor:

- (Design View) Select the Appearance Panel. Then select the Current Theme.
- (Source view) From the Flash Builder menu, select Project > Properties > Flex Theme

- 2 Select a theme and then click OK.

## Mobile themes

When you create a Flex mobile project, the default mobile theme is applied. To change the default mobile theme, see [Mobile theme](#).

The number of styles that are available with the mobile theme is smaller than with other themes. For a complete list of styles supported by the mobile theme, see [Mobile styles](#).

## Import themes

You can use Flash Builder to import themes. The files for a theme must be enclosed in a folder. All required files for a Flex theme must be present.

The name of the theme is determined by the name element in the `metadata.xml` file contained within the theme folder. If the name element is not specified, or if `metadata.xml` is not present, then the name of the theme folder becomes the name of the theme.

For more information on the required format for Flex themes, see [“Create themes”](#) on page 175.

Flash Builder themes can be in the following formats:

- Theme ZIP file

Extract the contents of the ZIP file before importing the theme. The extracted contents should contain all required files.

- CSS or SWC file for a theme

The CSS or SWC file must be in a folder containing all required files for a Flex theme. When you import a theme using Flash Builder, you select either the CSS or SWC file for the theme.

- MXP file

You can use Adobe Extension Manager CS5 to package files for Flex themes in an MXP file. The theme can then be imported into Flash Builder using the Extension Manager.

For more information on packaging themes in an MXP file, see [“Create an extension file \(MXP file\) for a Flex theme”](#) on page 176.

## Import Flex themes using Flash Builder

- 1 Open the Select Project Theme dialog from either Design View or Source View of the MXML Editor:
  - (Design View) Select the Appearance Panel. Then select the Current Theme.
  - (Source view) From the Flash Builder menu, select Project > Properties > Flex Theme
- 2 Select Import Theme, navigate to the folder containing the theme to import, select the CSS or SWC file, and click OK.

## Import Flex themes packaged in an MXP file

- 1 If you have not already done so, import Adobe Flash® Builder™ 4.6 into Adobe Extension Manager CS5:

From Adobe Extension Manager, select File > Import Product.

- 2 Open Adobe Extension Manager and select Flash Builder 4.6.
- 3 Select File > Install Extension, navigate to the MXP file for the theme, and click Open.

After you accept the license, Adobe Extension Manager installs the theme into Flash Builder. The theme is now available in Flash Builder from the Select Project Theme dialog.

**Note:** You can also double-click the MXP file to start Adobe Extension Manager, which then automatically installs the theme.

## Download themes

You can download themes that can then be imported into Flash Builder.

### Download Flex themes

1 Open the Select Project Theme dialog from either Design View or Source View of the MXML Editor:

- (Design View) Select the Appearance Panel. Then select the Current Theme.
- (Source view) From the Flash Builder menu, select Project > Properties > Flex Theme

2 Select Find More Themes.

Flash Builder opens your default web browser to a page containing themes to download. You can also navigate to any other site containing themes for Flex that you can download.

3 Select a Flex theme to download.

After you download the theme, you can import the theme, as described in “[Import themes](#)” on page 174.

### Create themes

You can create your own themes and import them into Flash Builder. A Flex theme typically contains the following files:

- SWC, SWF file, CSS, PNG, JPEG, and other files that make up your theme.

The files that make up the theme can vary, but must include a SWC file or CSS file.

- `preview.jpg` file

The preview image file for the theme. If your theme folder does not contain `preview.jpg`, then Flash Builder uses a default preview image for the theme.

- `metadata.xml` file

Contains information about the theme, including which versions of the SDK the theme is compatible with. If your theme folder does not contain this file, then Flash Builder creates one when importing the theme.

Typically you package a theme in ZIP file, but the ZIP file must be extracted before you can import the theme into Flash Builder. You can also package the theme files in an Adobe Extension Manager file (MXP file), and use Adobe Extension Manager to import the theme into Flash Builder.

For more information, see [About themes](#).

### Metadata.xml file

The following table lists the elements that can be included in `metadata.xml`.

Element Name	Description
Name	The name of the theme that appears in Flash Builder.  When importing a theme using Flash Builder, you can override the name specified in the <code>metadata.xml</code> file.
Category	Author of the theme. The category under which the theme is displayed in Flash Builder.
sdk	Specifies the Flex SDK versions for which the theme is compatible. This element is a parent element for <code>minVersionInclusive</code> and <code>maxVersionExclusive</code> .  If the <code>sdk</code> element is absent, then the theme is valid for all SDKs.

Element Name	Description
minVersionInclusive	Earliest Flex SDK version for which this theme is compatible. If absent, then this theme is compatible with all earlier versions of the Flex SDK.
maxVersionExclusive	Latest SDK version for which this theme is compatible. If absent, then this theme is compatible with all later versions of the Flex SDK.
mainFile	Top-level file for implementing the theme. This file can reference other files in the theme. For example, a CSS file could reference a SWC file or SWF file.  The -theme compiler argument references the specified file.

The following example shows a typical `metadata.xml` file for a theme created by Company ABC.

```
<theme>
  <name>WindowsLookAlike</name>
  <category>ABC</category>
  <sdk>
    <minVersionInclusive>2.0.1</minVersionInclusive>
    <maxVersionExclusive>4.0.0</maxVersionExclusive>
  </sdk>
  <mainFile>WindowsLookAlike.css</mainFile>
</theme>
```

According to the `metadata.xml` file, the theme is compatible with the Flex 2.0.1 SDK. It is also compatible with SDKs up to, but not including, Flex 4.0.0. When this theme is selected, `WindowsLookAlike.css` is the file that is added to the `-themes` compiler argument.

### Create an extension file (MXP file) for a Flex theme

You can use Adobe Extension Manager CS5 to create an extension file (MXP file) for a Flex theme. The MXP file can be imported into Flash Builder using Adobe Extension Manager CS5.

Place all your theme files in a staging folder, and then create an extension installation file (MXI file) that the Extension Manager uses to create the MXP file. For information on the format of an MXI file, see the [Extension File Format](#) document.

When creating the MXI file, you specify destination paths for each of the theme's files. The destination paths are in this format:

`$flexbuilder/<Theme Name>`

- `$flexbuilder` is defined in the Flash Builder configuration file, `XManConfig.xml`. Extension Manager expands `$flexbuilder` according to this definition. `XManConfig.xml` is at the following location on your file system:  
`/<Install Dir>/Flash Builder 4/configuration/XManConfig.xml`
- `<Theme Name>` is the name of the folder that contains the Flex theme.

### Create an MXP Extension file for a Flex theme

- 1 Place all the files for the theme, including the MXI file, in a staging folder.
- 2 From the Extension Manager, select File > Package Extension.
- 3 Navigate to the extension installation file and select it.
- 4 Navigate to a location for the package file, and name it using the extension `.mvp`.

You can then test the extension file by installing it using the Extension Manager.



## Add additional themes

You can specify more than one theme file to apply to an application. If there are no overlapping styles, both themes are applied completely. There are other considerations when adding additional themes, such as the ordering of the theme files.

To add additional themes, use the command line compiler, `mxmlc` with the `theme` compiler option to specify the path to the theme files.

Using themes provides details on specifying compiler arguments and the ordering of theme files.

## Apply styles

Styles affect the appearance of an application by specifying values for visual parameters for application components. You can set styles that apply to all components in an application, to individual components, or to a set of components referenced by a style selector.

For example, you can specify styles such as:

- Text  
Font family, size, weight, color, font (bold, italic, underline) and other text display settings
- Border  
Thickness, color, rollover color, border-style (solid, inset, outset, none), corner radius, and others
- Color  
Fill color and alpha

**Note:** *The styles available for a component varies, according to the component.*

You set style properties inline on an MXML tag or separately using CSS code. The CSS code can be inside `<fx:Style>` tags in an application or in a separate CSS file.

When you apply inline styles to components, you can convert component styles into a CSS rule in an external style sheet. You can use the CSS editor to edit CSS files.

You can also convert skins applied to a component into styles.

Use Design mode of the MXML editor to apply styles to an application or specific application components. You can also use Design mode to convert applied styles or skins to CSS style sheets.

## Styles compared to skins

Skinning is the process of changing the appearance of a component by modifying or replacing its visual elements. These elements can be made up of bitmap images, SWF files, or class files that contain drawing methods that define vector images. Skins can define the appearance of a component in various states. For example, you can specify the appearance of a Button component in the up, down, over, and disabled states.

Using Flash Builder, you can convert a component's skin to CSS styles.

## Apply styles to an application

Use the Appearance view to define styles that apply to an entire application. Flash Builder saves the style defined in Appearance view as a global CSS style selector.

- 1 In Design mode of the MXML editor, open an MXML application file that contains several components.
- 2 In Appearance view, specify global styles for the application.

If Appearance view is not visible, select Window > Show View > Other > Flash Builder > Appearance.

After you apply the styles, Flash Builder creates a global CSS style selector for the specified style.

If this style is the first style referenced in the application, Flash Builder generates a new CSS file and references it in the MXML application file.

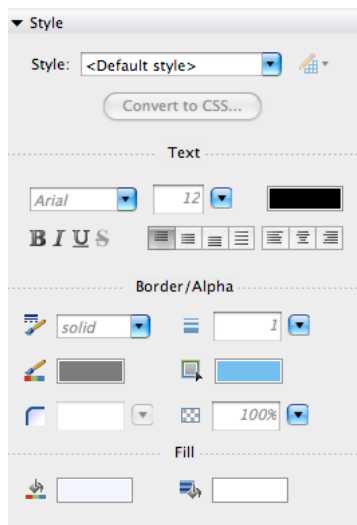
If your application already references a CSS file or a `<fx:Style>` block, Flash Builder updates the CSS with the global style selector.

## Apply inline styles to a component

Use the Properties view to define inline styles for selected components.

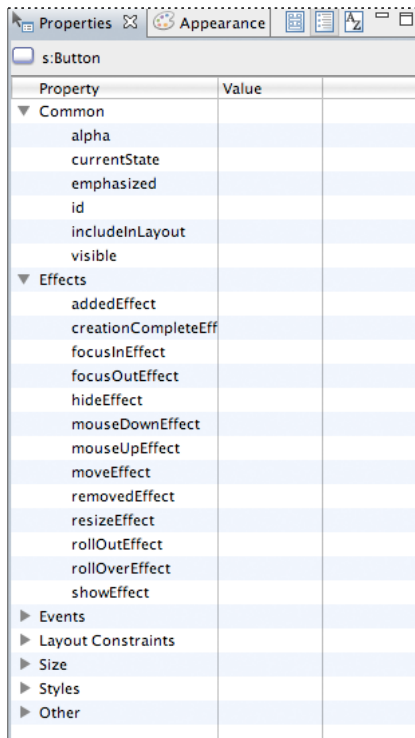
- 1 In Design mode of the MXML editor, open an MXML application file that contains several components.
- 2 Select a component and specify style property values in the Style area of Properties view.

The Style area changes, depending on the component you select.



*Properties view showing styles for a DataGrid*

- 3 In Properties view, select Category view to list all the styles that can be applied to the selected component.



**Note:** Multi-word style names in Flex can be written either like an ActionScript identifier (for example, `fontFamily`) or like similar HTML styles (for example, `font-family`).

- 4 After specifying a style, switch to Source mode to view the generated inline code that applies the style.

### Apply an external or embedded style to an application

You can embed CSS styles in an MXML application file or reference an external CSS file. The following example shows CSS code to apply styles to all Spark Button components in an application. It also creates a selector, `.myStyle`, that can be applied to any component:

```
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

s|Button { fontSize: 16pt; color: Red } /* type selector */
.myStyle { color: Red } /* class selector */
```

For styles applied to components, such as `s|Button`, the selector for components must specify a namespace. In this example, `s|Button` defines a style that is automatically applied to all Spark Button components.

Use the CSS period notation to create a selector that can be applied to any component. In this example, `.myStyle` does not have to declare a namespace and can be applied to any component.

Flex has specific requirements for creating and applying styles. For details, see *Using styles in Flex*.

### Apply styles to an application

- 1 In Design mode of the MXML editor, create an MXML application file that contains several Spark Button components and a CheckBox component.

- 2 (Embedded styles) In Source mode, add the following code to your application:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/halo";

    s|Button { fontSize: 16pt; color: Red } /* type selector */
    .myStyle { fontSize: 16pt; color: Blue } /* class selector */
</fx:Style>
```

- 3 (External stylesheet) Create a stylesheet that implements the `s|Button` and `.myStyle` selectors in Step 1. In Source mode, reference the file from the MXML application file:

```
<fx:Style source="styles.css"/>
```

- 4 Switch to Design mode. Notice that the Spark Buttons now have the style applied.

- 5 Select the Check Box component and in Properties view, select Styles > myStyle.

Now the CheckBox component has `.myStyle` style applied.

## Convert to CSS

You can convert inline styles and component skins to CSS styles. During the conversion, you can specify whether to make the styles global or to apply to a specific component. You can also specify a CSS style selector for the generated styles.

- 1 In Design mode of the MXML editor, select a component in the design area. Use inline styles or specify a `skinClass` property for the component.

See [“Generate and edit skins for Spark components”](#) on page 188 for information on specifying a `skinClass` for a component.

- 2 In Properties view, click Convert to CSS.

- 3 If you have multiple projects open in your workspace, select/deselect the resource files you want to save in the Save Resources dialog. Then click OK.

- 4 In the New Style Rule dialog box, select the `.css` file or click New to generate a new file.

- 5 Specify the Selector type. Choose from the following:

- All Components

The style is global and applies to all components in the application.

- All Components With Style Name

Components specify this style selector using the `styleName` attribute. If you choose this option, then specify a name for the Selector.

- Specific Component

The style applies only to the selected component.

- Specific component With Style Name

The style applies only to the selected component, and references the style by type selector name. If you choose this option, then specify a name for the type selector.

- 6 Click OK.

Flash Builder generates or updates the specified CSS file. Flash Builder also modifies the source code in the application to reference the type selector in the CSS file.

Flash Builder removes references to the inline style or the `skinClass` property for the component.

## Edit style rule

When external CSS styles are already applied to a component, you can quickly jump from the component to edit these styles.

- 1 Select a component.
- 2 In the Properties View, click the Edit Style Rule button next to the Style pop-up menu, then select the style you want to edit.

The CSS file opens in the CSS editor's Design mode. You use the Properties View to make further changes. You can also modify your CSS in Source mode.

## Create CSS files

Use the New CSS File wizard to create CSS files for a project. The New CSS File wizard creates a blank file that you can use to define your CSS styles.

By default, Flash Builder adds the default namespaces for Spark and MX styles.

To create a blank CSS file:

- 1 From the Flash Builder menu, select File > New CSS File.
- 2 Specify a source folder.

The source folder can be in the current project or another project.

- 3 Specify a package for the file

Select from the packages available in the project. If you want to place the file in a new package, then first create the package. Select File > New Package.

- 4 Specify a Name for the file.
- 5 Click Finish.

Flash Builder uses templates that define the contents of newly created files. You can customize the templates that Flash Builder uses. See ["Customize file templates"](#) on page 57.

## Use the CSS editor

Flash Builder provides a CSS editor which you can use to create and edit style sheets for your application. The CSS editor is available only in Source mode.

When you create a style sheet in Flash Builder, Flash Builder provides the following declarations for the Spark and MX namespaces:

```
@namespace s "library://ns.adobe.com/flex/spark";  
@namespace mx "library://ns.adobe.com/flex/mx";
```

Some Spark and MX components share the same local name. For example, there is a Spark Button component (in the `spark.components.*` package) and an MX Button component (in the `mx.controls.*` package). To distinguish between different components that share the same name, you specify namespaces in your CSS that apply to types.

If you do not use type selectors in your style sheets, then you are not required to declare namespaces. For more information, including examples, see About namespaces in CSS.

**Note:** Styles are handled differently for applications that use the Flex 3 SDK. If you are working in a Flex project that specifies the Flex 3 SDK, the CSS editor reverts to behavior implemented with Flex Builder 3. See the [Flex Builder 3 documentation](#) for information on using the CSS editor for applications that use the Flex 3 SDK.

## Add View states and transitions

You can use Adobe® Flash® Builder™ to create applications that change their appearance depending on tasks performed by the user. For example, the base state of the application could be the home page and include a logo, sidebar, and welcome content. When the user clicks a button in the sidebar, the application dynamically changes its appearance (its *state*), replacing the main content area with a purchase order form but leaving the logo and sidebar in place.

In Flex, you can add this interaction with view states and transitions. A *view state* is one of several views that you define for an application or a custom component. A *transition* is one or more effects grouped together to play when a view state changes. The purpose of a transition is to smooth the visual change from one state to the next.

### About view states and transitions

A *view state* is one of several layouts that you define for a single MXML application or component. You create an application or component that switches from one view state to another, depending on the user's actions. You can use view states to build a user interface that the user can customize or that progressively reveals more information as the user completes specific tasks.

Each application or component defined in an MXML file always has at least one state, the *base state*, which is represented by the default layout of the file. You can use a base state as a repository for content such as navigation bars or logos shared by all the views in an application or component to maintain a consistent appearance.

You create a view state by modifying the layout of an existing state or by creating a new layout. Modifications to an existing state can include editing, moving, adding, or removing components. The new layout is what users see when they switch states.

For a full conceptual overview of view states, including examples, see [View states](#).

Generally, you do not add pages to an application that is built in Flex as you do in an HTML-based application. You create a single MXML application file and then add different layouts that can be switched when the application runs. While you can use view states for these layouts, you can also use the ViewStack navigator container with other navigator containers.

When you change the view states in your application, the appearance of the user interface also changes. By default, the components appear to jump from one view state to the next. You can eliminate this abruptness by using transitions.

A *transition* is one or more visual effects that play sequentially or simultaneously when a change in view state occurs. For example, suppose you want to resize a component to make room for a new component when the application changes from one state to another. You can define a transition that gradually minimizes the first component while a new component slowly appears on the screen.

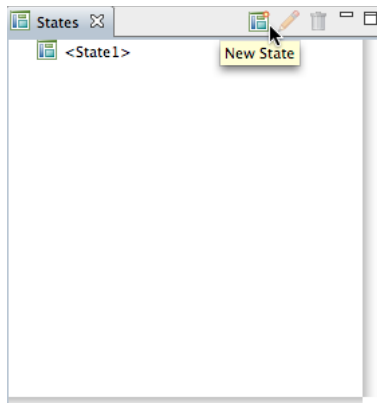
### Support for Flex 3 view states

Flash Builder provides support for view states as implemented in Flex 3. If you create a project that uses the Flex 3 SDK, the MXML editor in both Design and Source mode reverts to the Flex Builder 3 implementation. For information on editing states for the Flex 3 SDK, see the [Flex Builder 3 documentation](#).

## Create a view state

By default, an application has a single view state, which you typically use as the base state. Use the Flash Builder States View to add additional states and to edit the layout and components for each state.

- 1 Using the layout tools in Flash Builder, design the layout of the base state of your application or component.  
For more information, see [“Build user interfaces”](#) on page 168.
- 2 In the States view (Window > Show View > Other > Flash Builder > States), click the New State button in the toolbar.



The New State dialog box appears.

- 3 Enter a name for the new state.
- 4 Specify whether to create a state that is a duplicate of an existing state or to create a new, blank state. Click OK.
- 5 Use the layout tools in Flash Builder to modify the appearance of the state.  
You can edit, move, add, or delete components. As you make changes, the changes defining the new state become part of the MXML code.
- 6 Define an event handler that lets the user switch to the new state.  
For more information, see [“Switch view states at runtime”](#) on page 184.

## Set a non-base state as the starting view state

By default, an application displays the base state when it starts. However, you can set a state to be the state that displays when the application starts.

- 1 In States View (Window > States), double-click the view state that you want to use as the starting state.
- 2 In the Edit State Properties dialog box that appears, select the Set As Start State option and click OK.

## Set the view state of a component

If your application has multiple states, you can set the view state of a single component.

- 1 In Design View of the MXML editor, select a component in your layout.
- 2 In Properties View, use the In States field to select the states in which the component is visible.

## Switch view states at runtime

When your application is running, users need to switch from one view state to another. You can define event handlers for user controls so that users can switch states at runtime.

The simplest method is to assign the `currentState` property to the click event of a control such as a button or a link. The `currentState` property takes the name of the view state you want to display when the click event occurs. In the code, you specify the `currentState` property as follows:

