ADOBE® FLEX® BUILDER™ 3

USING ADOBE FLEX BUILDER 3

Fx

Adobe

Contents

Chapter 1: Learning Flex Builder

Adobe® Flex® Builder™ includes a variety of resources to help you learn the program quickly and become proficient in creating Flex applications. All the Flex and Flex Builder documentation is available in online help and Adobe PDF.

Topics

Using Flex and Flex Builder documentation

The Flex and Flex Builder documentation includes information for users who have a variety of backgrounds. This section helps you understand how to approach the documentation, depending on your background and what you want to accomplish with Flex and Flex Builder.

New users

- Start by going to the Flex Getting Started Experience site, which gives you an overview of essential Flex concepts and provides a series of tutorial lessons.
- Flex Builder is built on Eclipse (an open source IDE), so you need to know specific terms and concepts to be successful using it. For more information, see “Flex Builder Workbench Basics” on page 10 and especially “About the workbench” on page 10.

Web application designers

- Start by going to the Flex Getting Started Experience site, which gives you an overview of essential Flex concepts and provides a series of tutorial lessons.
- You can define a user interface entirely in code using the MXML editor in Source mode; however, in Design mode, Flex Builder contains design and layout tools that make designing Flex applications much easier. For more information, see “Flex Development perspective in Design mode” on page 16 and “Building a Flex User Interface” on page 64.

Experienced web application developers

- Start by going to the Flex Getting Started Experience site, which gives you an overview of essential Flex concepts and provides a series of tutorial lessons.
- For all the details of the Flex framework, including code samples, see the *Adobe Flex 3 Developer Guide*.
- For in-depth information about the steps involved in building and deploying a Flex application, see *Building and Deploying Adobe Flex 3 Applications*.
- If you are using Flex Builder as your IDE and want a quick overview of the capabilities and features of the Flex Builder workbench, see the following topics:
  - “What you can do with Flex Builder” on page 7
  - “About Flex Builder projects” on page 27
  - “About code editing in Flex Builder” on page 102

- “Understanding how projects are built and exported” on page 121
- “About running and debugging applications” on page 136

- For all the API details, see *Adobe Flex Language Reference*. This reference is also available from Flex Dynamic Help in the Flex Builder IDE.

Flash and ActionScript developers

- For all the API details, see *Adobe Flex Language Reference*. This reference is also available from Flex Dynamic Help in the Flex Builder IDE.
- For an overview of the ways you use ActionScript to develop Flex framework and Flash API applications in Flex Builder, see “About ActionScript projects” on page 44.
- To understand how to design and lay out applications in Flex Builder, see “Building a Flex User Interface” on page 64.

Eclipse users

- Experienced Eclipse users will find that Flex Builder uses familiar workbench conventions. For an overview of the perspectives, editors, and views that are contained within the Flex Builder plug-ins, see “Flex Builder Workbench Basics” on page 10.

Getting the most from the Flex and Flex Builder learning resources

Flex Builder includes a variety of media to help you learn the program quickly and become proficient in creating Flex applications. The Flex Builder help system includes several topics that help you learn about Flex Builder, Flex, MXML, and ActionScript. (All documentation is available online in LiveDocs format and Adobe PDF.) You can also consult other online resources as you learn how to build Flex applications.

The following table summarizes additional online resources for learning Flex Builder:

Resource	**Description/Audience**	**Where to Find It**
Flex Support Center	TechNotes, plus support and problem-solving information for Flex users.	www.adobe.com/go/flex_support
Flex Developer Center	Articles and tutorials to help you improve your skills and learn new ones.	www.adobe.com/go/flex_devcenter
Documentation Resource Center	Links to LiveDocs, product manuals in PDF format, and release notes.	www.adobe.com/go/flex_documentation
Adobe Online Forums	Discussion and problem-solving information by Flex users, technical support representatives, and the Flex development team.	www.adobe.com/go/flex_newsgroup
Adobe Training	Courses featuring hands-on tasks and real-world scenarios.	www.adobe.com/go/flex_training

You can purchase printed versions of select titles. For more information, see www.adobe.com/go/buy_books.

Using the Flex Builder help system

The online help system available in the Help menu provides detailed information on all tasks you can perform with Flex Builder.

Set Help preferences

You can control how Help is displayed in the workbench by setting Help preferences.

1 Open the Preferences dialog and select Help.

2 Set the following options as needed:

Use External Browser (This option is only available if your platform supports an embedded browser. See "Changing the default web browser" on page 140.) Lets you display help in the web browser of your choice. By default, the embedded browser of the IDE displays help. Select the Use External Browser option and then select the Web Browser link to select your web browser. The Use External Browser option is only available

Open Window Context Help Determines how to display context-sensitive help links for an open window. By default, context-sensitive help links are displayed in the Dynamic Help view which, when opened, is docked into the current perspective like all other views. You can specify to display context-sensitive help links in an *infopop* (similar to a tooltip).

Open Dialog Context Help Determines how to display context-sensitive help links for an open dialog box. By default, help is displayed in the dialog box. You can specify to display context-sensitive Help links in an *infopop* (similar to a tooltip).

Open Help View Documents Determines where to display documents opened from links in Dynamic Help. By default, help documents open in the Flex Builder IDE editing area. Select in-place to open help documents in the Dynamic Help view window.

Open Help

You can access in-product help while you work in Flex Builder.

❖ Select Help > Help Contents.

Use dynamic help

Dynamic Help is docked to the current perspective and displays topics related to the currently selected MXML tag or ActionScript class.

❖ Select Help > Dynamic Help.