```
click="currentState='viewstatename' "
```

If the view state is defined for a specific component, you must also specify the component name, as follows:

```
click="currentState='componentID.viewstatename' "
```

For more information, see [View states](#).

- 1 Ensure that the initial state has a clickable control, such as a Button control.

In Design mode of the MXML editor, select the control and enter the following value in the On Click field in the Properties view:

```
currentState='viewstatename'
```

*viewstatename* is the name for the state.

- 2 If you want to switch to the base state, enter:

```
currentState=' '
```

' ' is an empty string, represented by two single quotes.

- 3 To test that the states switch correctly in the application when the button is clicked, click the Run button in the Flash Builder toolbar.

You can define a transition so that the change between view states is smoother visually. For more information, see [“Create a transition”](#) on page 186.

## Create view state groups

Flex provides support for view state groups. The `stateGroups` attribute of the `<States>` tag lets you group one or more states together. For example, if multiple components appear in the same set of view states, you can create a view state group that contains all these view states. Then, when you set the `currentState` property to any view state in the group, the components appears. For more information, with examples, see [Defining view state groups](#).

Design mode of the MXML editor does not support editing state groups. Use Source mode to create and edit state groups. Source mode provides code hinting and a Show State pop-up menu to assist you in creating and editing state groups.

If you create view state group, be careful using Design View. If you delete a state using Design View, you can inadvertently leave a reference to a deleted component in a state group.

## Delete a view state

You can delete view states from an application using Design View of the MXML editor. However, if you have created a state group then use Source View to delete a state. This avoids inadvertently leaving a reference to a component in a deleted state.

- 1 In the Design View of the MXML editor, select the view state that you want to delete from the States View (Window > States).
- 2 Click the Delete State button on the States View toolbar.



## Work with multiple states in an application

If you have an application that contains more than one state, Design mode of the MXML editor allows you to switch the view for each state, displaying only the components that defined for a specific state. For each component, you can specify the state in which it is visible.

### Edit the component of a specific state

- 1 In Design View of the source editor, use the States View to add one or more additional states to an application.
- 2 Use the State drop-down list to switch the view to the selected state.
- 3 Add, move, delete, or modify the components in the state.

Changes you make to a specific state do not appear in other states unless you specify that the component appears in more than one state.

### Specify that a component appears in multiple states

- 1 In Design View of the source editor, use the States View to add one or more additional states to an application.
- 2 For any component in a state, select the component.
- 3 In the Properties View, select which states the component appears.

You can specify that the component appears in all states, or select one or more states in which the component appears.

If you specify a specific state for a component, the component does not display in the editor when editing another state.



*Be careful when editing applications that contain multiple states. Components sometimes seem to “disappear” when you switch the editor to a state that doesn’t contain a component visible in another state.*

## Create and edit view states in source code

Source mode of the MXML editor contains several features to help you edit source code for view states.

When an application declares view states, the MXML editor provides a Show State pop-up menu. When you select a specific view state in the Show State menu, components that do not appear in that state are de-emphasized in the editor.

The `includeIn` and `excludeFrom` properties for MXML components specify the view state or state group in which a component appears. Code hinting in the MXML editor assists you in selecting a view state or state group for these properties.

You can also use dot notation with component attributes to specify a view state in which the attribute applies. For example, if you want a Button component to appear in two view states, but also have the label change according to the view state, use the dot operator with the `label` property. Code hinting in the MXML editor assists you in selecting the view state. For example:

```
<s:Button label.State1="Button in State 1" label.State2="Same Button in State 2">
```

### Example working with view states in source code

- 1 Create an application that contains more than one view state.

In Source mode of the MXML editor, add the following code after the `<s:Application>` tag.

```
<s:states>
    <s:State name="State1" />
    <s:State name="State2" />
    <s:State name="State3" />
</s:states>
```

Notice that the MXML editor adds a Show State pop-up menu after you define states in the application.

- 2 In Source mode, add the following Button components:

```
<s:Button includeIn="State1" label="Show State 2"
    click="currentState='State2'" />
<s:Button includeIn="State2" label="Show State 3"
    click="currentState='State3'" />
<s:Button includeIn="State3" label="Show State 1"
    click="currentState='State1'" />

<s:Button
    label.State1="All States: State 1 Label"
    label.State2="All States: State 2 Label"
    label.State3="All States: State 3 Label"
    x="0" y="30"/>
```

By default, the editor displays code for all states.

**Note:** The click event handlers for the first three buttons cycle through the view states.

- 3 Still in Source mode, select different view states from the Show State pop-up menu.

For components that are not visible in the selected state, the editor displays the code as light gray.

All the code is editable, but de-emphasizing components that do not appear in the selected view state assists in maintaining code for each view state.

- 4 Switch to Design mode for the MXML editor.

Using either the States View or the States pop-up menu, select different view states. The editor displays the components according to properties defined for the selected view state.

- 5 Run the application. Click the top button to cycle through the view states.

For more information on creating and editing states in source code, see [Create and apply view states](#).

## Create a transition

When you change the view states in your application, the components appear to jump from one view state to the next. You can make the change visually smoother for users by using transitions. A transition is one or more effects grouped together to play when a view state changes. For example, you can define a transition that uses a Resize effect to gradually minimize a component in the original view state, and a Fade effect to gradually display a component in the new view state.

- 1 Make sure that you create at least one view state in addition to the base state.
- 2 In Source View of the MXML editor, define a Transition object by writing a `<s:transitions>` tag and then a `<s:Transition>` child tag, as shown in the following example:

```
<s:transitions>
    <mx:Transition id="myTransition">
    </mx:Transition>
</s:transitions>
```

To define multiple transitions, insert additional `<s:Transition>` child tags in the `<s:transitions>` tag.

- 3 In the `<s:Transition>` tag, define the change in view state that triggers the transition by setting the tag's `fromState` and `toState` properties, as in the following example (in bold):

```
<s:transitions>
    <mx:Transition id="myTransition" fromState="*" toState="checkout">
    </mx:Transition>
</s:transitions>
```

In the example, you specify that you want the transition to be performed when the application changes from any view state (`fromState="*"`) to the view state called `checkout` (`toState="checkout"`). The value `"*"` is a wildcard character specifying any view state.

- 4 In the `<mx:Transition>` tag, specify whether you want the effects to play in parallel or in sequence by writing a `<mx:Parallel>` or `<mx:Sequence>` child tag, as in the following example (in bold):

```
<mx:Transition id="myTransition" fromState="*" toState="checkout">
    <b><mx:Parallel>
    </mx:Parallel>
</mx:Transition>
```

If you want the effects to play simultaneously, use the `<mx:Parallel>` tag. If you want them to play one after the other, use the `<mx:Sequence>` tag.

- 5 In the `<mx:Parallel>` or `<mx:Sequence>` tag, specify the targeted component or components for the transition by setting the property called `target` (for one target component) or `targets` (for more than one target component) to the ID of the target component or components, as shown in the following example:

```
<mx:Parallel targets="{ [myVGroup1, myVGroup2, myVGroup3] }">
</mx:Parallel>
```

In this example, three `VGroup` containers are targeted. The `targets` property takes an array of IDs.

- 6 In the `<mx:Parallel>` or `<mx:Sequence>` tag, specify the effects to play when the view state changes by writing effect child tags, as shown in the following example (in bold):

```
<mx:Parallel targets="{ [myVBox1, myVBox2, myVBox3] }">
    <b><mx:Move duration="400"/>
    <b><mx:Resize duration="400"/>
</mx:Parallel>
```

For a list of possible effects and how to set their properties, see [Introduction to effects](#).

- 7 To test the transition, click the Run button in the Flash Builder toolbar, then switch states after the application starts.

## Add interactivity with effects

An *effect* is a visible or audible change to the target component that occurs over a time, measured in milliseconds. Examples of effects are fading, resizing, or moving a component.

Effects are initiated in response to an event, where the event is often initiated by a user action, such as a button click. However, you can initiate effects programmatically or in response to events that are not triggered by the user.

For example, you can create an effect for a `TextInput` component that causes it to bounce slightly when the user tabs to it, or you can create an effect for a `Label` component that causes it to fade out when the user passes the mouse over it.

You can define effects in Flash Builder as a property of an MXML component. Use Source view of the MXML editor to implement the effect.

Effects are implemented differently for Spark and MX components. For information on creating effects in MXML and ActionScript code, see [Introduction to effects](#).

## Create an effect for a component

Typically, you define effects in Source mode of the MXML editor. Effects are often invoked from a component's event handler. For example, you can use the click event handler for a Button to invoke an effect. See [Applying effects](#).

However, in Flash Builder you can define an effect for a property of an MXML component.

- 1 In the MXML editor's Design mode, click on a component in the design area.
- 2 Define the effect property for Spark components:
  - a In the Properties view, select the Category View icon.
  - b Select a property in the Effects category and specify an effect.

For example, for a Button's `rollOverEffect` property, you can specify the Fade effect. For a list of available effects, see [Available effects](#).
- 3 Save and run the file to see the effect.

## Modify user interfaces using skins

Skin classes modify the appearance of controls in a user interface. The way you create, edit, and import skins differs for Spark components and MX components.

### About Spark skins

Spark skins control all visual elements of a component, including layout. Spark skins can contain multiple elements, such as graphic elements, text, images, and transitions. Spark skins support states. You can use a skin to define the appearance of a component for each of the component's states. Skins typically specify minimum sizing requirements for the component. For details on how Spark skins are implemented in Flex, see [About Spark skins](#).

You can use Flash Builder to generate and edit skins for a Spark component. When Flash Builder generates a skin, it creates a skin class in MXML. You can modify the appearance defined by the skin in the MXML editor. Some changes can be made in Design mode of the editor, while others require you to edit the MXML file in Source mode. See [“Generate and edit skins for Spark components”](#) on page 188.

### About skins for MX components

Skins for MX components can be either a bitmap graphic or a vector graphic. A bitmap graphic, called a graphical skin, is made up of individual pixels that together form an image. A vector graphic, called a programmatic skin, consists of a set of line definitions that specify a line's starting and end point, thickness, color, and other information required by Adobe® Flash® Player to draw the line. For details on how skins are implemented for MX components in Flex, see [About MX component skinning](#).

You can use Flash Builder to import skin artwork for MX components. See [“Import skin artwork for MX components”](#) on page 192.

The `mx.skins.spark` package defines Spark skins for MX components.

## Generate and edit skins for Spark components

You can use Flash Builder to generate and edit skins for Spark components. When you generate a skin, Flash Builder uses a skin from the theme for a project. The default theme for a project is Spark. You can change the theme for a project from the Appearance view. See [“Apply themes”](#) on page 173.

When generating a skin for a component, Flash Builder creates an MXML file that implements the Skin class for the component. You can specify whether to generate the Skin class as a copy of an existing skin or generate a blank Skin class file.

Use a combination of the MXML editor in Design and Source mode to edit the skin. In Design mode, use the Outline view to select elements of the skin to edit. Use States view to navigate between the states of a component. Some parts of a skin cannot be edited in Design mode. Use Source mode to edit parts of skin that are not available in Design mode.

Some components contain subcomponents. For example, an HSlider component contains Button components that define the thumb and track of the HSlider. Subcomponents can only be skinned in Source mode.

### Component states, skin parts, and host components

Skins define the appearance of a component for each state of the component. For example, a Spark Button has four states, up, over, down, and disabled. When you generate a skin for a Spark Button, you can specify the appearance for each of these states.

Each component has parts that can be styled. For example, for a Button component, you can style the Button fill color, text attributes for the Button label, and the Rect components that make up the Button's border.

When using Flash Builder to create skins for a component, you specify a host component upon which the generated skin is based. By default, the host component is the base class of the component you are skinning. However, you can select a different host component.

**Note:** Specifying a host component for a skin class is required when generating skin classes using Flash Builder. However, if creating skin classes directly in source code, a host component is not required.

### Skinning contract between a skin and its host component

The skinning contract between a skin class and a component class defines the rules that each member must follow so that they can communicate with one another.

The skin class must declare skin states and define the appearance of skin parts. Skin classes also usually specify the host component, and sometimes bind to data defined on the host component.

The component class declares which skin class it uses. It must also identify skin states and skin parts with metadata. If the skin class binds to data on the host component, the host component must define that data.

The following table shows these rules of the skinning contract:

	Skin Class	Host Component	Required?
Host component	<code>&lt;fx:Metadata&gt; [HostComponent("spark.components.Button")] &lt;/fx:Metadata&gt;</code>	n/a	No
Skin states	<code>&lt;s:states&gt; &lt;s:State name="up"/&gt; &lt;/s:states&gt;</code>	<code>[SkinStates("up")]; public class Button { ... }</code>	Yes
Skin parts	<code>&lt;s:Button id="upButton"/&gt;</code>	<code>[SkinPart(required="false")] public var upButton:Button;</code>	Yes
Data	<code>text="{hostComponent.title}"</code>	<code>[Bindable] public var title:String;</code>	No

## Skin declaration

In Flash Builder, a Skin declaration is the file that implements the skin for a component. Flex defines a skin declaration for each visual component. When you generate a new skin for a component, Flash Builder generates the Skin declaration.

You can view the Skin declaration for selected components:

- 1 In Design mode of the MXML editor, select a Spark component in the design area.
- 2 From the context menu for the component, select Open Skin Declaration.

The class implementing the skin opens in Source mode of the editor.

If the class is one that you created, you can edit the file.

You can also do the following to open a skin declaration file:

- With the component selected, in the Style section of the Properties view click the icon near the Skin field.
- In Source mode, with a Spark component selected, from the Flash Builder menu, select Navigate > Open Skin Declaration.

## Generate and edit a skin for a Spark Button (Design mode)

This example generates a Skin class for a Spark Button, showing you how to use a combination of Flash Builder views to edit the skin. It assumes that you are working in a Flex project using the default Spark theme.

- 1 Create an application file. In Design mode of the editor, add a Spark Button to the application.
- 2 From the context menu for the Button, select Create Skin.

The New MXML Skin dialog opens.

You can also do the following to open the New MXML Skin dialog.

- In the Style section of the Properties view, select the icon near the Skin field.
  - From the Flash builder menu, select New > MXML Skin
  - In Design mode of the editor, select Design > Create Skin
- 3 Specify the following in the New MXML Skin Dialog:
    - Source Folder and Package for the generated Skin declaration.
    - Name  
The name for the Skin class you are creating.
    - Host Component  
To change the default component, click Browse and select a host component.
    - (Recommended) Select Create As Copy Of and do not remove ActionScript styling code  
If you are new to skinning, use a copy to get started creating a Skin class. Modify the ActionScript styling code.
    - (Advanced Users) Do either of the following if you are familiar with creating Skin classes:  
Remove ActionScript styling code or do not create a copy of an existing class.  
If you do not create a copy of an existing class, Flash Builder generates a blank Skin class file with some comments to guide you.  
The remaining steps of this procedure assume that you followed the Recommended option for generating a Skin class.

**4** Click Finish.

Flash Builder generates a Skin class file and opens it in Design mode of the MXML editor.

The Button component is selected.

The up state of the Button is selected in the States View.

**5** For each state of the Button, modify the Text styles, Content Background styles, and Color styles.

Use the editing tools in Style section of Properties view to make your changes.

**6** Open Outline view:

Notice that the top-level node, SparkSkin, is selected.

**7** In Outline view, select Rect shadow to define styles for the Button's shadow.

Notice that the Style section tools are not available.

**8** Switch to Source mode of the editor.

Flash Builder highlights the Rect component that defines the Button's shadow. Make any changes for the Button's shadow.

**9** Save the Skin class file and your application file.

In Design mode of the MXML editor you can view the skin for the button, assuming that you followed the recommended option in Step 3. If the styles do not show, Select the Refresh icon for the design area.

Notice that the application file added a reference to the Skin class that you created.

**10** Run the application to see how the skin changes for Up, Over, and Down states of the Button.

**Create and edit skins for Spark components (Source mode)**

You can open the New MXML Skin dialog directly in Source mode of the editor. For example, do the following to create a skinClass for a Spark Button component.

**1** In Source mode of the editor, place your cursor inside a `<s:Button>` tag and type the following:

```
<s:Button skinClass="
```

After you type the first quote for the skinClass name, a context menu appears.

**2** With Create Skin highlighted in the code hints, the Enter key opens the New MXML Skin dialog.

This dialog is the same dialog that opens in Design Mode.

See the instructions in [“Generate and edit a skin for a Spark Button \(Design mode\)”](#) on page 190 for creating the skinClass.

**3** Click Finish.

Flash Builder generates a new skinClass based on your selections in the New MXML Skin dialog. The editor switches to the source for the newly generated class.

**4** Edit the skinClass.

Save your class file and application file.

**Note:** You can convert the generated skin class to CSS to view the styles that are applied. See [“Convert a skin to a CSS style”](#) on page 192.

### Convert a skin to a CSS style

Using Flash Builder, you can convert a skin for a component into a CSS style. The advantage of converting the skin to style is to use the style as a type selector for all components of that class. Otherwise, set the `skinClass` property for each component.

The following procedure shows how to convert a skin for a Spark Button into a CSS style.

- 1 Generate and edit a skin for a Button component.
- 2 In Design mode of the editor, select the button. In the Styles section of the Properties view, click Convert to CSS.
- 3 In the New Style Rule dialog, select or create a CSS file for the style.

If you do not have a CSS file in the project that you want to use, click New to create a file.

- 4 Specify the Selector Type. Choose from the following:

- All Components

The style applies to all components in the application.

- All Components With Style Name

Components specify this style selector by name. If you choose this option, then specify a name for the type selector.

- Specific Component

The style applies only to the selected component.

- Specific component With Style Name

The style applies only to the selected component, and references the style by the type selector name. If you choose this option, then specify a name for the type selector.

- 5 After specifying a Selector Type, click OK.

Flash Builder generates or updates the specified CSS file. Flash Builder also modifies the source code in the application to reference the type selector in the CSS file.

Flash Builder removes references to the `skinClass` property for the component.

### Import skin artwork for MX components

You use the Import Skin Artwork wizard to import both vector graphics artwork and bitmap artwork from the CS5 versions of Flash Professional, Fireworks, Illustrator, and Photoshop. (For bitmap artwork, any .PNG, .JPG, or .GIF can be used). The artwork can then be used as skins for Flex components.

**Note:** Adobe provides a set of skinning templates to make it easy to create skins for the built-in Flex components. Use the templates with Flash, Fireworks, Illustrator, or Photoshop to create the artwork. You can also use Flash to create fully functional custom Flex components. For more information, see the articles in [Importing Flash Professional Assets into Flex](#).

- 1 Select File > Import > Flash Builder > Skin Artwork.
- 2 In the Import Skin Artwork dialog box:
  - Choose a folder of bitmaps or a SWC or SWF file to import skins from, or click Browse to locate one. Supported file types include the following:
    - AS3 SWF and AS3 SWC files created in Adobe Flash Professional CS5
    - Vector graphic files created in Adobe Illustrator® and exported as SWF files for Flash Player 8
    - Bitmap graphic files in PNG, GIF, and JPG formats



- Choose a folder to import the skins to. The folder must be a source folder for a Flex project (or you can specify a subfolder in the source folder). The default selection is the folder for the Flex project currently open.
  - In the Copy Artwork To Subfolder field, the default folder name is based on the folder or assets being imported. Click Browse to choose a different location.
  - In the Create Skin Style Rules In field, specify a name for a CSS file that contains the style rules. The default name is based on the name of the artwork folder or FLA file being imported.
  - Click the Delete All Existing Rules In File check box if you want the specified CSS file to be overwritten upon importing (as opposed to importing skins and keeping other existing definitions in the CSS file). The box is deselected by default, and if the CSS file does not exist it is disabled.
  - In the Apply Styles To Application field, the default is the selected file in the Flex Navigator or active editor view, or the main application file for the project.
  - Click Next.
- 3 In the next Import Skin Artwork dialog box, select the skins you want to import and specify which CSS style type and skin part property will be used. You can check items one at a time or click Check All or Uncheck All.
- If items do not have a valid style or skin part property name, they will not be checked by default. The following examples show the naming convention used in Flash Builder:
    - Button\_upSkin
    - Button\_glow\_downSkin (maps to downSkin property of Button.glow style rule)
    - TabBar-tab\_upSkin (upSkin property maps to tabStyleName property of TabBar style rule)
    - MyCustomComponent\_borderSkin
- For custom components, the item is checked if the component has been defined somewhere within the project you are importing to.
- If necessary, choose a style and skin part for the pop-up menus in each column.
  - Click Finish.
- A CSS file is created and displayed in the Source view. The CSS file is attached to the application specified in the wizard. If you import a SWC file, it is automatically added to the library path for the project.

## Generate custom item renderers

You can generate item renderers for list-based controls for desktop and mobile applications.

For desktop applications, you can generate custom item renderers for Spark list-based controls, such as List and ComboBox. You can also use Spark item renderers with some MX controls, such as the MX DataGrid and MX Tree controls.

For mobile applications, you can generate custom item renderers for mobile list-based controls.

Use custom item renderers to control the display of a data item in a DataGroup, SkinnableDataContainer, or in a subclass of those containers. The appearance defined by an item renderer can include the font, background color, border, and any other visual aspects of the data item. An item renderer also defines the appearance of a data item when the user interacts with it. For example, the item renderer can display the data item one way when the user moves the mouse over the data item. It displays the differently when the user selects the data item by clicking it.

Using Flash Builder, you can generate and edit item renderers. When Flash Builder generates item renderers for a desktop application, it uses one of the following templates:

- Spark components

Use this template for Spark list-based controls, such as List and ComboBox.

- MX Advanced DataGrid
- MX DataGrid
- MX Tree

When Flash Builder generates item renderers for list-based controls in a mobile application, it uses one of the following templates:

- Icon item renderer

Use this template to create a customized item renderer for list-based controls by specifying the label, message, and icon properties. The item renderer file is created in MXML.

For more information, see [“Generate and edit an item renderer for mobile components \(Design mode\)”](#) on page 195.

- Custom ActionScript item renderer

Use this template to create a basic item renderer for list-based controls. You can then customize the item renderer to define the appearance of the list, as necessary. The item renderer file is created in ActionScript.

You can open the New Item Renderer wizard from both Design mode and Source mode of the MXML editor. In the New Item Renderer wizard, you specify a name and template for the item renderer. Flash Builder generates an MXML or ActionScript file that implements the item renderer.

Components in the application reference the generated item renderer using the `itemRenderer` property.

For details on creating and using item renderers, see Custom Spark item renderers.



Read about [Creating Spark Item Renderers](#) by Adobe Flash Builder engineer, Balaji Sridhar.

### Generate and edit an item renderer for an MX Tree component (Design mode)

This example generates an item renderer for an MX Tree component, showing you how to use a combination of Flash Builder views to edit the item renderer. It assumes that you are working in a Flex project using the default Spark theme.

- 1 Create an application file. In Design mode of the editor, add an MX Tree component to the application.

Populate your Tree with data that can be displayed when you run the application.

- 2 From the context menu for the Tree, select Create Item Renderer.

The New Item Renderer dialog opens.

You can also do the following to open the New Item Renderer dialog.

- In the Common section of the Properties view, select the icon near the Item Renderer Field field.
- From the Flash builder menu, select New > Item Renderer.

- 3 Specify the following in the New Item Renderer Dialog:

- Source Folder and Package for the generated item renderer declaration.
- Name

The name for the item renderer class you are creating.

- Template

Select the template to use when generating the item renderer.

**4** Click Finish.

Flash Builder generates an ItemRenderer class file and opens it in Design mode of the MXML editor.

The ItemRenderer component is selected.

The normal state of the Tree is selected in States view.

**5** For each state of the Tree, modify the appearance in the generated ItemRenderer class.

**a** Open Outline view:

Notice that the top-level node, MXTreeItemRenderer, is selected.

In the Style section of Properties view, modify the appearance of tree items.

**b** In Outline view, select other components of the MXTreeItemRenderer to modify the appearance of those components.

Notice that the Style section tools are not always available.

If the Style section tools are not available, then use Source mode of the editor to define the appearance. When you switch to Source mode, the source for the selected component in Outline view is highlighted.

**6** Run the application to see how the ItemRenderer changes the appearance of the Tree.

### Generate and edit an item renderer for mobile components (Design mode)

This example generates a list-based item renderer for mobile components. It is assumed that you are working in a Flex Mobile project using the default Spark theme.

**1** Create a mobile application file. In Design mode of the editor, add components to the application.

**2** From the context menu for the mobile application file, select New > Item Renderer. You can also click Create Item Renderer in the Standard View of the Property Inspector.

The New Item Renderer dialog opens.

**3** Specify the following in the New Item Renderer dialog:

- Source Folder and Package for the generated item renderer declaration.

- Name

The name for the item renderer class that you are creating.

- Template

Select the Icon Item Renderer For Mobile List template.

- Label Field

The name of the field in the data that you want to set as the label.

- Message Field

The name of the field in the data that you want to set as the message content.

- Icon Field

The name of the field in the data that contains the path to the icon.

- Icon Width

The width of the icon. The default is 64 pixels.

- Icon Height

The height of the icon. The default is 64 pixels.

- Decorator Class

Select an image in GIF, JPEG, or PNG format. The selected image is embedded in the application. For example, `decoratorClass="@Embed('/foo/myfoo.png')`

You can also select an FXP file. An FXG file is referenced by adding the path to the FXP file. For example, `decoratorClass="{assets.Chevron}"`

- 4 Click Finish.

Flash Builder generates an `ItemRenderer` class file and opens it in Design mode of the MXML editor.

- 5 Run the application to see how the `ItemRenderer` functions.

### Create and edit item renderers (Source mode)

You can open the New Item Renderer dialog directly in Source mode of the editor. For example, do the following to create an item renderer for a Spark List component.

- 1 In Source mode of the editor, place your cursor inside a `<s:List>` tag and type the following:

```
<s:List itemRender="
```

After you type the first quote for the item renderer class name, a context menu appears.

- 2 Double-click Create Item Renderer to open the New Item Renderer dialog.

This dialog is the same dialog that opens in Design Mode.

For a desktop application, see the instructions in [“Generate and edit an item renderer for an MX Tree component \(Design mode\)”](#) on page 194 for creating the item renderer.

For a mobile application, see the instructions in [“Generate and edit an item renderer for mobile components \(Design mode\)”](#) on page 195 for creating the item renderer.

- 3 Click Finish.

Flash Builder generates a new item renderer based on your selections in the New Item Renderer dialog. The editor switches to the source for the newly generated class.

- 4 Edit the item renderer class.

Save your class file and application file.

### ItemRenderer declaration

In Flash Builder, an `ItemRenderer` declaration is the file that implements the custom `ItemRenderer` for a component.

You can view the custom `ItemRenderer` declaration for selected components:

- 1 In Design mode of the MXML editor, select a component that you have implemented a custom item renderer for.
- 2 From the context menu for the component, select Open Item Renderer Declaration.

The class implementing the item renderer opens in Source mode of the editor. You can also do the following to open the item renderer declaration:

- Select the component in Design mode. In the Common section of the Properties view, select the icon near the Item Renderer field.

- In Source mode, with the component selected, from the Flash Builder menu, select Navigate > Open Skin Declaration.

## Bind controls to data

When accessing a data service, Flash Builder provides tools to bind data to data controls, such as a DataGrid. Flash Builder creates columns in a DataGrid based on the data returned from the service. Typically, you need to configure the generated DataGrid columns. Flash Builder provides an editor to configure columns of a DataGrid and Advanced DataGrid component.

For more information on binding data to data controls, see [Binding service operations to controls](#).

## Configure DataGrid and AdvancedDataGrid components

You configure DataGrid columns in Design mode of the MXML editor. The following procedure shows how to configure the columns of a DataGrid component that access a data service. Similarly, you can configure the columns of an AdvancedDataGrid component.

### Configure DataGrid columns

- 1 In Design mode of the MXML editor, add a DataGrid (or AdvancedDataGrid) control. Bind the control to data returned from a data service.

For information, see [Binding service operations to controls](#).

- 2 Select the DataGrid and then select Configure Columns from the Property Inspector.

You can also select Configure Columns from the DataGrid's context menu.

- 3 In the Configure Columns dialog, use the Add, Delete, Up, and Down buttons to add, remove, or reorder the columns.
- 4 Use the Standard View of the Configure Columns dialog to edit the commonly used properties of a selected column.

- Data Binding

Select the data field to display in the column. The Bind To Field combo box displays all available fields from the returned data. If the DataGrid is editable, you can select whether data in this column is editable.

If the data to display does not come from a data service, then the Bind to Field is simply a text box. Use the Bind to Field to represent columns defined in the data source. For example, you could specify the names of columns defined in an XMLList. If the specified name does not correspond to a defined data source, the value is ignored and the column remains empty.

Columns that are programmatically added to a DataGrid cannot be configured using the Data Binding features.

- General Properties

Specify the header text and width of the column. Also, whether the column can be resized or sorted.

Specify Width in pixels. The default width is 100 pixels. If the DataGrid's `horizontalScrollPolicy` property is `false`, all visible columns are displayed. To ensure that all visible columns are displayed, the DataGrid does not always honor the specified value for width.

- Text Properties

Specify text formatting styles for the text in the column.

- 5 Use the Advanced View of the Configure Columns dialog to view and edit the settings for all properties of a selected column.

## Generate event handlers

Applications built with Flex are event-driven. User interface components respond to various events, such as a user clicking a button or the initialization of an object is complete. You write event handlers in ActionScript code that define how the component responds to the event.

**Note:** You can also generate event handlers for non-visible items such as *RemoteObject* and *HTTPService*.

Flash Builder provides event handler assistance that generates the event handler functions for a component. Within the generated function, you write the code that defines the component behavior in response to the event.

You access event handler assistance in three ways:

- Properties View
- Context menu for an item in Design mode of the MXML editor
- Content assist for an item in Source mode of the MXML editor

### About generated event handlers

When Flash Builder generates an event handler function, it places the event handler in the first Script block of the file. The function is placed at the end of the Script block. The generated event handler has protected access and accepts the appropriate subclass of *Event* as its only parameter.

Flash Builder generates a unique name for the event handler based on the component's class name or a custom name for the event handler that you specify. If you do not specify a custom name, the name is generated according to the following process:

- If an id property is defined, Flash Builder bases the name on the id property.
- If there is no id property defined for the component, Flash Builder generates a unique name, based on the component's class name.

You provide the body of the event handler. The following code block shows a generated event handler for a *Button*.

```
. . .
<fx:Script>
    <![CDATA[
        protected function myButton_clickHandler(event:MouseEvent):void
        {
            // TODO Auto-generated method stub
        }
    ]]>

</fx:Script>
<s:Button label="Button" id="myButton" click="myButton_clickHandler(event)"/>
. . .
```

Flash Builder designates a default event for each user interface component. For example, the default event for a *Button* is the click event. You can specify the event handler for the default event in the Standard View of the Properties View. To specify handlers for other events, in the Property Inspector select Category View > Events.

You can also use content assist in the Source View to generate event handlers.

### Generate event handlers using the Properties View

- 1 In Design mode, select an item and then select Standard View in the Property inspector.

An editing field for the default event handler is visible in the Common area.

- 2 To generate an event handler for the default event:

- a (Optional) In the On *Event* text field, specify a name for the event.

For example, in the On Click text field for a Button component, specify **MyButtonClick**.

If you do not specify a name, Flash Builder generates a unique name for the event.



*When specifying a name for the event handler, you have the option to specify an event parameter. If you do not specify the event parameter, Flash Builder generates the parameter with an appropriate event type.*

- b Click the “lightening bolt” icon, and select Generate Event Handler.

The editor switches to Source mode, with the body of the generated event handler highlighted. Type in your implementation for the event.

- 3 To generate an event handler for any event for a selected item:

- a Select Category View and expand the Events node to view all the events for the item.

- b (Optional) Double-click the name of the event to activate the text box for the event handler name. Type the name for the event handler.

- c Click the icon in the Value field to create the event handler.

The editor switches to Source mode, with the body of the generated event handler highlighted. Type in your implementation for the event.

### Generate event handlers using the context menu for an item

- 1 In Design View, open the context menu for an item.

- 2 Perform one of the following actions:

- Select the default event for the item.

For example, for a Button select Generate Click Handler.

- Select Show All Events to open the list of events in the Properties view.

Specify an event handler from the Properties view.

The editor switches to Source mode with the body of the generated event handler highlighted. Type in your implementation for the event.

### Generate event handlers using Content Assist

- 1 In an MXML block in code view, create a component, but do not specify any events.

- 2 Enable content assist for the properties of a component by typing a space after the class name.

- 3 From the list of selected properties, select an event (for example, doubleClick).

- 4 Press Control+Space and select Generate Event Handler.

Flash Builder generates a unique name for the event handler and places the event handler in the Script block.

**Note:** *If you specify a custom name for the event handler, then Flash Builder cannot generate the handler. If you want to use a custom name, first generate an event handler and then modify the name of the handler in both the event property and the generated handler.*

## Generating event handlers for components

You can generate event handlers for Flex components by clicking the Generate Event Handler button for events listed in the Property Inspector. Before generating the event handler, you can specify a custom name for the event handler. If you do not specify a custom name, Flash Builder generates a name based on the ID property of the component. If the ID property is not defined, Flash Builder generates a unique name derived from the name of the component.

### Generate an event handler for a component

- 1 In Design View of the code editor, select a component.
- 2 In the Property Inspector, select Category View, expand the list of events for the selected item, and locate the event for which you want to generate an event handler.
- 3 (Optional) Type a name for the event handler in the Value field.
- 4 Click the Generate Event Handler button.

The editor switches to Source View. Flash Builder inserts the event handler into a `<Script>` block and adds the event property to the component. The event property references the generated event.

**Note:** You can also use the option menu to generate an event handler for common events of a component. For example, for a Button, the option menu specifies Generate Click Handler.

- 5 Add the code for your implementation of the event handler to the generated event handler function.

## Access data services

In Flash Builder, you interact with data and the data-driven controls directly in your MXML and ActionScript code. You can work with data, automatically generate database applications, generate and use proxy code for web services, and generate and use code that works with the Flex Ajax Bridge. You can also manage Adobe Flash Player data access security issues and use Flash Builder with a proxy service.

## Work with data in Flash Builder

You work with data in Flash Builder by directly modifying your MXML and ActionScript application code.

### Data-driven controls and containers

Flex provides control and container components from which you build your application user interface. A number of these components present data, which users can select and interact with when using the application. Here are a few examples of how data-driven controls are used:

- On an address form, you can provide a way for users to select their home country (or other typical form input) by using the ComboBox or List controls.
- In a shopping cart application, you can use a Spark List component to present product data that includes images. For the List component you can specify the layout as VerticalLayout, HorizontalLayout, or TileLayout.
- You can provide standard navigation options by using containers such as the TabBar and ButtonBar controls.

You provide data input to all of the data-driven controls with a *data provider*.

For information about using the data-driven controls, see Spark list-based controls.



## Data providers and collections

A *collection* object contains a data object, such as an Array or an XMLList object, and provides a set of methods that let you access, sort, filter, and modify the data items in that data object. Several Adobe Flex controls, known as *data provider controls*, have a `dataProvider` property that you populate with a collection.

The following simple example shows how a data provider is defined (as an ActionScript ArrayCollection) and used by a control:

```
<!-- Simple example to demonstrate the Spark ComboBox control -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var complexDP:ArrayCollection = new ArrayCollection(
                [
                    {ingredient:"Salmon", category:"Meat"},
                    {ingredient:"Potato", category:"Starch"},
                    {ingredient:"Cucumber", category:"Vegetable"},
                    {ingredient:"Steak", category:"Meat"},
                    {ingredient:"Rice", category:"Starch"},
                    {ingredient:"Cumin", category:"Spice"}
                ]
            );

            <!-- Function to handel custom input strings -->
            private function myLabelToItemFunction(input:String):*
            {
                <!-- Returns object that matches items in dataProvider -->
                return {ingredient:input, category:"mystery"};
            }
        ]]>
    </fx:Script>
```

```
<s:Panel title="Spark ComboBox Example" width="75%" height="75%">
  <s:layout>
    <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
  </s:layout>

  <!-- Label that displayed current property values -->
  <s:Label text="Index : {cb.selectedIndex}
           Item : {cb.selectedItem.ingredient}
           Type : {cb.selectedItem.category}"/>

  <!-- ComboBox with custom labelToItem function -->
  <s:ComboBox
    id="cb"
    dataProvider="{complexDP}"
    width="150"
    labelToItemFunction="{myLabelToItemFunction}"
    selectedIndex="0"
    labelField="ingredient"/>
</s:Panel>
</s:Application>
```

For more information about data providers and collections, see [Data providers and collections](#).

## Remote data access

Flex contains data access components that are based on a service-oriented architecture (SOA). These components use remote procedure calls to interact with server environments, such as PHP, Adobe ColdFusion®, and Microsoft ASP.NET, to provide data to applications and send data to back-end data sources.

Depending on the type of interfaces you have to a particular server-side application, you can connect to an application by using one of the following methods:

- HTTP GET or POST by using the HTTPService component
- SOAP-compliant web services by using the WebService component
- Adobe Action Message Format (AMF) remoting services by using the RemoteObject component

**Note:** When you use Flash Builder to develop applications that access server-side data, use a `cross-domain.xml` file or a proxy if data is accessed from a domain other than the domain from which the application was loaded. See [“Manage Flash Player security”](#) on page 231.

You can also use Flash Builder to build applications that use Adobe ADEP Data Services, a separate product that provides advanced data service features. ADEP Data Services provides proxying for remote procedure call (RPC) service applications as well as advanced security configuration. ADEP Data Services also provides the following data services:

**Data Management Service** Allows you to create applications that work with distributed data. The Data Management Service also lets you manage large collections of data and nested data relationships, such as one-to-one and one-to-many relationships.

**Message Service** Allows you to create applications that can send messages to and receive messages from other applications, including applications built in Flex and Java Message Service (JMS) applications.

Flash Builder provides wizards and tools that to connect to data services and bind data service operations to application components. See [Building data-centric applications with Flash Builder](#).

## Data binding

In the code example in “[Data providers and collections](#)” on page 201, you may have noticed that the value of the ComboBox control’s `dataProvider` property is “`{complexDP}`”. This is an example of data binding.

Data binding copies the value of an object (the source) to another object (the destination). After an object is bound to another object, changes made to the source are automatically reflected in the destination.

The following example binds the text property of a TextInput control (the source) to the text property of a Label control (the destination), so that text entered in the TextInput control is displayed by the Label control:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600">
  <fx:Declarations>
    <!-- Place non-visual elements (e.g., services, value objects) here -->
  </fx:Declarations>

  <s:TextInput id="LNameInput" x="10" y="10"/>
  <s:Label text="{LNameInput.text}" x="10" y="50"/>
</s:Application>
```

To bind data from one object to another, you use either the curly brackets (`{ }`) syntax (as shown in the example) or the `<fx:Binding>` tag. For more information, see [Using data binding with data models and Data binding](#).