Use context-sensitive help

As you work in Flex Builder, you can display context-sensitive help for specific user interface elements of the workbench (views, dialog boxes, and so on) and language-reference help for code elements.

1 Select an editor, view, dialog box, or other user interface element in the workbench.

2 Press the keyboard shortcut for Dynamic Help: F1 (Windows) or Command+Shift+/ (Mac OS).

For quick access to the *Adobe Flex Language Reference* while writing code, see "Getting help while writing code" on page 107.

Searching Help

There are a few ways you can do full text searches of Flex and Flex Builder Help.

Search in-product help

1 Select Help > Help Contents (a separate Help window will appear).
2 Enter a word or phrase in the text box, and click Go.
3 Click a topic in the list of results to display it.

To search for a specific phrase, enclose it in double quotes.

You can also search directly from the Help menu by selecting Help > Search. The Help panel is opened in the IDE with the search box selected.

Select the scope of the search

You can select the documentation set that you want to search. For example, if you have other Eclipse plug-ins installed, the list of available documentation in help can be quite long. To search only the Flex documentation, you can define the scope of the search to eliminate unwanted search results.

1 In the Help viewer (Help > Help Contents), click the Search scope link.

   The Select Search scope dialog box appears.

2 To create a search scope, click New.
3 Enter a name for the search scope list.
4 Select the Help packages to include in the search.
5 Click OK to save the search scope list.
6 Click OK again to close the Select Search Scope dialog box.

   When you perform a new search, the search is limited to the selected help packages. You can create other search scope lists or search all help topics.

Using Help bookmarks

As you browse documentation in the Help viewer, you can bookmark topics to reference later.

Add a Help bookmark

❖ In Flex Builder Help (Help > Help Contents), with a help topic selected and displayed, select the Bookmark Document button in the Help viewer toolbar.

View your Help bookmarks

❖ In Flex Builder Help, select the Bookmarks tab.

   All the bookmarks you set are listed. Click a bookmark to display the topic. You can also delete one or all of your bookmarks.

Changing the Help viewer font size

The Help viewer in Flex Builder does not support changing the font size used to display Help documentation. However, you can select to run Help in a web browser and control the font size using the browser's font settings.

Display Help in an external browser

1 In Flex Builder, select Window > Preferences > Help.

2. Select the Use External Browser option.
3. (Optional) Select the Web Browser link to select a specific web browser.

   If you bypass this step, Help appears in the system default web browser.
4. Click OK.
5. Select Help > Help Contents.

   Help appears in a web browser. You can control the size of the display font by using the browser's font settings; for example, in Firefox, select Edit > Text Size.

Using the Flex Start page

The Flex Start page appears the first time you run Flex Builder. You can view the Flex Start page at any time by selecting Help > Flex Start Page.

The Flex Start page contains links to the Flex Getting Started Experience, which contains information about building Flex and ActionScript 3.0 applications in Flex Builder, sample applications, and other useful information. Use the Flex Start page much like a web page. Click any of the links to display information or work with sample applications.

Part 1: Getting Started with Flex Builder

Topics

Chapter 2: About Flex Builder

Adobe® Flex® Builder™ is an integrated development environment (IDE) for developing applications that use the Adobe® Flex® framework, MXML, Adobe® Flash® Player 9, ActionScript 3.0, Adobe® LiveCycle™ Data Services ES, and the Adobe® Flex® Charting components.

Flex Builder is built on top of Eclipse, an open-source IDE, and provides all the tools you need to develop Flex 2 and ActionScript 3.0 applications. It runs on Microsoft Windows, Apple Macintosh OS X, and Linux, and is available in several versions. Installation configuration options let you install Flex Builder as a set of plug-ins in an existing Eclipse workbench installation or to create an installation that includes the Eclipse workbench.

Topics

What you can do with Flex Builder

Using Flex Builder, you can develop Flex and ActionScript 3.0 applications in a full-featured IDE that lets you do the following tasks:

- Create Flex projects with or without using the Flex server. For more information, see "Creating Flex projects" on page 32.
- Create ActionScript projects that use the Flash API. For more information, see "About ActionScript projects" on page 45.
- Write and edit your application source code using editors that provide features such as code refactoring, code hinting, streamlined code navigation, and automatic syntax error checking. For more information, see "About code editing in Flex Builder" on page 102.
- Use the MXML editor in Design mode to simplify using view states and transitions, to design using absolute layout options, to drag components onto the design canvas and then reposition and resize them as needed, and so on. For more information, see "Building a Flex User Interface" on page 64.
- Create ActionScript functions within your MXML code or as separate files of ActionScript functions, classes, and interfaces.
- Create custom components and then easily access them using the Components view. For more information, see "Creating Custom MXML Components" on page 227.
- Manage your application projects by using the many features provided by the underlying Eclipse IDE. For example, you can add and delete projects and resources, link to resources outside your project, and so on. For more information, see "Managing projects" on page 36 and "Creating folders and files in a project" on page 41.
- Build your applications using predefined builders or create custom builders using Apache Ant. For more information, see "Building Projects" on page 121.
- Run your applications in a web browser or the stand-alone Flash Player. Create custom launch configurations to control how your applications run. For more information, see "Running your applications" on page 138 and "Managing launch configurations" on page 139.

- Test and debug your applications using the integrated debugging tools in Flex Builder. For more information, see "Running and Debugging Applications" on page 136.
- Publish your application source code so it can be viewed by users and other developers. For more information, see "Publishing source code" on page 134.
- Create library projects that generate shared component library (SWC) and run-time shared library (RSL) files for code reuse and distribution. For more information, see "About library projects" on page 47.
- Customize the IDE. For example, you can arrange the interface to include your favorite tools in the specific layout. For more information, see "Navigating and Customizing the Flex Builder Workbench" on page 52.

Flex Builder versions

Flex Builder is available in two versions: Standard and Professional.

Flex Builder Standard This version provides a full-featured IDE which allows you to create Flex and ActionScript applications using the Flex framework and Flash API. Flex Builder Standard also includes MXML, ActionScript, and CSS editors, as well as debugging tools. Under a trial license, Flex Builder Standard can also provide the advanced charting and advanced data tools included in the Professional version described below.

Flex Builder Professional In addition to the Standard version features, this version includes a library of interactive charts and graphs which enable you to create rich data dashboards, interactive data analysis, and data visualization components. It also includes memory and performance profiling and automated testing tools.

Flex Builder configurations

The Flex Builder installer provides the following two configuration options:

Plug-in configuration This configuration is for users who already use the Eclipse workbench, who want to add the Flex Builder plug-ins to their toolkit of Eclipse plug-ins (for example, someone who also uses Eclipse to develop Java applications). Because Eclipse is an open, extensible platform, hundreds of plug-ins are available for many different development purposes.

Stand-alone configuration This configuration is a customized packaging of Eclipse and the Flex Builder plug-ins created specifically for developing Flex and ActionScript applications. The user interface of the stand-alone configuration is more tightly integrated than in the plug-in configuration, which eliminates much of the potential confusion that the open, multipurpose plug-in configuration might create. The stand-alone configuration is ideal for new users and users who intend to develop only Flex and ActionScript applications.

If you aren't sure which configuration to use, follow these guidelines:

- If you already use and have Eclipse 3.11 (or later) installed, select the plug-in configuration. On Macintosh, Eclipse 3.2 is the minimum version.
- If you don't have Eclipse installed and your primary focus is on developing Flex and ActionScript applications, select the stand-alone configuration. This configuration also allows you to install other Eclipse plug-ins, so you can expand the scope of your development work in the future.

Both configurations provide the same functionality and you select the configuration you want when installing Flex Builder.

Activating Flex Builder

If you're a single-license user, you must activate your license within thirty days of installation. When you start Flex Builder, you will be prompted to enter the serial number. Likewise, if you installed the trial version of Flex Builder, you have 30 days to use it before the trial expires. Once expired, you need to purchase a license to continue using Flex Builder. Product activation does not require you to submit personal information, only your product serial number.

Managing Flex licenses

Each of the products within the Flex product family have separate licenses. For example, you need separate licenses for Flex Builder, Flex Charting, and Adobe LiveCycle Data Services ES. As noted above, when you purchase a Flex product, you have 30 days to activate it by entering the serial number. You can also install trial versions of these products and evaluate them for 30 days. When you acquire a license for a Flex product, you can activate the product from within Flex Builder.

Activate a Flex product in Flex Builder

1. Select Help > Manage Flex Licenses.
2. Select the Flex product you want to activate and enter the serial number.
3. Click Restart. Flex Builder restarts, properly activated with the serial number that you entered.

If you're using the plug-in configuration of Flex Builder (see "Flex Builder configurations" on page 8) and the 30-day activation or trial has expired, the Eclipse workbench and all other plug-ins will continue to work properly. You will not, however, have access to the Flex Builder features (for example, opening an MXML file). When you acquire a serial number you can unlock Flex Builder (or other Flex products) by entering the serial number using the procedure described above.

For more information, visit the Adobe Product Activation Center at www.adobe.com/go/activation.

Chapter 3: Flex Builder Workbench Basics

Adobe® Flex® Builder™ is built on Eclipse, an open-source, integrated development environment (IDE). You use it to develop Flex and ActionScript 3.0 applications using powerful coding, visual layout and design, build, and debugging tools.

Topics

About the workbench

The Flex Builder workbench is a full-featured development environment that is tailored to assist you in developing Adobe Flex, AIR™, and ActionScript applications. Flex Builder is built on Eclipse, an open-source IDE. Flex Builder is a collection of Eclipse plug-ins that let you create Flex and ActionScript 3.0 applications. Much of the basic functionality of the Flex Builder IDE comes from Eclipse. For example, managing, searching, and navigating resources are core features. The Flex Builder plug-ins add the features and functionality needed to create Flex and ActionScript 3.0 applications, and they modify the IDE user interface and some core functionality to support those tasks.

The information you need to use Flex Builder is contained in the Flex Builder documentation. Unless you are using other Eclipse plug-ins (such as CVS or Java) with Flex Builder, or you want to extend the Flex Builder plug-ins (see "Extending the Flex Builder workbench" on page 25), you do not need to be concerned with the underlying Eclipse framework.

Workbench The term *workbench* refers to the Flex Builder development environment. 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. The workbench is the container for all of the development tools you use to develop applications. You might equate it to Microsoft Visual Studio, which provides a framework and core functionality for a variety of development tools.

Perspective A perspective is a group of views and editors in the workbench. Essentially it is a special work environment that helps you accomplish a specific type of task. For example, Flex Builder contains two perspectives. The Flex Development perspective is used for developing applications, and the Flex Debugging perspective is used when debugging your applications. Flex Builder Professional also contains the Flex Profiling perspective.

If you use the Flex Builder plug-in configuration (see "Flex Builder configurations" on page 8), your workbench might contain additional perspectives such as a Java perspective that contains editors and views used to develop Java applications.

For more information about perspectives, see "About Flex Builder perspectives" on page 13.