## Data models

A *data model* is an object that you can use to temporarily store data in memory so that it can be easily manipulated. You can define a data model in ActionScript, in MXML by using a tag such as `<fx:Model>`, or by using any object that contains properties. As an example, the following data model shows information such as a person’s name, age, and phone number:

```
<fx:Declarations>
  <fx:Model id="reg">
    <registration>
      <name>{nme.text}</name>
      <email>{email.text}</email>
      <phone>{phone.text}</phone>
      <zip>{zip.text}</zip>
      <ssn>{ssn.text}</ssn>
    </registration>
  </fx:Model>
</fx:Declarations>
```

The fields of a data model can contain static data (as in the example), or you can use data binding to pass data to and from the data model.

You can also define the data model within an XML file. You then reference the XML file through the file system or through a URL by using the `<fx:Model>` tag’s `source` property, as the following example shows:

```
<fx:Model source="content.xml" id="Contacts"/>
<fx:Model source="http://www.somesite.com/companyinfo.xml" id="myCompany"/>
```

For more information about data models, see [Storing data](#).

## Data validation

You use data validation to ensure that the data the user enters into your application is valid. For example, if you want the user to enter a valid ZIP code you use a ZIP code *data validator*.

Flex provides predefined data validators for the following types of data: credit card, currency, date, email, number, phone number, regular expression, social security, string, and ZIP code.

Data validators are nonvisual Flex components, which means that you do not access them from the Components panel. Instead, you work with them in code, as the following MXML example shows:

```
<!-- Define the ZipCodeValidator. -->
<mx:ZipCodeValidator id="zcV" source="{zipcodeInput}" property="text"/>
<!-- Define the TextInput control for entering the zip code. -->
<s:TextInput id="zipcodeInput"/>
```

In this MXML example, the validator is defined with the appropriate MXML tag, and it is bound to the ID property of a TextInput control. At runtime, when the user enters the phone number into the TextInput control, the number is immediately validated.

You can use data validators in ActionScript by defining a variable as an instance of a validator class and then creating a function to bind it to the input control.

Data validators are often used with data models. For more information, see [Validating Data](#).

## Data formatting

To display the proper format of certain types of data in your application, you use a *data formatter*. Flex provides predefined data formatters for the following types of data: currency, date, number, phone, and ZIP code.

Data formatters are bound to input controls, and they format data correctly after the user enters it. For example, a user can enter a date in this format:

120105

Bound to the text input control is a data formatter that stores and displays the date in this format:

12/01/05

As with data validators, data formatters are nonvisual Flex components that you can work with either as MXML tags or as ActionScript classes.

For more information, see [Formatting Data](#).

## Configure access to data services

The New Project wizard for Flex projects and Flex Mobile projects provide options for configuring access to data services. The steps in the wizard are the same for both Flex projects and Flex mobile projects. You can configure access to the following types of services:

PHP services
ColdFusion services
Java services
ASP.NET

## Configure access to PHP services

To access data from PHP services, a server hosting the services must be available. The server can be either a local server or a server available from a local network.

- 1 In the New Project wizard, specify Project Location. For Mobile projects, specify Mobile Settings.

See “[Flex projects](#)” on page 81 and “[Flex Mobile Projects](#)” on page 31.

- 2 For application server type, choose PHP.
- 3 Specify the Web Root and Root URL for the service. Click Validate Configuration.

Typically, you specify a Web root and root URL that is local to your environment. However, you can also access network servers. Make sure that the directory is a shared directory and the account under which Flash Builder is running has write access.

Make sure that you have mapped or mounted a drive for the network server. Then specify a path to the server. The path is platform-specific. For example:

(Windows) \\10.192.18.12\server\webroot

(Windows) Z:\webroot

(Mac) /Volumes/webroot

- 4 (Optional) Specify the output folder for your application.
- 5 Click Finish, or click Next to select more configuration options.

See “[Build paths, native extensions, and other project configuration options](#)” on page 87.

## Configure access to ColdFusion services

To access data that uses ColdFusion, you must have either Adobe ColdFusion® 8 or Adobe ColdFusion 9. For more information, see the [ColdFusion product page](#).

- 1 In the New Project wizard, specify Project Location. For Mobile projects, specify Mobile Settings.

See “[Flex projects](#)” on page 81 and “[Flex Mobile Projects](#)” on page 31.

- 2 For application server type, select ColdFusion, then choose from the following options:

**Use Remote Object Access Service** If you deselect Use Remote Object Access Service, specify your Web Root and Web Root URL in the next step.

If you select Use Remote Object Access Service, you have the following choices:

- ADEP Data Services

Specify ADEP Data Services as a ColdFusion application type only if your ColdFusion 8 installation is configured for ADEP Data Services 2.6.1 (formerly known as LiveCycle Data Services). See [Integrating Adobe LiveCycle Data Services ES 2.6 with Adobe ColdFusion 8](#).

Typically for ADEP Data Services, you specify Java as the application server type, not ColdFusion. See “[Configure access to Java services](#)” on page 206.

- BlazeDS

Specify BlazeDS as a ColdFusion application type only if your ColdFusion 8 installation is configured for Adobe BlazeDS 3.1. See [Integrating BlazeDS with a ColdFusion 8 Installation](#).

Typically for BlazeDS, you specify Java as the application server type, not ColdFusion. See “[Configure access to Java services](#)” on page 206.

- ColdFusion Flash Remoting

Use this option if you plan to use data-centric development tools available with Flash Builder. This option also applies if you use Flash Remoting to invoke methods in ColdFusion Components (CFCs). See Building data-centric applications with Flash Builder.

**3 Specify a server location, Web Root, Web Root URL, and Context Root:**

If accessing a remote object service, you can configure a stand-alone ColdFusion configuration or a ColdFusion configuration deployed to a Java application server:

- Standalone

Use the Standalone option if your ColdFusion installation uses the server configuration.

Specify the location of the ColdFusion server, location of the Web Root, and Web Root URL.

- Deployed to Java Application Server

Use the Deployed to Java Application Server option if your ColdFusion installation uses either the multiserver or Java configurations.

Specify a Web Root, Root URL, and Context Root. If you are using the ColdFusion multiserver configuration, you typically do not have to specify the Context Root.

The context root typically matches the last segment of the root URL path when you deploy ColdFusion as a web application in the ColdFusion Java configuration.

When specifying the location of the server and web root, navigate to a local directory or specify a path to a directory on a network server. Make sure that the directory is a shared directory and the account under which Flash Builder is running has write access.

Make sure that you have mapped or mounted a network drive for the network server. The path to a network server is platform-specific. For example:

(Windows) \\10.192.18.12\server\webroot

(Windows) Z:\webroot

(Mac) /Volumes/webroot

**4 Click Validate Configuration to ensure that the setup is correct.**

If the web root directory is not writable, then Flash Builder displays a warning.

**5 Choose an output folder for the compiled application.**

**6 Click Finish, or click Next to select more configuration options.**

See [“Build paths, native extensions, and other project configuration options”](#) on page 87.

## Configure access to Java services

This project configuration lets you create Flex projects that use Java-based service classes with the remote object access service option. When no option is selected, and Java server is used, an output folder is created under the server root. If you installed the Eclipse Web Tools Project (WTP) plug-in, you can create combined Java and Flex projects with or without remote object access service.

**Note:** ADEP Data Services (formerly known as LiveCycle Data Services) and BlazeDS support specific versions of the Flex SDK. Check the [ADEP Data Services Compatibility Matrix](#) to see which versions of the Flex SDK your version of ADEP Data Service supports. The compatibility matrix also lists the versions of the Flex SDK that BlazeDS supports.

- 1 In the New Project wizard, specify Project Location. For Mobile projects, specify Mobile Settings.

See “[Flex projects](#)” on page 81 and “[Flex Mobile Projects](#)” on page 31.

- 2 For application server type, choose Java.

- 3 (Optional) Select the Use Remote Object Access Service option.

Data Services ES is automatically selected. You can select BlazeDS. If you installed WTP, you can also create a combined Java and Flex project that uses WTP (the Java source folder is selected for you).

- 4 Configure the Java application server.

- If you selected the Use Remote Access Service and Data Services ES or BlazeDS options, specify the root settings:

**Root Folder** Physical location of the web application server that serves your application’s data (for example, C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\myservices). If you are using a remote server, it must be on a mapped drive or accessible using UNC.

**Root URL** Root URL of the web application. This URL corresponds to the physical location named in Root Folder, above. For BlazeDS on Tomcat, this URL can be:

`http://localhost:8080/myservices`

For Data Services, the default root URL is:

`http://localhost:8400/lcds/`

If you use a remote server, the URL can be as follows:

`http://myserver.com:8400/lcds/.`

**Context Root** The context root typically matches the last segment of the root URL path. For the examples in Root URL, above, the context root would be `/myservices` for BlazeDS and `/lcds` for Data Service.

- If you selected the Create Combined Java/Flex Project Using WTP option (with or without ADEP Data Services):

- Specify the names of your Java and Flex source folders and target runtime.

When you create a Flex project with ADEP Data Services, Flash Builder either creates a directory with the same name as your project, or uses an existing directory with that name. That directory is a subdirectory of the root folder that you specified for the project.

- With Data Services ES, specify a flex.war file, which is located in the server installation folder.

**Note:** Regardless of which option you choose for an ADEP Data Services project in Flash Builder, specify a valid root folder and root URL. These values map the root of an ADEP Data Services web application. If you deselect the options, enter only your web root and root URL.

- 5 Specify the location to compile your project.

- 6 Click Finish, or click Next to select more configuration options.

See “[Build paths, native extensions, and other project configuration options](#)” on page 87.

## Create a Flex project with ASP.NET

With Microsoft Windows and Microsoft Visual Web Developer installed, you can create Flex projects that use ASP.NET for deployment. Also, if you have access to an Internet Information Service (IIS) development server, you can create Flex projects with a Flex output folder under IIS.

- 1 In the New Project wizard, specify Project Location. For Mobile projects, specify Mobile Settings.

See [“Flex projects”](#) on page 81 and [“Flex Mobile Projects”](#) on page 31.

- 2 For application server type, choose ASP.NET.

- 3 Select the ASP.NET server:

- If you are using an ASP.NET Development Server, there is no need to specify a server location.
- If you are using IIS, enter the Web Application Root and Web Application URL.
- Specify the output folder for your application.

- 4 Click Finish, or click Next to select more configuration options.

See [“Build paths, native extensions, and other project configuration options”](#) on page 87.

## Change server options of existing projects

At times, the original server configuration for a project does not meet your current needs. You can then reconfigure the server configuration for a web application or desktop application from the Project Properties window.

In the Project Properties window, select the Flex Server option to add or change the server options for the project:

- Select None to remove the server configuration from a project.

Removing the server configuration from a project removes any added SWCs on the library path for that server type.

- Select a server type to change or add the server configuration of a project

All the server options for the selected server configuration are available. See [“Flex projects”](#) on page 81 for details on server configuration settings.

Changing the server type of a project can result in errors in existing code that relies on the original server type. Investigate and correct any resulting errors in your code.

## Monitor applications that access data services

The Network Monitor is a useful tool for monitoring and debugging applications that access data services. The Network Monitor allows you to examine the data that flows between an application and a data service. It also examines XML, AMF, and JSON data, which are sent using SOAP, AMF, HTTP, and HTTPS protocols.

The Network Monitor is active in the Flash Development and Flash Debug perspectives.

## Enable network monitoring

You enable the Network Monitor for individual Flex projects. The monitor state (enabled or disabled) applies to all applications within that project. You cannot enable or disable the Network Monitor on an individual application basis.

By default the Network Monitor is not enabled. You enable the Network Monitor by selecting the Enable Monitor icon in the Network Monitor toolbar.



This procedure assumes that you are in the Flex Development or Flex Debug perspective.

- 1 If the Network Monitor view is not open, from the Flash Builder menu select Window > Show View > Other > Flash Builder > Network Monitor.
- 2 If the Network Monitor is not enabled, in the Network Monitor toolbar click the Enable Network Monitor button. This button is a toggle for enabling or disabling the Network Monitor.

## Monitor remote services

To monitor your application, run either the development or debug version of the application with the Network Monitor enabled.

In general, the Network Monitor captures and stores all event data until you either quit the application or explicitly clear the data. The events are displayed in chronological order.

### Start a monitoring session

- 1 Run either a development or debug version of the application that accesses remote services.
- 2 For each access to a remote service, the Network Monitor lists the following:
  - Time of the request
  - Requesting service
  - The operation and URL if applicable
  - The time of the response
  - Elapsed time
- 3 Select a column header to sort the returned data according to the values in that column.  
Click the column again to invert the order of the data.
- 4 Select the request and parameter tabs at the bottom of the monitor to view the details about the request operation.  
The actual data sent in the request, as well as other information about the request, can be viewed from these tabs.
- 5 Select the response and result tabs at the bottom of the monitor to view the details about the response.  
The actual data sent in the response, as well as other information about the response, can be viewed from these tabs.
- 6 Double-click an entry to go to the source code for that operation.  
The Flash Builder source editor opens with the relevant line of source code highlighted.  
***Note:** For most events, the Network Monitor can correlate an event with the Flex source code. For some events that are triggered outside the scope of the Network Monitor, the monitor cannot find the Flex source code.*
- 7 Click the Save button on the Network Monitor toolbar to write all captured information to an XML file.  
***Note:** Use the generated XML file to study the data offline. You cannot import the data from this file back into the Network Monitor.*
- 8 Click the Clear icon in the Network Monitor toolbar to remove all captured information from the monitor.

### Suspend a monitoring session

You can suspend and resume network monitoring. Suspending and resuming a session applies to all applications in the Flex project. For example, you cannot suspend one application in the project and continue monitoring another.

- 1 Click the Suspend button in the Network Monitor toolbar to suspend the monitoring of a session.

- 2 Click the Resume button in the toolbar to continue monitoring the session.

## Stop a monitoring session

To stop monitoring a session, you disable the Network Monitor.

- 1 (Optional) Close the Network Monitor.

***Note:** Simply closing the Network Monitor does not stop the monitoring session. Monitoring is still active, even if the Network Monitor is closed.*

- 2 Click the Enable Network Monitor button.

This button is a toggle for enabling or disabling the Network Monitor.

***Note:** Disabling the Network Monitor applies to all applications in the Flex project.*

## Support for HTTPS protocol

The Network Monitor supports monitoring HTTPS calls to a server certified by a certificate authority (CA) or that has a self-signed certificate.

To monitor calls over the HTTPS protocol, modify the default preference for the Network Monitor to ignore SSL security checks. Open the Preferences dialog and navigate to Flash Builder > Network Monitor.

## View Network Monitor data

The leftmost panel of the Network Monitor provides information about the source of the data. It displays the following information:

- Source URL for the data service
- The type of service displayed  
For example RemoteService, HTTPService, or WebService.
- The request time, response time, and elapsed time for the data request
- The name of the operation called from the data service.

The Network Monitor has two tabs for viewing data, allowing you to view the request data and response data.

For each request and response, you can view the data in Tree View, Raw View, or Hex view. Select the corresponding icon for each view to change how the Network Monitor displays the data.

- Tree View  
Shows the XML, JSON, and AMF data in a tree structure format. This is the default view for data.
- Raw View  
Shows the actual data that is transferred.
- Hex View  
Shows the data in hexadecimal format. Hex view is useful when debugging binary data sent over a network.

By default, the Network Monitor clears all recorded data with every launch of an application. However, you can change the default behavior and retain all monitor data. Retained monitor data includes the data from all applications in all projects. Open the Preference dialog and navigate to Flash Builder > Network Monitor. Deselect Clear Entries on Start.

## Save Network Monitor data

You can save Network Monitor data to an XML. Click the Save button in the Network Monitor view to save Network Monitor data.

## Monitor multiple applications

You can monitor multiple applications simultaneously. There are two scenarios for monitoring multiple applications:

- Monitoring multiple applications in the same project

You can only have one Network Monitor per Flex project. When monitoring multiple applications in the same project, events from all applications appear in the monitor according to the time the event occurred.

You cannot filter events in the monitor according to specific applications.

- Monitoring multiple applications in different projects

You can open a Network Monitor for each active Flex project. Each Network Monitor is independent of the other monitor, displaying only the events for its specific project.

Suspending or disabling a Network Monitor in one project does not apply to monitors in other projects.

## Monitor mobile applications

The Network Monitor lets you examine the data that flows between a mobile application and a data service.

You can enable the Network Monitor to monitor mobile applications by following these steps:

- 1 Run the debug configuration of the application with the Network Monitor enabled.
- 2 Specify a launch method.

- **On Desktop** Select this launch method if you don't have a mobile device and you want to monitor the application on your desktop.

Select On AIR Simulator to monitor the application on a simulated device that is created using the AIR Debug Launcher (ADL). The device is simulated according to the device configuration that you select.

- **On Device** Select this launch method to monitor the application on a mobile device.

Typically, you use the On Device launch method to monitor applications that access the device's native code. Flash Builder can access the device either by connecting to your computer's USB port or over the network via Wi-Fi.

Whether you connect your device over USB or over Wi-Fi, the Network Monitor records calls only if the device and the host machine are connected to the same network via Wi-Fi.

If the Network Monitor cannot establish a connection via Wi-Fi, Flash Builder displays a dialog requesting the IP address of the host machine. Once the connection is established, the network calls from the application are rerouted to Flash Builder via the device's Wi-Fi network.

- 3 Monitor the mobile application as you would monitor a web or desktop application that accesses data services. For more information, see [Monitor applications that access data services](#).

## Limitations of the Network Monitor

Be aware of the following limitations when monitoring network data:

- The Network Monitor does not support applications that were created using pure ActionScript and Library projects.
- The Network Monitor does not support the Real Time Messaging Protocol (RTMP). For example, you cannot monitor streaming video.

## Use Flex library projects

Library projects let you build custom code libraries that you can share between your applications or distribute to other developers. A library project generates a SWC file, which is an archive file for Flex components and other resources. For example, the Flex framework is contained in SWC files. When you create a Flex project, the Flex framework SWC files are added to the project's library path. You can view and edit the library path by accessing the project's build path properties page (for Flex projects, select Project > Properties > Flex Build Path).

Archived into a SWC file is a SWF file that contains components and resources and a catalog.xml file that is the manifest of the elements contained within the SWF file. Typically, the SWF file contains one or more components and any other required resources. Adding the library to a project lets you use those components in your application and also enables code hinting for those components.

In addition to providing a convenient way to package and distribute components, SWC libraries are used as themes, the visual appearance of applications built in Flex. A SWC theme file contains a CSS file and all the related graphic assets. For more information about creating and using themes, see About themes.

Libraries are useful if you create components entirely in ActionScript and use them in Design mode in Flash Builder. ActionScript components are not visually rendered in Design mode until they are compiled into a SWF file. By adding ActionScript components to a library project, you create a SWF file that is contained in a SWC file. You can add the library to a project's library path, and the ActionScript components visually render in Design mode when you add them to the application.

## Configure libraries for your applications

You use SWC libraries in your projects in the following ways:

**Merged into the application** When you add a SWC file to the project's library path, the components contained in the library are available to use in your application. When you build the application, only the library components you actually used are compiled into the application SWF file. In other words, all of your application code is merged into a single SWF file. This is the most common and straightforward way of using library components.

**External to the application** You can keep library components separate from the compiled SWF file, so they are not merged into the file. The compiler resolves all code contained in the library that is used by the application, but does not merge the code into the application SWF file. The advantage of this approach is that you make the application SWF file smaller. The components contained in the SWC file are retrieved and loaded into memory as needed, at runtime.

**Runtime Shared Library** In Flex projects only, you can also use SWC files as a Runtime Shared Library (RSL), which is similar to a dynamically linked library on other platforms. Use SWC files as an RSL when you have a collection of components that are used by more than one application.

There are several advantages to sharing components between applications by using an RSL. First, the library is loaded into memory once, cached, and then available to all the applications that use those components. Second, the components contained within the library are only loaded when they are needed, which reduces the application's

startup time because the size of each application is smaller. The potential problem to this approach is that the entire RSL is loaded into memory, rather than the individual components that the applications use. For more information about when to use SWC files as an RSL, see Runtime Shared Libraries.

## Create Flex library projects

When you create a library project, the New Flex Library Project wizard guides you through the steps, prompting you for the project name, location, and build path information.

The first step in creating a SWC file in Flash Builder is to create a Flex Library project. After you create the Library project, you add components, specify the library project elements to include in the SWC file, and then build the project to generate the SWC file.

- 1 Select File > New > Flex Library Project.
- 2 Enter a Project name, and then specify the following:

**Project Location** The default location is the current workspace. On Windows platforms, the default workspace location is C:\Documents and Settings\username\Adobe Flash Builder\. On the Macintosh, the default workspace location is /Users/username/Adobe Flash Builder/. You can choose a different project location by deselecting the Use Default Location option.

**Configuration** You can specify if the Flex Library project uses generic libraries or mobile libraries. Generic libraries are used for web, desktop, and mobile projects.

You can also specify if the Flex Library project has to be compatible with Flash Catalyst. In that case, generic libraries are used. Mobile libraries are not supported for Flash Catalyst compatible projects.

**Flex SDK Version** Choose default or specific. You can also click the Configure SDKs link to add, edit, or remove SDKs on the main Preferences page.

**Include Adobe AIR libraries** Select this option if your library must use AIR features, such as access to the AIR APIs. Flash Builder then changes the library path of this new Flex Library project so that it includes airglobal.swc and airframework.swc. Web-based Flex projects cannot use this library.

Do not select this option if you are writing a generic library intended to be used only in a web-based application, or in either a web-based or AIR-based application.

- 3 Click Next.
- 4 (Optional) Set the build path information. For example, you can add folders to the project's source path that contains the components to include in the SWC file. You can also add other projects, folder, or library SWC files to include in your library project. See [“Use SWC files in your projects”](#) on page 215.
- 5 When you finish entering the project settings, click Finish.



For best practices while creating Flex libraries, see the article [Three points to remember when creating a library](#) by Adobe Flex Community Expert, Xavi Beumala.

## Create an AIR library project

To create an AIR code library for multiple AIR projects, create an AIR library project using the standard Flex library project wizard.

- 1 Select File > New > Flex Library Project.
- 2 Specify a project name.
- 3 Select Include Adobe AIR Libraries and then click Next.

- 4 Modify the build path as needed and then click Finish. For more information about creating library projects, see “About library projects” in the Flash Builder Help.

## Add components to the library project

You add components to the library project in the following ways:

- Add new or existing custom components, ActionScript classes, and other assets to the project.
- Link to existing components in other projects in the workspace. (See “[Link to resources outside the project workspace](#)” on page 40.)
- Add a linked folder that contains components to the library project’s source path. (See “[Add an external resource folder to the source path](#)” on page 42.)

***Note:** All the components you include in the library project must be associated with the library project (directly or as linked resources).*

## Select library project elements to include in the SWC file

To select the elements (components and resources) to include in the SWC, follow these steps:

- 1 Select Project > Properties > Flex Library Build Path.

The components that you added to the project (either directly or by linking to them) appear in the Classes tab.

- 2 Select the component classes to include in the SWC file.
- 3 (Optional) Select the Resources tab and then select the resources to include in the SWC file.
- 4 After you make your selections, click OK.

## Build library projects

After you select elements to include in the SWC file, and if you selected the Build Automatically option, the SWC file is immediately compiled and generated into the project’s output folder. If you build your projects manually, you can build the library project when you want by selecting Project > Build Project or Build All.

Building your library project generates a SWC file, which you can share with other applications or users.

A SWC file is an archive file. You can open the SWC file in any archive utility, such as Winzip. Inside the SWC file are the library.swf and catalog.xml files. There also are properties files and other embedded assets.

You can export the library as an open directory rather than as a SWC file. You typically export a library as an open directory when you plan on using the library.swf file inside the SWC file as an RSL.

You do this by setting the `directory` and `output compiler` options. You set the `output` option to the name of a directory to create, and set the `directory` option to `true` to indicate that you want an open directory and not a SWC file when you build the library. To edit the compiler options, select Project > Properties > Flex Library Compiler, and add the options to the “Additional compiler arguments” field; for example:

```
-directory=true -output=myOpenDir
```

Flash Builder creates a directory in the project named `myOpenDir` and stores the contents of the SWC file in that directory.

## Use SWC files in your projects

To use SWC files in your Flex projects, you add them to the project's library path. The SWC files can be located in the project, in a Flex library project, in a shared folder within the workspace, or any other location that has been linked to the project (using a shared folder that was added to the project's source path, for example).

When you use SWC files in applications, there are configuration options that determine whether they are statically or dynamically linked to the application, merged into the application SWF file, or external to it and accessed separately at runtime.

### Add a SWC file to the library path

- 1 With a project selected in the Package Explorer, select Project > Properties > Flex Build Path.
- 2 Click the Library Path tab.
- 3 Select any of these options to add SWC files:

**Add Project** Adds a Flex library project.

**Add SWC Folder** Lets you add a folder that contains SWC files.

**Add SWC** Adds a compiled SWC file.

**Add Flex SDK** Lets you add other Flex SDKs. If your project already has a Flex SDK in its library path, this button is disabled. If you remove the existing Flex SDK from your library path, the button is enabled. When you click this button, a Flex SDK node is added, but you are not prompted which one is added. To control which Flex SDK to use, select Project > Properties > Flex Compiler.

- 4 Enter or browse to and select the location of the SWC file, project, or folder. Click OK.  
The SWC file, library project, or folder is added to the library path.

### Merge the SWC file into the application SWF file when compiled

- 1 With a project selected in the Package Explorer, select Project > Properties > Flex Build Path.
- 2 Click the Library Path tab, and then select and expand the SWC file entry to display the SWC options.
- 3 Double-click the Link Type option. The Library Path Items Options dialog box appears.
- 4 Select the Merged into Code option, and click OK.

This procedure is the equivalent of using the `library-path` compiler option.

### Set the SWC file as an external library file

- 1 With a project selected in the Package Explorer, select Project > Properties > Flex Build Path.
- 2 Select the Library Path tab, and then select and expand the SWC file entry to display the SWC options.
- 3 Double-click the Link Type option. The Library Path Items Options dialog box appears.
- 4 Select the External option, and click OK.

This procedure is the equivalent of using the `external-library-path` compiler option.

## Use the SWC file as an RSL

You can externalize the shared assets in your applications' SWF files into stand-alone files. These stand-alone files can be separately downloaded and cached on the client. The shared assets are transferred and loaded only once to the client but any number of applications can use them at runtime. These shared files are known as *Runtime Shared Libraries* or *RSLs*.

- 1 With a project selected in the Package Explorer, select Project > Properties > Flex Build Path.
- 2 Select the Library Path tab, and then select and expand the SWC file entry to display the SWC options.
- 3 Double-click the Link Type option or click Edit. The Library Path Items Options dialog box appears.
- 4 Select Run-time Shared Library (RSL) as the link type.
- 5 You can add a URL to specify where the SWC library resides when the application is deployed. You can also edit an existing URL, if necessary.
- 6 Select Force Load RSL to force-load an unused RSL during compile time.
- 7 You can specify the application domain in which the cross-domain RSLs have to be loaded.

Using the SWC files as an RSL simplifies the process for using RSLs manually. To do so, you extract the SWF file from the SWC file and set the values of the `runtime-shared-library-path` compiler option.

For more information about using SWC files as an RSL, see *Runtime Shared Libraries* in *Using Adobe Flex*.

## Create custom MXML components

You can create custom components to add functionality to an existing component, or to build a reusable component, like a search box or the display of an item in a data grid. You can also write a new component that isn't available in the Flex framework.

The components that you create are displayed in the Components view in Design mode. You can distribute the components using SWC files. For more information, see ["Use SWC files in your projects"](#) on page 215.

If your component is composed mostly of existing components, it is convenient to define it using MXML. However, if it is a new type of component, define it as an ActionScript component. For more information, see ["Create an ActionScript class"](#) on page 38.

- 1 Select File > New > MXML Component.

The New MXML Component dialog box appears:

- 2 Specify the parent folder for your custom component file.

Save the file in a folder in the current project folder or in the source path of the current project if you want the component to appear in the Components view.

- 3 Specify a filename for the component.

The filename defines the component name. For example, if you name the file `LoginBox.mxml`, the component is named `LoginBox`.

- 4 In the Based On field, select the base component of your custom component.

Custom components are typically derived from existing components. Containers are commonly used as the base components of layout custom components.

Starting with Flex 4, Flash Builder suggests `spark.components.Group` as the base component.



Select Browse to open the Open Type dialog and select a component.

Modify or clear the suggested component to broaden the selection in the Open Type dialog. For example, specify **spark.components**, before clicking Browse.

You can filter the selection in the Open Type dialog, based on your needs. See “[Browse and view classes](#)” on page 76 for information on using the Open Type dialog.

- 5 (Optional) If you base the component on any container, you get options to set the width and height of the component.

You can set these options to a fixed width and height or to percentages, or you can clear them. When you create an instance of the component, you can override the component’s width and height in the instance.

If you set a percentage width and height or if you set no width and height, you can preview how the component looks at different sizes using the Design Area pop-up menu in the toolbar of the MXML editor’s toolbar in Design mode. For more information, see “[Design components visually](#)” on page 217.

- 6 Click Finish.

Flash Builder saves the file in the parent folder and opens it in the editor.

If you saved the file in the current project or in the source path of the current project, Flash Builder also displays the component in the Components view so that you can rapidly insert it in your applications. For more information, see “[Add components in MXML Design mode](#)” on page 21.

**Note:** The Components view lists only visible custom components (components that inherit from the `UIComponent` class). For more information, see the [ActionScript 3.0 Reference for the Adobe Flash Platform](#).

- 7 Create your custom component.

For more information, see Simple MXML components.

## Design components visually

You can lay out a custom component visually in the MXML editor as you would a regular MXML application file. All the tools in Design mode are available. For example, you can add child controls from the Components view and set properties in the Properties view. For more information, see “[Work with components visually in MXML Design Mode](#)” on page 20.

The following rules apply to the size of the custom component:

- If your component has a fixed width and height, Flash Builder automatically sets the design area to that width and height.
- If the component has no width and height, or has a 100% width and height, you can select the size of the design area from the Design Area pop-up menu in the editor’s toolbar.

Selecting different sizes allows you to preview the component as it might appear in different circumstances. The design area defaults to 400x300 pixels for containers and to Fit to Content for controls.

- If the component has a percentage width and height other than 100%, Flash Builder renders it as a percentage of the selected size in the Design Area menu.

## Provide ASDoc comments for custom components

You can document your custom components by adding ASDoc comments to the code that implements the components. ASDoc comments are then available with Content Assist in the MXML and ActionScript editors. For more information, see “[Content Assist](#)” on page 44.

Add ASDoc comments to ActionScript source files to provide API reference documentation. You can also add ASDoc comments to document MXML elements. See [ASDoc](#) for details on creating ASDoc comments for your source files.

## Edit and distribute custom MXML components

Flash Builder renders any visible custom components (components that inherit from `UIComponent`) in the MXML editor's Design mode. You can double-click a custom component in the layout to open its file and edit it.

- 1 Open an MXML file that uses a custom component.
- 2 In Design mode, double-click a custom component in the layout.

The component file opens in the editor.



*You can also select **Open Custom Component** from the context menu.*

- 3 Edit the custom component.

Distribute custom components by creating library projects. For more information, see [“Use Flex library projects”](#) on page 212.

## Create modules

You can create, add, optimize, and debug modules in Flash Builder. For information on writing module code, see [Modular applications](#).

The following steps describe how to create a module in Flash Builder. After you create a module, you can compile it.

- 1 In Flash Builder, select **File > New > MXML Module**. The New MXML Module dialog box appears.
- 2 Select a parent directory for the module. You typically store modules in the same directory as the main application so that relative paths to shared resources are the same.

Because modules are runnable, they must be in the project's source folder.

- 3 Enter a filename for the module; for example, `MyModule`.
- 4 Enter the Width, Height, and Layout properties for the module.
- 5 Specify whether to optimize the module.

If you optimize a module for an application, classes used by the application are excluded from the module. This can result in smaller download sizes for your SWF files. For more information, see [“Optimize modules in Flash Builder”](#) on page 222.

- **Optimize for Application**

If you select this option, specify an application to optimize the module against.

- **Do Not Optimize**

If you select this option, all classes are included in the module, whether they are defined in the main application. This can improve the performance of the incremental compilation. In addition, you can load the module into any application because it has all of its dependencies compiled into it.

- 6 Click **Finish**.

Flash Builder adds a new MXML module file in your project.

## Compile modules

In Flash Builder, you can either run the module as if it were an application or you can build the module's project. If the modules are in the same project as your application, then when you run your application, Flash Builder compiles SWF files for all modules in the project. The SWF files are then loaded into the application at runtime.

You cannot run the module-based SWF file as a stand-alone SWF file or load it into a browser window. It must be loaded by an application as a module. If you run the module in Flash Builder to compile it, close Adobe Flash Player or the browser window and ignore any errors. Modules should not be requested by the Player or through a browser directly.

The module SWF files and main application SWF file are typically in the same directory, although Flash Builder compiles the modules at the same time as your application, regardless of their location. Modules can be in the same directory as the application or in subdirectories.

You can also create a separate Flex or ActionScript project for each module or for groups of modules. This gives you greater control over how modules are compiled because each project can have different compiler options than the application or other modules. It also lets you compile the module's project or projects without compiling the application. However, this approach requires that you manually compile each module before compiling the application. One way to do this is to compile all open projects in Flash Builder at one time.

If you compile modules separately from the main application, be sure to include or exclude debugging information, based on whether you want to debug your application and modules. For more information, see [“Debug modules in Flash Builder”](#) on page 222.

The Flash Builder workflow is designed around associating modules with a single application. If you want to use modules across multiple applications, consider encapsulating the code in a library component or class and including that in a simple module for each application. Modules are not intended to be used for cross-application code reuse; that is for libraries.

## Use multiple projects for modules

When you set up your project's architecture, you can decide to include modules in your application's project, create a separate project for each module, or create a separate project for all modules.

### Use one project for each module

Using one project for each module has the following benefits:

- Module projects can be located anywhere in the workspace.
- Module projects can have their own compiler settings, such as a custom library path.

Module projects can use the `load-externs` compiler option to remove overlapping dependencies.

Using one project for each module has the following drawbacks:

- Many projects use more memory.
- Many projects in a single workspace can make the workspace crowded.
- By default, when you compile the application, not all module projects are compiled even if they have changed.
- To optimize your module's file size, manually apply the `load-externs` and `link-report` compiler options.

## Use one project for all modules

A related approach is to use a single project for all modules, while keeping the application in its own separate project. This has some of the drawbacks of using a single project for both the application and the modules, but it has many of the same benefits as using a separate project for each module.

Using one project for all modules has the following benefits:

- The module project can be located anywhere in the workspace.
- You can compile just the modules or just the application, without having to recompile both at the same time.
- The module project can use the `load-externs` compiler option to remove overlapping dependencies.

Using one module project for all modules has the following drawbacks:

- All of the modules in the module project must use the same compiler settings, such as the library path.
- By default, when you compile the application, the module project is not compiled even if the module project has changed.
- To optimize your module's file size, manually apply the `load-externs` and `link-report` compiler options.

## Create projects for modules

When creating a separate project for modules, you change the module project's output folder to a directory that the application uses. You also suppress the generation of wrapper files.

### Create a separate project for modules in Flash Builder

- 1 Create a main project.
- 2 Create a project for your module or modules.
- 3 From the context menu for the module's project, select Properties. The Properties dialog box appears.
- 4 Select the Flex Build Path option.
- 5 Change the Output Folder to point to the MainProject modules directory. For example, change it to the following:  

```
${DOCUMENTS}\MainProject\assets
```

This redirects the output of your module's compilation to your application project's (MainProject) assets directory. In your main application, you can point the ModuleLoader `url` property to the SWF files in the assets directory. The value of this property is relative to the output folder.
- 6 Click OK to save your changes.
- 7 Open the project properties again and select the Flex Compiler option.
- 8 Deselect the Generate HTML Wrapper File option. This prevents the module's project from generating the HTML wrapper files. You typically use these files only for the application. For modules, they are not necessary.
- 9 Click OK to apply the changes.

## Compile projects for modules

Compiling multiple projects in Flash Builder is a common operation. First you choose the order in which you want to compile the projects. Then you compile all the projects at the same time.

### Compile all projects at the same time in Flash Builder

- ❖ From the main menu, select Project > Build All.

Flex builds all projects in the workspace. The application files are added to each project's output folder. If you haven't already chosen to save files automatically before a build begins, you are prompted to save the files.

If you want to change the build order, use the Build Order dialog box. Changing the build order is not always necessary. Projects that use modules need to be compiled only by the time the main project application runs, not as it is compiled. In most cases, the default build order is adequate.

However, if you want to eliminate overlapping dependencies, you might need to change the build order so that the main application is compiled first. At that time, you use the `link-report` compiler option to generate the linker report. When you compile the modules, you use the `load-externs` compiler option to use the linker report that is generated by the shell application. For more information on reducing module size, see [“Optimize modules in Flash Builder”](#) on page 222.

### Change the build order of the projects

- 1 Open the Preferences dialog and select General > Workspace > Build Order.

The Build Order dialog box appears.

- 2 Deselect the Use Default Build Order checkbox.
- 3 Use the Up and Down buttons to reorder the projects in the Project Build Order list. You can also use the Remove Project button to remove projects that are not part of your main application or that are not modules used by the application. The removed project is built, but only after all the projects in the build order list are built.
- 4 Click OK.
- 5 Modify the build order as needed and then click OK.

If you create dependencies between separate projects in the workspace, the compiler automatically determines the order in which the projects are built, so these dependencies are resolved properly.

When you use a separate project for each module, you can compile a single module at a time. This can save time over compiling all projects at once, or over compiling a single project that contains all module and application files.

### Compile a single module's project

- 1 Right-click the module's MXML file in the module's project.
- 2 Select Run Application. The Player or browser window tries to run the module after it is compiled. You can close the Player or browser window and ignore any error messages that appear at runtime. Modules are not meant to be run in the Player or in a browser directly.

## Add modules to your project

In some cases, you use modules that are not in your main application's project. You can have the module in a separate project so that you can use custom configuration options, or you may want to share the module across multiple applications. Add the module's source code to your application's source path and then add the module to the application's module list before you can use it in your project.

### Add an already-compiled module to your project

- 1 Select the main application in the Flex Package Explorer.
- 2 Select Project > Properties > Flex Build Path to add the module's source to the application project's source path.