Editor An editor allows you to edit various file types. The editors available to you depend on the number and type of Eclipse plug-ins installed. Flex Builder contains editors for writing MXML, ActionScript 3.0, and Cascading Style Sheets (CSS) code. For more information about Flex Builder code editing, see "About code editing in Flex Builder" on page 102.

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

Some views support the core functionality of the workbench itself. For example, the Flex Navigator view allows you to manage files and folders within the workbench and the Tasks view displays all of the tasks that are either automatically generated by the workbench or added manually.

The term *view* is synonymous with the term *panel* as it is used in earlier versions of Flex Builder, 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 Flex Builder. For more information, see "Managing projects" on page 36.

Resource The term *resource* is used generically to refer to the files and folders within the projects in a workspace. For more information, see "Creating folders and files in a project" on page 41.

Project All of the resources that make up your applications are contained within projects. You cannot build an application in Flex Builder without first creating a project. You can create three types of projects in Flex Builder: Flex, ActionScript 3.0, and Library projects. For more information, see "Working with Projects" on page 27.

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. For more information, see "Running your applications" on page 138.

About Flex Builder editors

Flex Builder contains editors that allow you to edit MXML, ActionScript 3.0, and CSS code. Editors are the essence of the workbench and views, and the perspectives in which they are contained support their functionality.

Editors are associated with resource types, and as you open resources in the workbench, the appropriate editor is opened. The workbench is a document-centric (and project-centric) environment for developing applications.

When you develop Flex applications in Flex Builder, you use the MXML, ActionScript 3.0, and CSS editors. Each editor provides the functionality needed to author the given resource type. Flex Builder contains the following editors:

MXML editor You use the MXML editor to edit MXML and to embed ActionScript and CSS code in `<mx:Script>` and `<mx: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. For more information, see "About code editing in Flex Builder" on page 102.

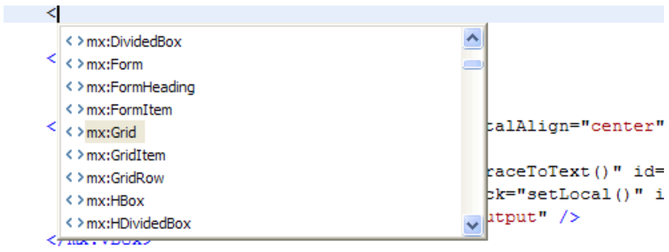
ActionScript editor You use the ActionScript editor to edit ActionScript class and interface files. Although you can embed ActionScript functions into an MXML file by using the `<mx:Script>` tag, it is a common practice to define classes in separate ActionScript files and then import the classes into MXML files. Using this method, you can define most of your Flex 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 "Working with components visually" on page 69 and "Using Styles and Themes" on page 470 in the *Adobe Flex 3 Developer Guide*.

Code hinting

Editors contain many features that simplify and streamline code development. Foremost among these features is Content Assist, which displays code completion hints as you enter MXML, ActionScript, and CSS code.

Code hints appear automatically as you enter your code. (You can also display code hints by pressing Control+Space.) The following example shows code hints in the MXML editor:



```
<
alAlign="center"
aceToText()" id=
k="setLocal()" i
utput" />
```

Code hints appear whenever you begin typing a code expression that Flex or the language (MXML, ActionScript, and CSS) recognizes. For example, if you type the name of a Flex component, you are prompted with a list of all properties of that component.

ActionScript code hinting is also supported. ActionScript code hints are displayed within embedded `<mx:Script>` tags in an MXML document and within stand-alone ActionScript files. Content Assist displays code hints for all ActionScript language elements: interfaces, classes, variables, functions, return types, and so on.

Content Assist also provides code hints for custom MXML components or ActionScript classes that you create yourself. For example, if you define a custom MXML component and add it to your project, code hints are displayed when you refer to the component in your MXML application file.

For more information, see "About Content Assist" on page 104.

Code navigation

Code navigation simplifies the burden of working with code, especially in large projects with many resources. Code navigation includes the ability to select a code element (a reference to a custom Flex 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.

Other code navigation features include code folding, which allows you to collapse and expand multiline code statements. Another feature is the Outline view, which hierarchically presents, and allows you to navigate to, all user interface and code elements in a file. For more information, see "Navigating and organizing code" on page 107.

Code formatting

As you write code, Flex 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). For more information, see "Formatting and editing code" on page 111.

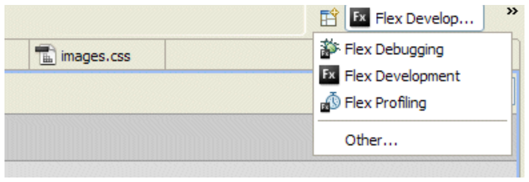
Find references and code refactoring

Flex Builder lets you find all references and declarations to identifiers in a given file, project, or workspace, including references found in elements linked from SWC files and other entries on a library path (for example, classes, interfaces, functions, variables, and some metadata). You use refactoring to rename identifiers in your code while updating all references to them in your entire code base. For more information, see "Finding references and refactoring code" on page 113.

About Flex Builder perspectives

To support a particular task or group of tasks, editors and supporting views are combined into a *perspective*. Flex Builder contains two perspectives: Flex Debugging and Flex Development. Flex Builder Professional contains an additional perspective, Flex Profiling.

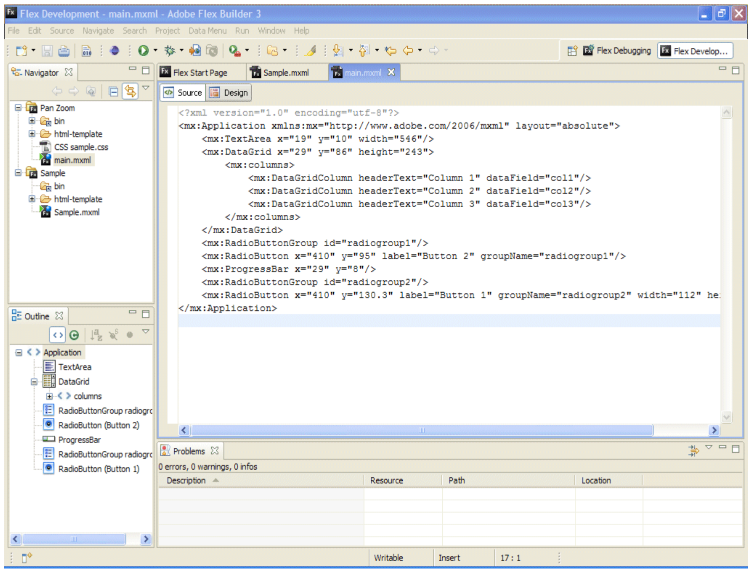
Perspectives change automatically to support the task at hand. For example, when you create a Flex project, the workbench switches to the Development perspective; when you start a debugging session, the Debugging perspective is displayed when the first breakpoint is encountered. You can also manually switch perspectives yourself by selecting Window > Perspective from the main menu (Window > Open Perspective in plugin version). Or, you can use the *perspective bar*, which is located in the main workbench tool bar.



If you use the plug-in configuration of Flex Builder and have other Eclipse plug-ins installed, you might have many additional perspectives. While predefined perspectives are delivered with each Eclipse plug-in, you can customize them to your liking or create your own. Customizing or creating a perspective is a matter of selecting, placing, and sizing the editors and views you need to accomplish your development tasks. For more information about working with and customizing perspectives, see "Working with perspectives" on page 52.

The Flex Development perspective

The Flex Development perspective includes the editors and views you need to create Flex and ActionScript 3.0 applications. When you create a project, Flex Builder switches into the Development perspective so you can begin developing your application. The following example shows the Flex Navigator, Outline, and Problems views:



```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:TextArea x="19" y="10" width="546"/>
    <mx:DataGrid x="29" y="86" height="243">
        <mx:columns>
            <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
            <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
            <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:RadioButtonGroup id="radiogroup1"/>
    <mx:RadioButton x="410" y="95" label="Button 2" groupName="radiogroup1"/>
    <mx:ProgressBar x="29" y="8"/>
    <mx:RadioButtonGroup id="radiogroup2"/>
    <mx:RadioButton x="410" y="130.3" label="Button 1" groupName="radiogroup2" width="112" he
</mx:Application>
```

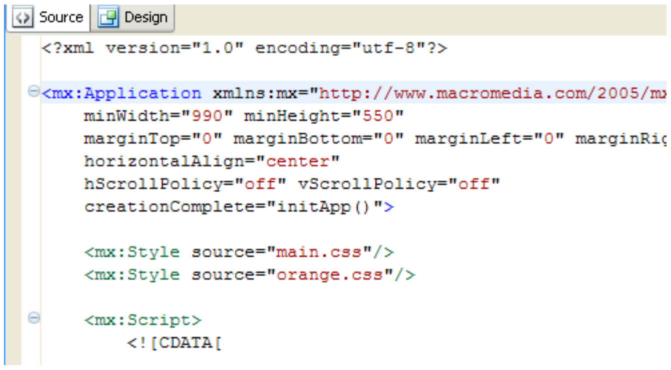
The focal point of the perspective (and the workbench generally) is the editor area. The editor area contains all of the currently open documents in a multitab interface. The supporting views are placed around the editor area. Perspectives predefine the layout of all the elements within it, but you may rearrange them to your liking. For more information, see "Navigating and Customizing the Flex Builder Workbench" on page 52.

In Source (code editing) mode, the Development perspective contains the following elements:

Flex Builder editors

The editor area contains all of the open documents. 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.

The MXML and CSS editors operate in two modes (Source and Design) and the Development perspective is modified to accommodate each set of tasks as you toggle between the two modes. The ActionScript editor is a single-purpose editor that is used to create ActionScript files.

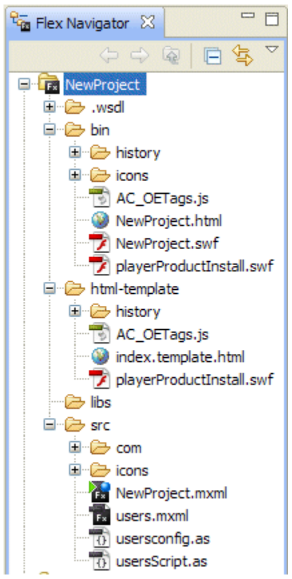


```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application xmlns:mx="http://www.macromedia.com/2005/mx
    minWidth="990" minHeight="550"
    marginTop="0" marginBottom="0" marginLeft="0" marginRig
    horizontalAlign="center"
    hScrollPolicy="off" vScrollPolicy="off"
    creationComplete="initApp()">

    <mx:Style source="main.css"/>
    <mx:Style source="orange.css"/>

    <mx:Script>
        <![CDATA[
```

For more information about using the MXML editor, see "About code editing in Flex Builder" on page 102 and "Working with components visually" on page 69.