- 3 Click the Add Folder button and browse to the module's source path. Click OK to select the module. Do so for each external module that you add to your application's project.
- 4 Click OK again to save your changes.
- 5 Click Project > Properties > Flex Modules to add the module to the application's module list. The Flex Modules dialog box lists all modules that have been added to the current project or that are in the current project. When you first create a project, this dialog box is empty.
- 6 Click the Add button. The Add Module dialog box appears.
- 7 Use the Browse button or enter the location of the module's MXML file in the Source field. All modules that are in the project's source path are available to add by using this dialog box.
- 8 Select one of the options under Module SWF Size to enable or disable module optimization. If you select Optimize for Application, Flash Builder compiles the module against the selected application and excludes all classes that are defined in the main application. These can include framework classes or custom classes. When you select this option, you cannot use the same module in another application, because the list of excluded classes might be different. For more information, see ["Optimize modules in Flash Builder"](#) on page 222.
- 9 Click OK to save your changes. Flash Builder adds the module to the list of available modules in your application's project.

## Optimize modules in Flash Builder

In Flash Builder, you typically select a single application to optimize the module against when you first create the module or add it to a project. If you later decide to change the application that you optimize the module against, or if do not want to optimize the module, you can edit the module's properties within the project. For more information, see Reducing module size.

This procedure assumes that the module and application are in the same Flash Builder project. If the modules are in a separate project, manually add the `load-externs` and `link-report` compiler options.

- 1 Right-click the application's project in the Flex Package Explorer and select Properties. The project's Properties dialog box appears.
- 2 In the left pane, select Flex Modules.
- 3 From the list of modules, select the module and then click the Edit button. The Edit Module dialog box appears.
- 4 To remove optimization, select Do Not Optimize under Module SWF Size.
- 5 To optimize the module for a different application, select the new application from the Optimize for Application pop-up menu.
- 6 Click OK.

To further optimize a module's file size, you can remove debugging information. If you build a module in Flash Builder, debugging information is included in the module by default. By removing debugging information, you can further reduce the size of the module. For instructions on how to remove debugging information from modules, see ["Debug modules in Flash Builder"](#) on page 222.

## Debug modules in Flash Builder

To debug modules and applications, include debug information in the SWF files when they are compiled. To do this in Flash Builder, run the application, because debug information is included by default. On the command line, you set the `debug` compiler option to `true`. The default is `true`, but if you disabled it in a configuration file, make sure to override it.

By default, Flash Builder builds a single SWF file that includes debug symbols, so both Run and Debug work when you execute an application that uses modules in Flash Builder. However, including debug symbols in a module's SWF file makes the SWF file larger. To exclude debug symbols before deployment, disable debugging for the application's modules. To do so, export the release version of the modules by selecting Project > Export Release Build.

To exclude debugging information from SWF files in Flash Builder, you can either set the `debug` option to `false` in the Additional Compiler Arguments text box, or you can output the SWF files by using the Export Release Build feature, which generates non-debug SWF files. This includes the modules, if those modules are in the current project.

If you create a separate project for your modules, you can enable or disable debugging for the entire project, without changing the settings of your main application.

When you want to debug a module, you must also debug the application that loads the module. The Flex debugger does not connect to an application that does not contain debug information, even if the modules that the application loads contain that information. In other words, you cannot exclude debug information from the application if you want to debug the module that the application loads.

When you are using modules in an AIR application, the module SWF must be located in the same directory as the main application SWF or one of its subdirectories.

## Integrate Flex with HTML applications

You use the Create Ajax Bridge feature to generate JavaScript code and an HTML wrapper file that let you easily use an application built in Flex from JavaScript in an HTML page. This feature works with the Flex Ajax Bridge JavaScript library, which lets you expose an application to scripting in the web browser. The generated JavaScript code is lightweight, as it is intended to expose the functionality that the Flex Ajax Bridge already provides. For more information about the Flex Ajax Bridge, see Flex Ajax Bridge.

The Create Ajax Bridge feature generates JavaScript proxy code that is specific to the application APIs that you want to call from JavaScript. You can generate code for any MXML application or ActionScript class in a Flash Builder project.

For MXML application files, you can generate code for any or all of the following items in the MXML code:

- List of inherited elements, which can expand non-recursively
- Public properties, including tags with `id` properties
- Public constants
- Public functions, including classes defined in line

For ActionScript classes, you can generate code for any or all of the following items:

- List of inherited elements
- Public properties; for each property, a get and set method is displayed
- Public constants
- Public methods

In a directory that you specify, the Create Ajax Bridge feature generates `*.js` and `*.html` files that correspond to the MXML applications and ActionScript classes that you select for generation. It places a copy of the Flex Ajax Bridge library (`fabridge.js`) in a subdirectory of the code generation directory. This feature also generates MXML helper files in the project's `src` directory; these files are used to complete the JavaScript code generation.

## Generate Ajax Bridge code

- 1 Right-click a project in the Flex Package Explorer and select Create Ajax Bridge.
- 2 In the Create Ajax Bridge dialog box, select the MXML applications and ActionScript classes for which you want to generate JavaScript code. You can select the top-level check box to include the entire object, or you can select specific members.
- 3 Specify the directory in which to generate proxy classes.
- 4 Click OK to generate the code. The following example shows a .js file generated for an application that displays images:

```
*
* You should keep your JavaScript code inside this file as light as possible,
* and keep the body of your Ajax application in separate *.js files.
*
* Do make a backup of your changes before regenerating this file. (Ajax Bridge
* display a warning message.)
*
* For help in using this file, refer to the built-in documentation in the Ajax Bridge
application.
*
*/

/**
* Class "DisplayShelfList"
* Fully qualified class name: "DisplayShelfList"
*/
function DisplayShelfList(obj) {
    if (arguments.length > 0) {
        this.obj = arguments[0];
    } else {
        this.obj = FABridge["b_DisplayShelfList"].
            create("DisplayShelfList");
    }
}

// CLASS BODY
// Selected class properties and methods
DisplayShelfList.prototype = {

    // Fields form class "DisplayShelfList" (translated to getters/setters):

    // Methods form class "DisplayShelfList":

    getAngle : function() {
        return this.obj.getAngle();
    },

    setAngle : function(argNumber) {
        this.obj.setAngle(argNumber);
    },

    setCurrentPosition : function(argNumber) {
        this.obj.setCurrentPosition(argNumber);
    },
}
```



```
        setSelectedIndex : function(argNumber) {
            this.obj.setSelectedIndex(argNumber);
        },

        setPercentHeight : function(argNumber) {
            this.obj.setPercentHeight(argNumber);
        },

        setPercentWidth : function(argNumber) {
            this.obj.setPercentWidth(argNumber);
        },

        DisplayShelfList : function() {
            return this.obj.DisplayShelfList();
        },

        setFirst : function(argNumber) {
            this.obj.setFirst(argNumber);
        },

        setFormat : function(argString) {
            this.obj.setFormat(argString);
        },

        setLast : function(argNumber) {
            this.obj.setLast(argNumber);
        }
    }

    /**
     * Listen for the instantiation of the Flex application over the bridge.
     */
    FABridge.addInitializationCallback("b_DisplayShelfList", DisplayShelfListReady);

    /**
     * Hook here all of the code that must run as soon as the DisplayShelfList class
     * finishes its instantiation over the bridge.
     *
     * For basic tasks, such as running a Flex method on the click of a JavaScript
```

```

    * button, chances are that both Ajax and Flex have loaded before the
    * user actually clicks the button.
    *
    * However, using DisplayShelfListReady() is the safest way, because it lets
    * Ajax know that involved Flex classes are available for use.
    */
function DisplayShelfListReady() {

    // Initialize the root object. This represents the actual
    // DisplayShelfListHelper.mxml Flex application.
    b_DisplayShelfList_root = FABridge["b_DisplayShelfList"].root();

    // YOUR CODE HERE
    // var DisplayShelfListObj = new DisplayShelfList();
    // Example:
    // var myVar = DisplayShelfListObj.getAngle ();
    // b_DisplayShelfList_root.addChild(DisplayShelfListObj);

}

```

- 5 Edit the generated .js files. In the xxxReady() function of the generated .js files, add the code that must run as soon as the corresponding class finishes its instantiation over the Ajax Bridge. Depending on your application, default code can be generated in this method. The bold code in the following example shows custom initialization code for the sample image application:

```

...
function DisplayShelfListReady() {

    // Initialize the root object. This represents the actual
    // DisplayShelfListHelper.mxml Flex application.
    b_DisplayShelfList_root = FABridge["b_DisplayShelfList"].root();

    // Create a new object.
    DisplayShelfListObj = new DisplayShelfList();
    // Make it as big as the application.
    DisplayShelfListObj.setPercentWidth(100);
    DisplayShelfListObj.setPercentHeight(100);
    //Set specific attributes.
    DisplayShelfListObj.setFirst(1);
    DisplayShelfListObj.setLast(49);
    DisplayShelfListObj.setFormat("./photos400/photo%02d.jpg");
    //Add the object to the DisplayList hierarchy.
    b_DisplayShelfList_root.addChild(DisplayShelfListObj.obj);

}

```

- 6 Edit the generated .html files. In the part of the HTML pages that contains the text “Description text goes here,” replace the text with the HTML code that you want to use to access the application from the HTML page. For example, this code adds buttons to control the sample image application:

```
<h2>Test controls</h2>
<ul>
<li><input type="button" onclick="DisplayShelfListObj.setCurrentPosition(0) "
value="Go to first item"/>
</li>
<li><input type="button" onclick="DisplayShelfListObj.setCurrentPosition(3) "
value="Go to fourth item"/>
</li>
<li><input type="button" onclick="DisplayShelfListObj.setSelectedIndex(0) "
value="Go to first item (with animation)"/>
</li>
<li><input type="button" onclick="DisplayShelfListObj.setSelectedIndex(3) "
value="Go to fourth item (with animation)"/>
</li>
<li><input type="button" onclick="alert(DisplayShelfListObj.getAngle()) "
value="Get photo angle"/>
</li>
<li><input type="button" onclick="DisplayShelfListObj.setAngle(15);"
value="Set photo angle to 15&deg;"/>
</li>
</ul>
```

- 7 Open the HTML page in a web browser to run the application.

# Chapter 9: Using Flash Builder with Flash Professional

Use Flash Professional projects to access code in FLA or XFL files created with Flash Professional CS5.5. Typically, you create a project and files in Flash Professional, then you create a corresponding project in Flash Builder to edit and debug the files. This feature allows Flash Professional developers to use the editing and debugging environment available with Flash Builder.

**Note:** *The features of Flash Professional projects are only available in Flash Builder if you have installed Flash Professional CS5.5.*

## Create a Flash Professional project

To create a Flash Professional project in Flash Builder, you must already have started the project in Flash Professional. That is, the FLA or XFL file must exist.

- 1 Select File > New Flash Professional Project.
- 2 Navigate to the target FLA or XFL file for the project.  
The name of the file becomes the name of the project.

- 3 Specify a project location:  
You can use either the default project location in the workspace or navigate to a new project location.

- 4 Click Finish.  
Flash Builder opens the new project in the Package Explorer. The folder containing the target FLA file is accessible. The selected FLA file becomes the target file in the project. ActionScript files that are dependent to the target files are available for editing.

If Flash Professional is not running, Flash Professional starts.

## What you can do in a Flash Professional project

You can do the following with source files in a Flash Professional project:

- Edit the ActionScript files that are dependent to the target FLA file.
- Debug the file in the Flash Builder debugger or Flash Professional Debugger.

When editing the files in Flash Builder, you can set breakpoints in the project's ActionScript files.

To debug in Flash Builder, select Run > Debug *file* or click the Debug button from the toolbar.

To debug the file in Flash Professional, select Run > Debug Movie or click the Debug Movie button in Flash Builder. The Flash Professional debugger recognizes breakpoints set in Flash Builder.

- Publish the file in Flash Professional CS5.5:  
Select Project > Publish Movie or click the Publish in Flash Professional button from the toolbar.

- Run the file in either Flash Builder or Flash Professional:

To run the file in Flash Builder, select Run > Run *file* or click the Run button from the toolbar.

To run the file in Flash Professional, select Run > Test Movie or click the Test Movie button from the toolbar.

## Set project properties for Flash Professional projects

- 1 Select Project > Properties > Flash Professional.
- 2 Select Add to add additional files to the project.

A project can have only one target FLA or XFL file as the default target file. Use the Set as Default button to specify the default target file for the project.

- 3 Click OK.

## Create and edit Flash components

Adobe Flash® Professional CS5.5 creates applications compatible with Adobe Flash Player 10 or later. Applications built with Flex also support Flash Player 10 or later, which means that you can import assets from Flash Professional CS5.5 to use in your applications. You can create controls, containers, skins, and other assets in Flash Professional CS5.5, and then import those assets into your application as SWC files. For information on creating components using Flash Professional CS5.5, see [Flex Skin Design Extensions and Flex Component Kit for Flash Professional](#).

In the Design View of the Flash Builder editor, you can insert a new Flash component by adding a placeholder for a Flash component or container. You can invoke Flash Professional CS5.5 from Flash Builder to create the component or container. You can also invoke Flash Professional CS5.5 to edit previously created Flash components.

If your application contains an SWFLoader component to launch the SWF files, you can launch Flash Professional CS5.5 to create or edit the associated FLA and SWF files.

### Create a Flash component or a Flash container

- 1 In Design mode of the Flash Builder editor, make sure that the Components View is visible.
- 2 From the Custom folder in the Components View, drag either a New Flash Component or a New Flash Container to the design area.

You can resize or position the component or container.

- 3 From either the context menu or the Standard View of the File Properties window, select Create in Adobe Flash.

**Note:** You can also double-click the component in Design View to create the item in Adobe Flash.

- 4 In the dialog that opens, specify names for the class and the SWC file, then click Create to open Adobe Flash Professional CS5.5.
- 5 In Flash Professional CS5.5, edit the component or container. Select Done when you are complete to return to Flash Builder.

## Edit a Flash component or Flash container

This procedure assumes that you have previously inserted a Flash component or container into Flash Builder.

- 1 In Design mode of the Flash Builder editor, select the Flash component or container you want to edit.
- 2 From either the context menu or the Standard View of the Flex Properties window, select Edit in Adobe Flash.  
*Note:* You can also double-click the component in Design View to edit the item in Adobe Flash.
- 3 In Flash Professional CS5.5, edit the component or container. Select Done when you are complete to return to Flash Builder.

## Create or edit a SWF file associated with a SWFLoader component

This procedure assumes that your application contains an SWFLoader component.

- 1 In Design mode of the Flash Builder editor, select the SWFLoader component.
- 2 From either the context menu for the component or the Standard View of the Flex Properties window, launch Flash Professional CS5.5 by doing one of the following:
  - Select Create in Adobe Flash to create a SWF file associated with the SWFLoader component.
  - Select Edit in Adobe Flash to edit the SWF file associated with the SWFLoader component.
- 3 After you have finished editing the file, select Done to return to Flash Builder.

## Import assets from Flash CS3 or later

You can add Flash components that were created in Adobe Flash CS3 Professional to your user interface.

*Note:* Before you can create Flex components in Flash CS3, install the Flex Component Kit for Flash. For more information, see the article [Importing Adobe Flash CS Professional assets into Adobe Flex](#).

- 1 Ensure that the Flash component is saved in the library path of the current project.  
  
The library path specifies the location of one or more SWC files that the application links to at compile time. The path is defined in the Flex compiler settings for the project. In new projects the `libs` folder is on the library path by default.  
  
To set or learn the library path, select the project in the Package Explorer and then select Project > Properties. In the Properties dialog box, select the Flex Build Path category, and then click the Library Path tab. For more information, see [“Build projects manually”](#) on page 100.
- 2 Open an MXML file and add a Flash component in one of the following ways:
  - In the MXML editor’s Design mode, expand the Custom category of the Components view and drag the Flash component into the MXML file. For documents that are already open, click the Refresh button (the green circling arrows icon) to display the component after you insert it.
  - In Source mode, enter the component tag and then use Content Assist to quickly complete the tag.

## Manage Flash Player security

Flash Player does not allow an application to receive data from a domain other than the domain from which it was loaded, unless it has been given explicit permission. If you load your application SWF file from `http://mydomain.com`, it cannot load data from `http://yourdomain.com`. This security sandbox prevents malicious use of Flash Player capabilities. (JavaScript uses a similar security model to prevent malicious use of JavaScript.)

To access data from an application that is built in Flex, you have three choices:

- Add a cross-domain policy file to the root of the web server that hosts the data service.
- Place your application SWF file on the same server that hosts the data service.
- On the same server that contains your application SWF file, create a proxy that calls a data service hosted on another server.

### Use cross-domain policy files

A cross-domain policy file is a simple XML file that gives Flash Player permission to access data from a domain other than the domain on which the application resides. Without this policy file, the user is prompted to grant access permission through a dialog box. You want to avoid this situation.