Flex Navigator view

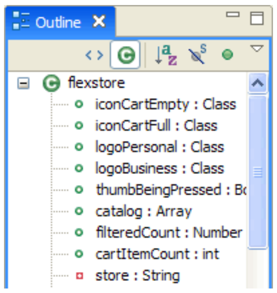
The Flex Navigator view contains all of the projects and resources in the workspace and is therefore an essential element of the Flex Builder workbench. It is always displayed in the Development and Debugging perspectives.



For more information about the Flex Navigator view and working with projects, see "Working with Projects" on page 27.

Outline view

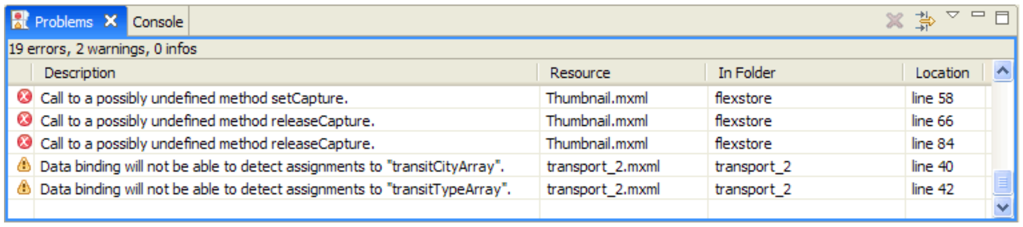
In Source mode, the Outline view presents a hierarchical view of the code structure of the selected MXML or Action-Script document so that you can inspect and navigate the sections or lines of code in the document. The Outline view also displays syntax error alerts that the compiler generates. This view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



For more information about using the Outline view in Source mode, see "Using the Outline view to navigate and inspect code" on page 108.

Problems view

As you enter code, the Flex Builder compiler detects syntax and other compilation errors, and these are displayed in the Problems view. When you debug your applications, errors, warnings, and other information are displayed in the Problems view.



19 errors, 2 warnings, 0 infos

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

For more information, see "Running and Debugging Applications" on page 136.

***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 "About markers" on page 114. For an introduction to the optional views that are available in Flex Builder, see "Other useful workbench views" on page 22.*

Flex Development perspective in Design mode

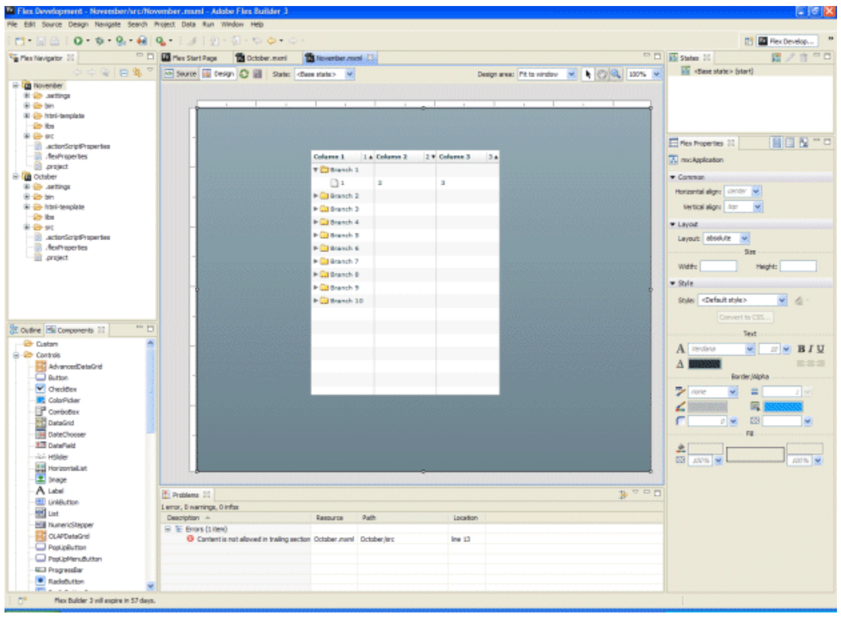
You visually lay out and design your Flex applications in the MXML editor in MXML Design mode. Design mode is the visual representation of the code that you edit in Source mode. In Design mode, however, additional views are added to support design tasks. These are the Components, Flex Properties, and States views. In addition, when you are in Design mode, the Outline view displays the MXML structure of your Flex applications. You can also display and edit CSS in CSS Design mode. For more information about designing Flex applications in Flex Builder, see "Building a Flex User Interface" on page 64.

***Note:** Design mode is not available when working with ActionScript projects. To preview the user interface of your ActionScript applications, you need to build and run them. For more information about working with ActionScript, see "About ActionScript projects" on page 44 and "Running and Debugging Applications" on page 136.*

In Design mode, the development perspective contains the MXML editor and the Components, States, and Flex Properties, and Outline views.

The MXML editor

In MXML Design mode, you interact with your Flex applications visually; dragging and dropping components on to the design area, selecting and resizing components, and so on. You can also expand the MXML editor in Design mode to clearly see and select individual components, and use pan and zoom to get a closer look at items; this is especially useful when you have embedded container components. For more information about working in Design mode, see “Building a Flex User Interface” on page 64.



1 error, 0 warnings, 0 infos

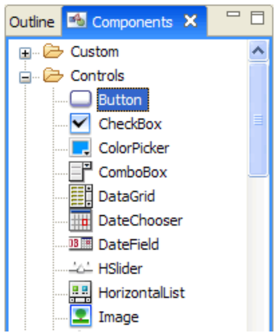
Errors (1 item)

Content is not allowed in trailing section October.mxml October/src line 13

Flex Builder 3 will expire in 57 days.

Components view

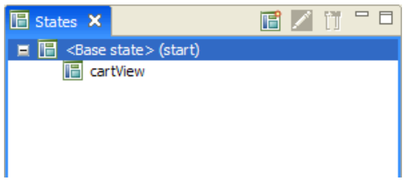
The Components view contains all of the standard Flex components, which you can select and add to the design area. As you create your own custom components, they are also displayed in the Components view.



For more information, see “Adding and changing components” on page 66.

States view

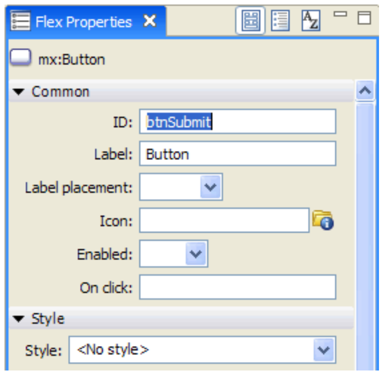
Flex allows you to create applications that change their appearance based on events that are triggered directly by the user or events that are generated programmatically. These user interface transformations are referred to as *view states*. You create and manage view states in the States view.



For more information about view states, see “Adding View States and Transitions” on page 93.

Flex Properties view

When a Flex component is selected, its properties are displayed in the Flex 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.

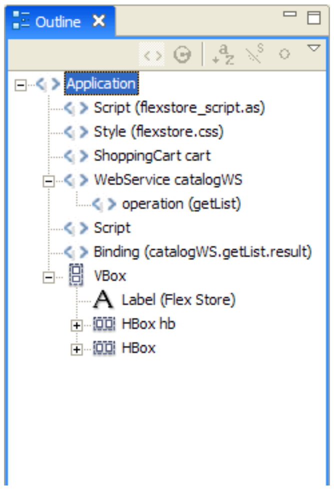


ID: btnSubmit
Label: Button
Label placement:
Icon:
Enabled:
On click:
Style: <No style>

For more information, see “Setting component properties” on page 74.

Outline view

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

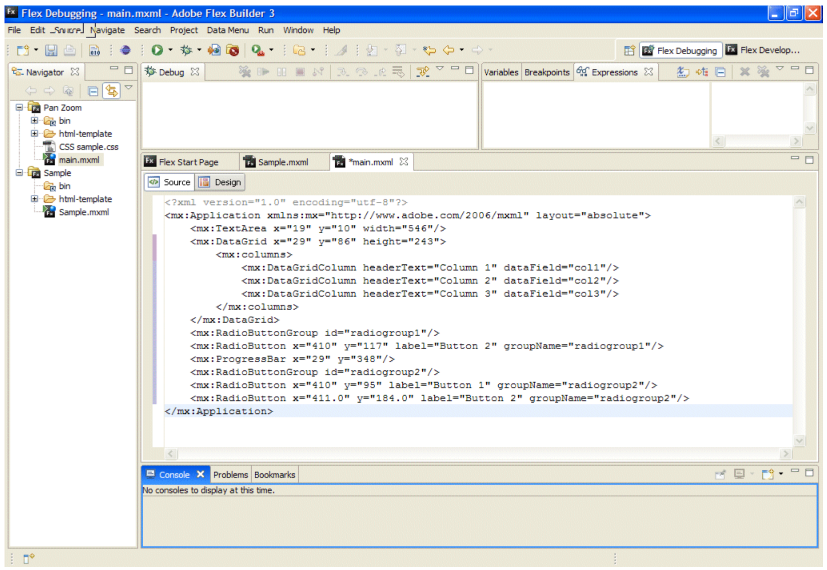


For more information about working with the Outline view in Design mode, see “Inspecting the structure of your MXML” on page 77.

The Flex Debugging perspective

The Flex Debugging perspective contains the tools you need to debug your Flex and ActionScript applications. Like the Development perspective, the primary tool within the debugging perspective is the editor. In the context of debugging your applications, the editor works with the debugging tools to locate and highlight lines of code that need attention so that you can fix them and continue testing your application.

For example, you can set breakpoints in your script to stop the execution of the script so that you can inspect the values of variables and other information up to that point. You can also move to the next breakpoint or step in to a function call to see the variable values change.



```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:TextArea x="19" y="10" width="546"/>
    <mx:DataGrid x="29" y="86" height="243">
        <mx:columns>
            <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
            <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
            <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:RadioButtonGroup id="radiogroup1"/>
    <mx:RadioButton x="410" y="117" label="Button 2" groupName="radiogroup1"/>
    <mx:ProgressBar x="29" y="348"/>
    <mx:RadioButtonGroup id="radiogroup2"/>
    <mx:RadioButton x="410" y="95" label="Button 1" groupName="radiogroup2"/>
    <mx:RadioButton x="411.0" y="184.0" label="Button 2" groupName="radiogroup2"/>
</mx:Application>
```

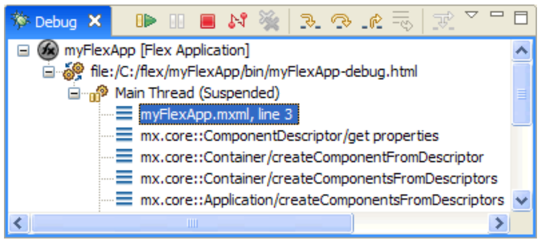
No consoles to display at this time.

The Debugging perspective appears automatically when the first breakpoint is reached. You can also switch to the Debugging perspective manually by selecting it from the Perspective bar, which is located at the right edge of the main workbench toolbar.

The Debugging perspective contains Debug, Breakpoints, Console, Variables, and Expressions views.