The cross-domain policy file (named `crossdomain.xml`) is placed in the root of the server (or servers) containing the data that you want to access. The following example shows a cross-domain policy file:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="www.yourdomain.com" />
</cross-domain-policy>
```

For more information about configuring cross-domain policy files, see [Security](#).

### Set up a proxy to access remote data

Another option for managing Flash Player security (aside from using a cross-domain policy file) is to use a proxy. ADEP Data Services and BlazeDS provide a complete proxy management system for applications built in Flex. You can also create a simple proxy service by using a web scripting language such as ColdFusion, JSP, PHP, or ASP.

The proxy service processes requests from the application to the remote service and responses from the remote service back to the application (Flash Player).



*When developing your applications, a common technique is to host the proxy on your local computer. To do so, run a web server and scripting language on your local development computer.*

### Set up Flash Builder to use a proxy for accessing remote data

After you have set up a proxy to access data from a remote service, you place the application files in the same domain as the proxy. In Flash Builder, you can modify both the project build settings and the launch configuration to manage the use of a proxy.

If you use Flash Builder to compile your applications and the proxy server is also set up on your local development computer, you can modify the project build settings to automatically copy the compiled application files to the appropriate location on your web server.

### Modify the project build path

- 1 In the Flex Package Explorer, select a project.
- 2 Right-click and select Properties. The Project Properties dialog box appears.
- 3 Select the Flex Build Path properties page.
- 4 Change the existing output folder by entering a new path or by browsing to the appropriate folder of your web server (for example, C:\inetpub\wwwroot\myApp\).
- 5 Click OK.

To run and debug the application from the web server, modify the project's launch configuration.

**Note:** *If your proxy server is not your local machine, copy the contents of the bin directory to your server before running or debugging your program.*

### Modify the launch configuration

- 1 With the project's main application file open in Flash Builder, right-click in the editor and select Run As > Run Configurations. The Create, Manage, and Run Configurations dialog box appears.
- 2 From the list of configurations, select the project's launch configuration.
- 3 (Optional) On the Main tab, deselect the Use Defaults option to modify the URL or path to launch.
- 4 In the Debug text box, enter the URL or path to the debug version of the application.
- 5 In the Profile text box, enter the URL or path to the profiler version of the application.
- 6 In the Run text box, enter the URL or path to the main application file.
- 7 Click Apply and then click Close.



# Chapter 10: Using Flash Builder with Flash Catalyst

## Workflows between Flash Catalyst and Flash Builder

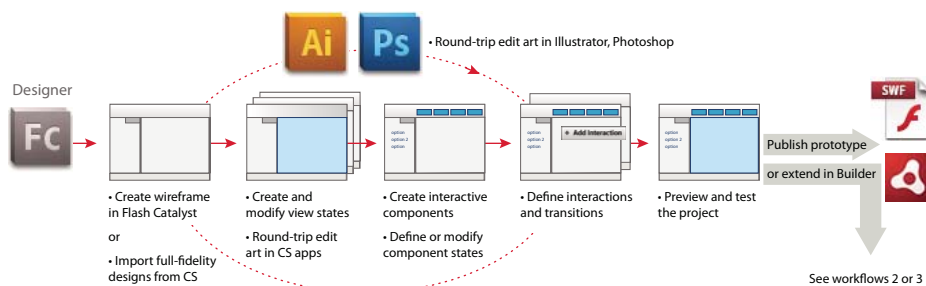
Flash Catalyst CS5.5 and Flash Builder 4.5 or a higher version enable designers and developers to collaborate and work in-tandem on projects. Here are overviews of three common workflows.



Watch this video by Adobe Community Professional, Steven Peeters, for an [overview of the workflows between Flash Catalyst and Flash Builder](#).

### Wireframe and prototype workflow within Flash Catalyst

This workflow outlines a common set of steps designers will often complete within Flash Catalyst and other CS apps to produce wireframes and rapidly prototype applications. With the involvement of a developer, Flash Catalyst projects can be extended further using Flash Builder—for example to connect to a database or web services. Workflows 2 and 3 show how files and parts of applications, like skins and components, can be exchanged between designers and developers.




**Plan the application** Start with a detailed project specification. This specification describes each page or screen, the artwork and interactive components on each page, user navigation, and the different states of each component. The specification also describes any data list components used to retrieve and display external data.

**Create wireframe or import a full-fidelity design** Using the graphical tools in Flash Catalyst, create a wireframe of the application's layout, or import a design comp from Adobe Illustrator, Photoshop, or Fireworks.

**Create or acquire additional artwork, video, and sound** Create additional artwork, video, and sound for the application.

**Bring in artwork, video, and sound** Bring the layered artwork into Flash Catalyst. You can also import individual graphic files or create simple graphics using the built-in vector drawing tools. Import additional assets, such as video, sound, and SWF content. For data-centric components, such as a data list, import a representative sample of the data (text or images). For more information, see [Importing artwork](#).

 After importing or creating artwork in Flash Catalyst, you can launch and edit artwork in Illustrator or Photoshop, and then return the edited artwork to Flash Catalyst. Roundtrip editing extends the graphic drawing and editing capabilities of Flash Catalyst and improves the iterative design process. For more information, see [roundtrip editing](#).

**Create and modify view states** Create states according to the project specification. For more information, see [Types of states](#).

**Create interactive components and define component states** Convert artwork to ready-made components (buttons, scroll bars, data lists, and so on). Use the Common Library panel to quickly add common components with a generic appearance. Create custom components for behaviors that you can't capture with the built-in components. For more information, see [What is a component?](#)

For data-centric applications, use design-time data to design data list components. Design-time data allows the use of dummy content, such as sample database records or bitmap images, without having to actually connect to a back-end system. A Flex developer can replace the design-time data with real data from a database or web service. For more information on using Design-time data, see [Data lists and scrolling panels](#).

**Create or modify component states** Components can have multiple states, such as the Up, Over, Down, and Disabled states of a button. Create or modify the different states of each interactive component, according to your project specification.

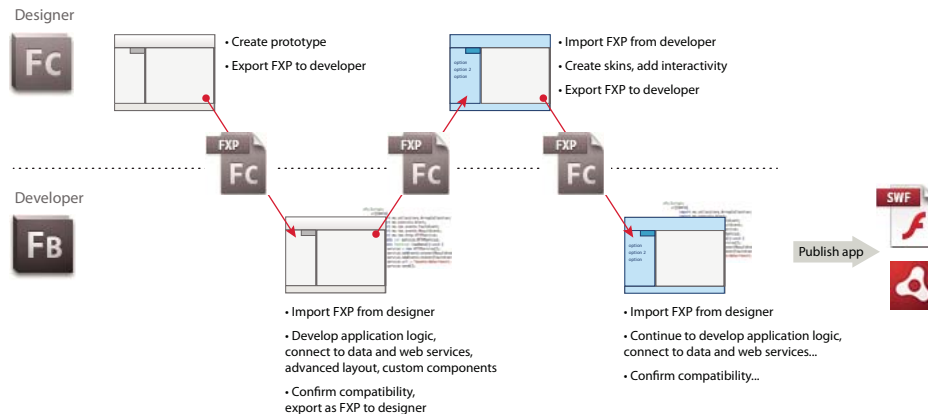
***Note:** The steps of creating page states and creating interactive components are interchangeable. Some designers prefer to create all interactive components first, and then add those components to pages and states.*

**Define interactions and transitions** Add interactions that define what happens as users interact with the application. For example, you can add interactions that transition from one page or component state to another when a user clicks a button. You can also add interactions that play animation, control video playback, or open a URL. Use the Timelines panel to add and modify smooth animated transitions between pages and component states. For more information on interactions, see [Create navigation and behavior with interactions](#). For more information on transitions, see [Animations](#).

**Test the project** Test the project frequently during development to ensure that your interactions are working properly. See [Previewing your project in a web browser](#).

## Small team workflow between Flash Catalyst and Flash Builder

For simpler projects consisting of a 1-2 person team, FXP files can be used to exchange a complete Flex project between a designer working in Flash Catalyst and a developer working in Flash Builder. The FXP format is an archive format that includes project folders, files, and metadata about the project.



An FXP file contains all of the assets necessary for working with the project in Flash Builder. The developer can provide the designer with skins, basic layout, motion, and basic interactivity for the project through the FXP file.

Using Flash Builder, the developer can add business logic, data connectivity, advanced layout, and custom skinnable component behaviors. The developer can send the FXP file back to the designer to further edit the skins, add interactivity and motion as the project proceeds. All the while, the developer has to ensure that Flash Catalyst compatibility is maintained in the project. For more information on Flash Catalyst compatibility in Flash Builder, see [Guidelines to create a Flash Catalyst compatible project](#).

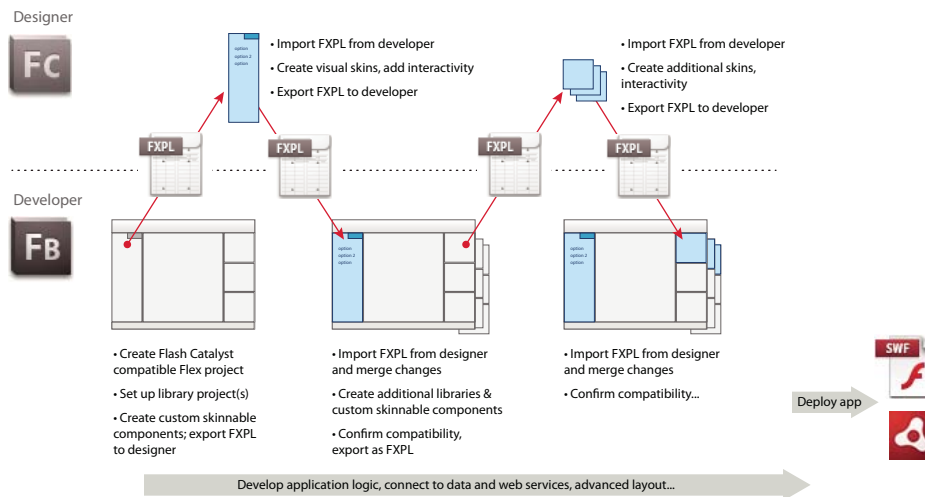
If additional design work is required, the designer can reopen the FXP file in Flash Catalyst and modify the skins, basic layout, and basic interactivity of the project. The designer can then resave the FXP and allow the developer to incorporate those changes into the project in Flash Builder.

**Note:** Some components and other aspects of the project may be uneditable in Flash Catalyst once they have been modified in Flash Builder.

Once complete, the project can be finalized and published in Flash Builder.

## Multi-person workflow between Flash Builder and Flash Catalyst

For more complex projects, developers can let designers edit parts of the design in Flash Catalyst, and move only those parts to a Flex library project. The developer can then reference the library project from the main project, and export the library project as an FXPL file. The designer can import the FXPL file in Flash Catalyst, make changes as needed, and export the FXPL file for handoff back to the developer.



The project is separated into a main project and subsidiary libraries of components. Only the libraries are edited in Flash Catalyst. To create a library in Flash Builder, see “[Create Flex library projects](#)” on page 213. For more information about working with libraries in Flash Catalyst, see “[Use Flex library projects](#)” on page 212.

An FXPL file is exported from the Project Library and contains information about the component that a developer requires to work with that component in Flash Builder. In Flash Builder, the developer integrates the FXPL as a part of a larger project.

The developer can export FXPL files back to the designer as custom skinnable components that can be edited in Flash Catalyst. For more information on creating custom skinnable components in Flash Builder, see “[Create an ActionScript skinnable component](#)” on page 38.

You can then import the design assets into Flash Builder as a new project and compare and merge changes using the Flash Builder tools. For more information, see “[Passing files between Flash Builder and Flash Catalyst](#)” on page 239

If additional design work is required, you can pass the FXPL files back and forth between Flash Builder and Flash Catalyst. All the while ensuring that Flash Catalyst compatibility is maintained in the project. For more information on maintaining Flash Catalyst compatibility in the project, see “[Ensure Flash Catalyst compatibility](#)” on page 238.

You can then use the tools in Flash Builder to create a fully functional application. You can define data services that fetch the data at runtime, and then bind the returned data to visual components in your project. For example, when you click a button, you can call an HTTPService component. The component can use remote procedure calls to interact with server environments, such as ColdFusion or PHP, and provide data to the application. For more information, see [Building data-centric applications with Flash Builder](#).

# Flash Catalyst and Flash Builder integration

## Project structure for ensuring Flash Catalyst compatibility

Flash Catalyst CS5.5 supports a subset of features and components that are available in Flex 4.5 or a higher version. Therefore, it is important that you structure your project such that it works seamlessly in Flash Catalyst.

### Plan your project

When you plan your project, identify the visual parts of the project that a designer owns and the logical parts that a developer owns. Typically, a designer owns the visual objects, animations, and the basic layout of the project. A developer typically owns the architecture, functionality, and the application-level layout of the project.

Flash Catalyst and Flash Builder provide several tools that help you define a clear contract between design and development. Separating your project into a main project and Flash Catalyst compatible library projects clearly defines which parts of the project the designer can edit. Using skinnable components provides a clear separation between logic and visuals at the component level.

### Use library projects

Library projects let you separate the design and logic at the project level. Developers can use library projects to segregate parts of the project that need to be edited in Flash Catalyst.

Library projects also simplify the merge process because the developer does not usually edit the library project while the designer is working on it.

### Create a main project and a subsidiary Flash Catalyst-compatible library project

- 1 Create the main Flex project. For more information, see [“Flex projects”](#) on page 81.
- 2 Create a Flex library project.

Ensure that the project is Flash Catalyst-compatible by selecting Make Project Flash Catalyst Compatible in the New Flex Library Project wizard. For more information, see [“Create Flex library projects”](#) on page 213.

- 3 Add the library project to the build path of the main Flex project.

To do so, go to Project > Properties, and select Flex Build Path. Select the Library Tab, and click Add Project. For more information, see [“Modify a project build path”](#) on page 99.

- 4 Place the skinnable components and skins in the library project and the code files in the main project.

**Note:** The main project depends on the library project. Therefore, the library cannot contain code that depends on the main project and has to be self-contained.


### More Help topics

[“Passing files between Flash Builder and Flash Catalyst”](#) on page 239

### Use skinnable components

The skinning architecture in Flex lets you create extendable skinnable components that are compatible with Flash Catalyst. Creating skinnable components in Flash Builder and creating skins visually in Flash Catalyst lets you separate design and logic at the component level.

A skinnable component contains the logical part of a component, while the skin contains the visual assets and layout rules. Additionally, a skinnable component can indicate that its skin contains parts and states. The host component can programmatically control the parts and states of the skin. Restricting communication between the skinnable component and its skins to the parts and states lets the two parts to be highly independent and flexible. For more information on creating custom skinnable components in Flash Builder, see [Example: Creating a skinnable Spark component](#).

 *After you create a skinnable component definition in Flash Builder, create an initial sample skin. If you do so and import the project into Flash Catalyst, the skin shows up in the Components panel in Flash Catalyst. You can then easily edit the skin in Flash Catalyst. Creating a sample skin helps the designer understand the basic structure of the skin.*

## More Help topics

[Custom Skinnable Components](#)

[“Create an ActionScript skinnable component”](#) on page 38

[Skinning custom components using Flash Builder and Flash Catalyst \(video\)](#)

## Ensure Flash Catalyst compatibility

Like any visual coding tool, Flash Catalyst only supports a subset of Flex code. However, Flash Builder provides a compatibility checker that tells you exactly which parts of your code are editable by Flash Catalyst.

When you create a Flash Catalyst compatible project in Flash Builder, the Flash Catalyst compatibility checker (Project > Properties > Flash Catalyst) is automatically turned on. When you create a project in Flash Catalyst and import it into Flash Builder, it is automatically configured to be Flash Catalyst compatible. If you introduce any incompatibilities while editing the project, Flash Builder displays the compatibility errors in the Problems view indicating which parts of the code can be edited in Flash Catalyst.

You do not necessarily need to resolve all compatibility issues before opening a project in Flash Catalyst. The "Type" column of the Problems view displays the impact of each compatibility problem. Problems that prevent the project from opening in Flash Catalyst appear as warnings. Less severe problems appear as "info" notifications.

Incompatibility type	Description
Project incompatibility	The project cannot be opened in Flash Catalyst
File incompatibility	The file cannot be edited in Flash Catalyst. Any incompatibilities in the main application makes the project incompatible.
Skinnable component incompatibility	The component cannot be skinned in Flash Catalyst or the skin part cannot be assigned in Flash Catalyst.
Design-time data incompatibility	The list data is controlled by application code that is not editable in Flash Catalyst. You can edit the list appearance in Flash Catalyst.
Flash Catalyst compatibility warning	This warning does not reduce editability of the project in Flash Catalyst. It can, however, indicate an editing experience that is not seamless.

For more details about Flash Catalyst compatibility issues, see [Guidelines to create a Flash Catalyst compatible project](#).

## More Help topics

[Ensuring Compatibility in Flash Builder and Flash Catalyst Projects \(video\)](#)

## Passing files between Flash Builder and Flash Catalyst

You can use FXP files to exchange a complete Flex project or to exchange components, component skins, and assets between a designer and developer.

For complex projects, you can use a library project to structure your project's user interface as a series of skinnable components and skins. For more information, see [“Use Flex library projects”](#) on page 212.

You can then pass FXPL files back and forth between Flash Builder and Flash Catalyst to exchange only the component skins and design assets. To make sure that things are orderly, the designer can import the FXPL files into an empty project in Flash Catalyst, create visual skins, and add interactivity.

Use the following export and import workflows to pass the FXP and FXPL files back and forth between Flash Builder and Flash Catalyst.

### Pass files from Flash Builder to Flash Catalyst

- To export an FXP or FXPL file from Flash Builder to Flash Catalyst, select Project > Flash Catalyst > Export Flash Catalyst Project.
- To import the FXP or FXPL file from Flash Builder into Flash Catalyst, go to File > Import, and select Adobe FXG File (.fxg) or Library Package (.fxpl).

### Pass files from Flash Catalyst to Flash Builder

- To export an FXP or FXPL file from Flash Catalyst to Flash Builder, go to File > Export, and select either Adobe FXG File (.fxg) or Library Package (.fxpl)
- To import an FXP or FXPL file from Flash Catalyst into Flash Builder, select Project > Flash Catalyst > Import Flash Catalyst Project.

### Edit FXP files using the Edit in Flash Catalyst command

If Flash Builder and Flash Catalyst are both installed on the same computer, you can use the Edit in Flash Builder command to edit FXP files.

For more information, see [“Edit in Flash Catalyst command”](#) on page 240.

## Merge changes from Flash Catalyst to Flash Builder

When you import FXP or FXPL files from Flash Catalyst into Flash Builder, you can compare and merge differences between the two projects using the Compare Editor tool. You can also use a third-party merge tool of your choice.

When you create design assets in Flash Catalyst, you can bring the assets into Flash Builder by importing the FXP or FXPL file in the following ways.

- Import the design assets into a new project, and then compare and merge changes using the Flash Builder tools.
- Import contents of the library into the existing project.
- Overwrite the existing FXP file with the updated version only if no changes have been made to the project in the meantime.

**Note:** You cannot overwrite library projects (.fxpl).

To compare projects using the Compare Editor tool, select the projects by pressing the Ctrl key and clicking the projects to compare. Then, right-click the selected projects, and select Compare With > Each Other. The results are displayed in the Text Compare panel.

**Note:** While merging, you can see several changes in setting files, like `actionScriptProperties`, or output files, like `bin-debug` file, or other parts of the project. You can typically ignore these changes, and focus only on the changes in the source folder.

For more information on using the Compare Editor tool, see the Eclipse documentation.

## Edit in Flash Catalyst command

Use the Edit in Flash Catalyst command if you have both Flash Builder and Flash Catalyst installed on the same computer. You can launch Flash Catalyst from within Flash Builder and make any design changes directly to the project without passing the project files between Flash Builder and Flash Catalyst.

- 1 To launch Flash Catalyst directly from within Flash Builder, select **Project > Flash Catalyst > Edit Project In Flash Catalyst**.

You can also select **Flash Catalyst > Edit Project In Flash Catalyst** from the context menu for the project.

- 2 Change the application design, as required. When you edit the project in Flash Catalyst, the project is locked in Flash Builder. The project is locked to ensure that there are no conflicting changes.
- 3 Save changes and close the project in Flash Catalyst. You don't have to exit Flash Catalyst.
- 4 In Flash Builder, select **Project > Resume Working On Project in Flash Builder**. You can also select **Flash Catalyst > Resume Working On Project in Flash Builder** from the context menu for the project.

When you do so, you are prompted to save the changes made in Flash Catalyst. Saving the changes brings the newly saved project back into Flash Builder. In the background, the originally exported project version from Flash Builder is deleted after the newly saved project version is successfully imported from Flash Catalyst.

The design changes are added to the project, and the project is opened for editing in Flash Builder.



# Chapter 11: Customizing Flash Builder

Adobe® Flash® Builder™ is a plug-in to the Eclipse development platform. Specify Flash Builder preferences in addition to the general preference you specify for Eclipse. For some features, you specify preferences in both Eclipse and Flash Builder nodes of the Preference dialog. For example, when setting preferences for the Flash Builder debugger, you specify custom behavior under the Eclipse Run/Debug node as well as the Flash Builder > Debug node.

How you open the Preference dialog varies according to platform and whether you are using the stand-alone or plug-in version of Flash Builder.

## Specify Flash Builder preferences

- 1 Open the Preferences dialog:
  - (Windows) Select Windows > Preferences
  - (Macintosh, stand-alone) Select Windows > Preferences
  - (Macintosh, Plug-in) Select Eclipse > Preferences
- 2 Expand the Flash Builder node to view and specify user preferences.

## Adobe preferences

Set preferences for Adobe plug-in modules.

### RDS Configuration

Remote Development Server (RDS) configuration information applies to users of ADEP (Adobe Digital Enterprise Platform) Data Services or Adobe BlazeDS. RDS provides access to Data Services and BlazeDS destinations.

The default RDS configuration is a starting point for connecting to data services. Modify the default configuration to provide access to your server destination or database.

See the [ADEP Data Services](#) documentation for information on configuring RDS.

**Important:** Your RDS configuration has security implications. See the ADEP Data Services documentation for information on application server security implications when specifying an RDS configuration.

## Customize the workbench

You can customize the workbench to suit your individual development needs. For example, you can customize how items appear in the main toolbar, create keyboard shortcuts, or alter the fonts and colors of the user interface.

### Rearrange the main toolbar

Flash Builder lets you rearrange sections of the main toolbar. Sections of the main toolbar are divided by a space.

- 1 Ensure that the toolbar is unlocked. From the context menu for the toolbar, deselect Lock the Toolbars.
- 2 Move the pointer over the vertical line “handle” that is on the left side of the toolbar section you want to rearrange.

- 3 Click the handle and drag the section left, right, up, or down. Release the mouse button to place the section in the new location.



*To prevent accidental changes, lock the toolbar again from the toolbar context menu.*

## Change keyboard shortcuts

- 1 Open the Preferences dialog and select General > Keys.
- 2 In the View screen of the Keys dialog box, select the command you want to change.
- 3 In the Binding field, type the new keyboard shortcut you want to bind to the command.
- 4 In the When pop-up menu, select when you want the keyboard shortcut to be active.
- 5 Click Apply or OK.



*For a list of all keyboard shortcuts, go to Help > Key Assist*

## Change fonts and colors

By default, the workbench uses the fonts and colors that your computer's operating system provides. However, you can customize fonts and colors in a number of ways.

- 1 Open the Preferences dialog and select General > Appearance > Colors and Fonts.
- 2 Expand the Basic, CVS, Debug, Text Compare, or View and Editor Folders categories to locate and select the font and colors to change.

**Note:** You can also click Use System Font instead of Change to set the font to a reasonable value that the operating system chooses. For example, in Windows, selecting this option causes Flash Builder to use the font selected in the Windows Display Properties control panel.

- 3 Set the font and color preferences as desired.

## Control single and double-click behavior

You can control how the workbench responds to single and double-clicks.

- 1 Open the Preferences dialog and select General.
- 2 In the Open Mode section, make your selections and click OK.

## Set workbench preferences

You can set preferences for many aspects of the workbench. For example, you can specify that Flash Builder prompts you for the workspace you want to use at startup, you can select which editor to use when opening certain types of resources, and you can set various options for running and debugging your applications.

Your Flash Builder preferences apply to the current workspace only. You can, however, export your workbench preferences and then import them into another workspace. This may be helpful if you are using multiple workspaces, or if you want to share your workbench preferences with other members of your development team.

You can also set preferences for individual projects within a workspace. For example, you can set separate compiler or debugging options for each of your Flex projects.

### Set Flash Builder workbench preferences

- 1 Open the Preferences window.
- 2 Expand General and select any of the categories of workbench preferences and modify them as needed.
- 3 Click OK.

## Flash Builder preferences

### Flash Builder

#### Warn before upgrading old Flash Builder projects

Flash Builder updates projects that were created with an earlier version of Flash Builder. Flash Builder updates the project files and structure to conform to the current structure of Flash Builder projects. A converted project is no longer accessible under the earlier version of Flash Builder.

By default, Flash Builder warns you before making the conversion. Disable this option if you want conversions to happen silently.

### Data/Services

The following user preferences are available for Data/Services. These preferences apply to all projects in your Flash Builder developer environment.

You can override these preferences on a project basis. Select Project > Properties > Data/Services to override the preferences specified here.

#### Code Generator

Flash Builder provides a default code generation utility for generating access to data services. Using Flash Builder extensibility features, you can create your own code generation utilities and add them to Flash Builder as a plug-in. For more information, see [Flash Builder Extensibility API Reference](#).

Any code generation extensions you add as a plug-in to Flash Builder are available from the Code Generator combo box.

#### Enable Service Manager to use single service instance (singleton) during code generation

By default this option is not enabled. During code generation, each client application in a project creates their own instance of a data service.

If you want a single instance of a service that all client applications in the project share, enable this option.

This option is only available if you specify the default Flash Builder code generation utility.

### Debug

The following options for debugging in Flash Builder are automatically enabled. For other options that affect a debugging session see:

- Preferences > General > Web Browser
- Preferences > Run/Debug

### **Warn when trying to launch multiple debugging sessions**

Some platform/browser combinations do not allow concurrent debugging sessions. If you attempt to start a second debugging session, the original debugging session terminates or is disconnected.

Leave this option enabled for Flash Builder to warn you when you attempt to start a second debugging session.

### **Warn about Security Error after launch**

In some cases, a web browser issues a security error because it cannot read Flash Player's security trust file. In most cases, restarting the web browser corrects the security error.

Leave this option enabled if you want Flash Builder to warn you about this type of security error.

For more information, see the [TechNote for Flash Player security warning](#).

### **Automatically invoke getter functions**

When stepping through a debugging session, variables representing accessor (getter) functions are automatically evaluated. Typically, this behavior is useful during a debugging session.

Disable this option if you do not want to automatically evaluate variables representing accessor functions when stepping through a debugging session.

### **Stand-alone Flash Player version management**

When you launch or debug an application from Flash Builder, you can specify the Flash Player executable to use.

To do so, in the Preferences dialog box, select Flash Builder > Debug. Then, define the location of the Flash Player executable that Flash Builder uses to debug or launch your application.

## **Device Configurations**

Flash Builder uses device configurations when previewing device screen sizes in Design View or when launching applications on the desktop using the AIR Debug Launcher. See "[Manage launch configurations](#)" on page 111.

Flash Builder provides default configurations for Nexus One, Droid, Apple iPad, Apple iPhone, HTC, and Samsung Android phones. You cannot edit the default configurations.

## **Flash Builder Editors**

### **General**

Flash Builder provides user options for code folding, reporting problems as you type code, and using subword caret positioning.

### **Braces**

Flash Builder provides user options for indenting, inserting, and highlighting curly brackets representing blocks of code.

## Code assist

When using the MXML or ActionScript source editor, Flash Builder provides code hints to help you complete your code expressions. This assistance includes help in selecting recommended classes, properties, and events available.

- **Enable Auto-activation**

Disable Enable Auto-activation if you do not want code hints to automatically appear as you type. If you disable this option, you can access code hints by pressing Control+Spacebar.

- **Activate After**

Specify the time, in milliseconds, for code hints to appear after you begin typing. The default time is 100 milliseconds.

- **Use Additional Custom Triggers**

Apart from accessing code hints by pressing Control+Spacebar, you can specify your own trigger keys.

Type the keys that you want to use as a trigger for accessing code hints within MXML or ActionScript code. If you type even one key of the trigger, code hints are invoked. For example, if you specify the trigger keys as `ab`. Typing either `a` or `b` invokes code assist.

## More Help topics

[“Code Development Tools in Flash Builder”](#) on page 44

## Indentation

Flash Builder provides user options to indent using tabs instead of spaces, and automatically indent new lines on typing.

## General Flash Builder editor preferences

By default, Flash Builder provides code folding and automatic indentation of lines as you type or paste in the source editor. You can disable the default preferences for these features.

## Eclipse general editor preferences

Additional general editor preferences are available by selecting Preferences > General > Editors

## ActionScript code

When editing ActionScript files in the source editor, Flash Builder provides default features for wrapping and organizing code. By default, Flash Builder places all import statements at the head of an ActionScript file and removes import statements that are not referenced in the code body.

By default, ActionScript reference documentation, when available, appears with code hints or by hovering the pointer over a language element.

Each of these default features can be disabled.

As you enter ActionScript code into the Flash Builder editors, Content Assist prompts you with a list of options to complete your code expression. You can auto-complete your partially entered ActionScript code by using completion trigger keys. To do so, Flash Builder provides default trigger keys. You can choose to replace the default trigger keys with your own. You can also choose to insert the code on pressing the spacebar.

### Content Assist cycling

When entering ActionScript code, you press Control+Space multiple times to cycle through filters for displayed code hints.

You can specify which code hints to display, and the order for cycling through the code hints. To change the default setting, do the following:

- 1 Open the Preferences Dialog and select Flash Builder > Editors > ActionScript code > Content Assist Cycling.
- 2 Select a type of hint, and click Up or Down to change the order of cycling.

The first type of hint in the list is the initial code hint that is displayed.

### Indentation

By default, Flash Builder indents package contents, function declarations, and switch statements. Each of these default features can be disabled.

### More Help topics

[“Code Development Tools in Flash Builder”](#) on page 44

[“Content Assist”](#) on page 44

### Code templates

You can customize the predefined stub code that Flash Builder generates when you create methods or get and set accessor functions, generate event handlers, or override methods.

Flash Builder also provides predefined code templates that help you quickly create MXML, ActionScript, and CSS code. You can create and edit code templates. You can also import and export the code template files.

For more information about using, creating, and editing code templates, see [“Create and edit code templates”](#) on page 52.

### Design mode

Flash Builder provides the following user preferences for Design mode of the source editor:

- Automatically show design-related views

When switching to Design mode of the source editor, Flash Builder automatically opens related views, such as the Properties view, States view, Components view, and Appearance view.

Disable this preference if you want to customize which views are open in Design mode.

- Enable snapping

In Design mode, Flash Builder provides an invisible grid that automatically “snaps” in place components that are added or rearranged.

Disable this preference if you want more control of the placement of components.

- Render skins when opening a file

Design mode draws skinned components that are in the design area.

If you do not want to render the skins of components in Design mode, disable this preference.

- Collapse data binding expressions

In Design mode, data binding expressions typically do not accurately reflect the shape or size of the data to which the expressions are bound.

Enable this preference to allow Design mode of the editor to more accurately render components that are bound to data binding expressions.

- Always update references when changing IDs in Properties view

When you change a component ID in Properties view, it can affect code in your workspace that references that ID.

Enable this preference if you want Flash Builder to automatically update all references to component IDs when you edit a component ID in Properties view.

- Always open MXML files in code editor

When you open an MXML file in the Package Explorer, the file is always opened in the editor's Source mode.

Deselecting this preference opens the MXML file in the mode that the editor last opened it in. For example, if the MXML file was last opened in Design mode, and then closed. The next time you open that MXML file, it opens the file again in Design mode.

## MXML code

When editing MXML files in the source editor, Flash Builder provides default features for code completion. Each of these features can be disabled.

### Content Assist cycling

Flash Builder provides default behavior for the content assist feature for editing MXML files. When content assist is available, Flash Builder displays hints on available language elements for insertion into the code. You cycle through the elements displayed in the Content Assist window by pressing Control+Spacebar multiple times.

You can specify which code hints to display, and the order for cycling through the code hints. To change the default setting, do the following:

- 1 Open the Preferences Dialog and select Flash Builder > Editors > MXML code > Content Assist Cycling.
- 2 Specify which types of hints to display.
- 3 Select a type of hint, and click Up or Down to change the order of cycling.

The first type of hint in the list is the initial code hint that is displayed.

## More Help topics

[“Content Assist”](#) on page 44

[“Code Development Tools in Flash Builder”](#) on page 44

### Formatting

By default, Flash Builder formats auto-generated MXML code. For example, Flash Builder formats MXML code that is auto-generated when you add a component in the design view. Or when you use Flash Builder tools to connect to a data service and access data.

You can change the order and grouping of the attributes in the Formatting preferences dialog.

Code formatting is not applicable for hand-written code. To format a selected snippet of hand-written code, select Source > Apply Formatting (Ctrl+Shift+F). The code is then formatted, irrespective of whether you selected the Keep MXML Attributes Organized option or not.

## Indentation

By default, Flash Builder uses tabs for indenting code. You can change the default setting to use spaces. The default tab and indent sizes are each equal to four spaces.

Select ActionScript and MXML in the Indentation preferences dialog to preview indentation settings. You can also customize the indentation heuristics.

## Syntax coloring

Flash Builder provides default syntax coloring and text highlighting for ActionScript, CSS, and MXML files. You can override these default features. Expand the language node and select a language feature to override the default feature.

General Eclipse editor preferences are also available. See:

- Preferences > General > Text Editors
- Preferences > General > Colors and Fonts

## File Associations

You can select Flash Builder as the default application to open MXML and ActionScript files. Specifying a file association overrides operating system settings.

- 1 Open the Preferences dialog and select Flash Builder > File Associations.
- 2 Select ActionScript Files (\*.as) and MXML Files (\*.mxml) to set the file associations.

## File Exclusions

You can specify the file types that you do not want Flash Builder to copy to the output folder during project compilation.

- 1 Open the Preferences dialog, and select Flash Builder > File Exclusions.
- 2 You can add new filename extensions and filenames, or remove any existing ones from the list.
- 3 To add new filename extensions or filenames, click New, and specify a valid filename extension or filename.
- 4 To restore the default list of extensions at any point, click Restore Defaults.

## File Templates

Flash Builder uses file templates when creating new MXML, ActionScript, and CSS files. You can customize the templates Flash Builder uses to create these files. You can also import template files and export template files.

In the File Types area select a template for a file type. Click Edit to modify the template. Disable Automatically Indent New Files if you do not want to use the Flash Builder indenting preferences when creating files.

You can change the Flash Builder indentation preferences. Go to Preferences > Flash Builder > Indentation.

## More Help topics

[“Customize file templates”](#) on page 57

[“Indentation”](#) on page 245



## FlexUnit

### Default launch mode

By default, when you run a FlexUnit test, Flash Builder launches in Debug mode.

Use this preference to change the Flash Builder launch mode for FlexUnit tests.

Default launch configurations for Flash Builder are the following:

- Run mode  
By default, launches Flash perspective.
- Debug mode  
By default, launches Flash Debug perspective.
- Profile mode  
By default, launches Flash Profile perspective.

The Eclipse preferences for Launching and Perspectives configure launch modes. See:

- Preferences > Run/Debug > Launching
- Preferences > Run/Debug > Perspectives

### Custom application name

Specify the fully qualified name of the custom FlexUnit application file. If the custom FlexUnit application file is not within the selected Flex project, Flash Builder automatically generates a default FlexUnit application file.

### Port number

Specifies the port number to use to connect Flash Builder with the FlexUnit application on the mobile device. The port number is a hard-coded value 8765.

### Use an alternative framework

By default, the framework SWC files that are available with Flash Builder are added to the FlexUnit build path.

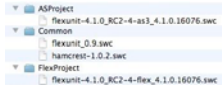
To specify an alternative FlexUnit framework, select Use Alternative Framework. Browse to the location of the framework directory that you want to select.

When you create a FlexUnit test case class or a FlexUnit test case suite for the selected project, Flash Builder automatically adds the SWC files in the alternative framework directory to the FlexUnit build path.

When you create an alternative FlexUnit framework directory, it is recommended to have the following directory structure for the SWC files:

- `flexunit-x.x-as3_x.x.swc` files in the "ASProject" folder. The SWC files in this folder are automatically added to the build path for an ActionScript project.
- `flexunit-x.x-flex_x.x.swc` files in the "FlexProject" folder. The SWC files in this folder are automatically added to the build path for a Flex project.

- Any other SWC files that your project requires or that your FlexUnit code has dependencies on (that are not already on the build path) in the "Common" folder. The SWC files in this folder are automatically added to the build path for both ActionScript and Flex projects.



If Flash Builder cannot detect the recommended directory structure, the selected alternative directory is added irrespective of whether it is a Flex project or an ActionScript project.



Adobe Community Professional, Paul Robertson, blogged about [creating a custom FlexUnit framework](#).

### More Help topics

[“FlexUnit test environment”](#) on page 160

[“Flash Builder perspectives and views”](#) on page 6

## Installed Flex SDKs

Flash Builder 4.6 installs the Flex 4.6 SDK and the Flex 3.6 SDK. Flash Builder uses the Flex 4.6 SDK for projects that do not specify an SDK.

You can add additional SDKs, remove SDKs, and also change which SDK is the default for projects that do not specify an SDK. Flash Builder prompts you for which SDK to use when importing projects created in an earlier version of Flash Builder. You can disable this prompt for importing projects.

The following types of SDKs are available:

**Free Adobe Flex SDK** Released versions of official Adobe products. These SDKs contain a mixture of open and closed source components.

**Open Source Flex SDK** For users who want a package that contains only open source code.

**Adobe Add-ons for Open Source Flex SDK** This package contains all of the items that are in the Adobe Flex SDK and not in the Open Source Flex SDK. Downloading this file allows you to bring the Open Source Flex SDK to parity with the Adobe Flex SDK.

## Adding a Flex SDK

- 1 Download a Flex SDK to a temporary location on your system:

Check the download pages for an explanation of the different build types, requirements, and license information.

- [Download Flex 4.6 SDKs](#)
- [Download Flex 4.5 SDKs](#).
- [Download Flex 4 SDKs](#).
- [Download Flex 3 SDKs](#).

**Note:** If you want to use a stable, tested release, download a milestone release build. A milestone release build has been declared a major release by the development team. It includes a signed version of the Flex framework RSLs (required to use RSL caching).

- 2 Flex SDKs downloads are ZIP files. Unpack the ZIP file to a new folder on your system.

It's a good idea to name the top-level folder with the version number of the SDK.

Typically, you place SDKS in the Flash Builder installation directory at the following location:

`<Flash Builder Install Directory>/sdks/`

- 3 From Flash Builder preferences, select Flash Builder > Installed SDKS.
- 4 Click Add and browse to the unpacked Flex SDK. Click Apply.
- 5 For each project that uses the Flex SDK, open this preference dialog and select the SDK to make it the default SDK for that project. Click OK.

## Network Monitor

The Network Monitor preferences page lists the ports on which the Network Monitor captures events and listens to http requests.

By default, the Network Monitor clears all monitoring entries upon startup. You can disable this preference.

You can also enable the following preferences:

- Suspend monitoring on start
- Ignore SSL Security Checks

Enabling this preference is useful when monitoring network traffic from a self-signed server.

## Profiler

By default, both memory and performance profiling are enabled.

- Memory profiling

You typically use memory profiling to examine how much memory each object or type of object is using in the application. Use the memory profiling data when to reduce the size of objects, reduce the number of objects that are created, or allow objects to be garbage collected by removing references to them.

Memory profiling uses much more memory than performance profiling, and can slow your application's performance.

- Performance profiling

You typically use performance profiling to find methods in your application that run slowly and that can be improved or optimized.

You can also specify the following preferences for the profiler:

- Connections

Specify the port number that Flash builder listens to when profiling an application. The default port number is 9999. You cannot set it to port 7935, which the Flash debugger uses.

- Exclusion filters

Defines the default packages that are excluded from profiler views.

- Inclusion Filters

Defines the default packages that are included in the profiler views. All other packages are excluded.

- Object References

Specifies the number of back-reference paths to instances of an object to display. The back-reference paths help determine if a path has a back-reference to the garbage collector (GC Root). An instance that is expected to be released, but has references to GC Root, indicates a memory leak.

By default, the profiler displays ten back reference paths. You can specify a different maximum to display or you can specify to show all back-reference paths.

- **Player/Browser**

You can specify which stand-alone Adobe Flash Player to use and which Web browser to use for profiling external applications. Use a debugger version of Flash Player to successfully profile the application.

When profiling external applications, by default, Flash Builder uses the following Flash Players:

- **URL to a SWF file**

Flash Builder launches the application's SWF file using the default browser for the system.

- **File system location for a SWF file**

Flash Builder opens the application with the default debugger version of the stand-alone Flash Player.

### More Help topics

[“Use the profiler”](#) on page 149

[“Profiler views”](#) on page 133

## Target platforms

For developing mobile applications for the Android platform, Flash Builder installs necessary portions of the Android SDK.

### Select a primary network interface

When you debug your application over the network, your host machine can be connected to multiple network interfaces simultaneously. However, you can select a primary network interface to use for debugging in the Android APK package or the debug iOS package.

### More Help topics

[Run and debug mobile applications](#)

## Extending Flash Builder

The Flash Builder Extensibility API allows you to extend Flash Builder to support custom components. Using the Flash Builder Extensibility API, you can extend Flash Builder Design view and Properties view so they properly handle custom components.



See [Flash Builder Design View extension FAQ](#) for answers to some of the most frequently asked questions about Design View extensions.

Additionally, you can extend Flash Builder support for services available from the Services wizard.

For more information, see the [Flash Builder Extensibility API Reference](#).