Debug view

The Debug view (in other debuggers this is sometimes referred to as the *callstack*) displays the stack frame of the suspended thread of the Flex application you are debugging. You use the Debug view to manage the debugging process. For example, the Debug view allows you to resume or suspend the thread, step into and over code statements, and so on.



```
myFlexApp [Flex Application]
    file:/C:/flex/myFlexApp/bin/myFlexApp-debug.html
        Main Thread (Suspended)
            myFlexApp.mxml, line 3
            mx.core::ComponentDescriptor/get properties
            mx.core::Container/createComponentFromDescriptor
            mx.core::Container/createComponentsFromDescriptors
            mx.core::Application/createComponentsFromDescriptors
```

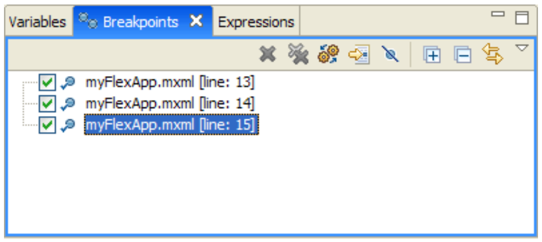
For more information about working with the Debug view, see “Managing the debugging session in the Debug view” on page 143.

Flex applications are single-threaded (not multithreaded like Java, for example) and you can debug only one Flex application at a time. Therefore, when you debug a Flex 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; Flex 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.

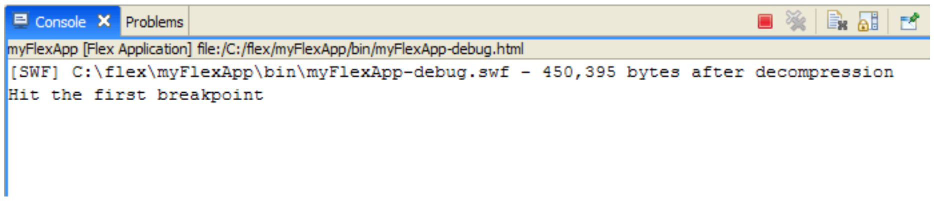


```
myFlexApp.mxml [line: 13]
myFlexApp.mxml [line: 14]
myFlexApp.mxml [line: 15]
```

For more information, see "Managing breakpoints in the Breakpoints view" on page 143.

Console view

The Console view displays the output from trace statements placed in your ActionScript code and also feedback from the debugger itself (status, warnings, and errors).

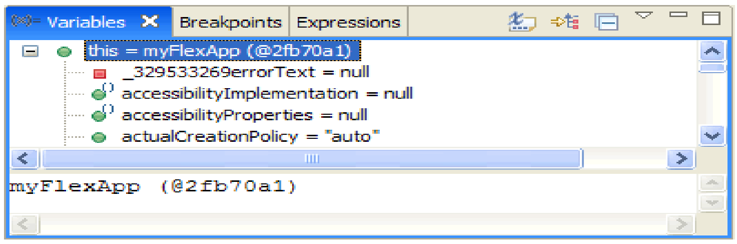


```
myFlexApp [Flex Application] file:/C:/flex/myFlexApp/bin/myFlexApp-debug.html
[SWF] C:\flex\myFlexApp\bin\myFlexApp-debug.swf - 450,395 bytes after decompression
Hit the first breakpoint
```

For more information, see "Using the Console view" on page 144.

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 need to resolve.

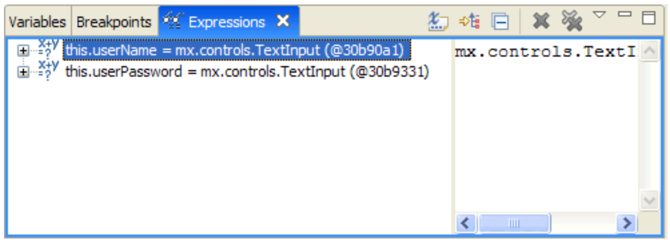


```
this = myFlexApp (@2fb70a1)
    _329533269errorText = null
    accessibilityImplementation = null
    accessibilityProperties = null
    actualCreationPolicy = "auto"
myFlexApp (@2fb70a1)
```

For more information, see "Managing variables in the Variables view" on page 144.

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.



```
this.userName = mx.controls.TextInput (@30b90a1)
this.userPassword = mx.controls.TextInput (@30b9331)
mx.controls.TextI
```

When you debug your application, you can then monitor and, if needed, modify the values. You can also add and remove variables in the Expressions view. For more information, see "Using the Expressions view" on page 145.

For more information about debugging Flex and ActionScript applications, see "Running and Debugging Applications" on page 136.

The Flex Profiling perspective

Flex Builder Professional contains an additional perspective. The Adobe Flex profiler helps you identify performance bottlenecks and memory leaks in your applications. 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. For more information about the profiler, see "About profiling" on page 155.

Other useful workbench views

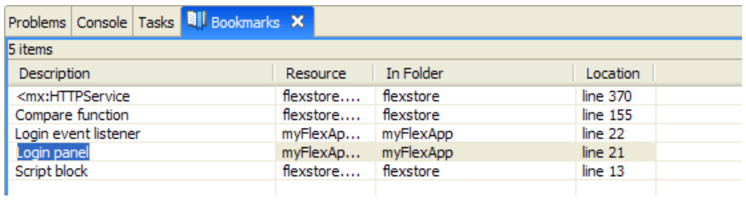
In addition to the editors and views associated with Flex Builder's default development, debugging, and profiling perspectives, the workbench contains other views that help you streamline the application development process.

You can access views that are not already displayed with a perspective and add them to the workbench by selecting Window > Other Views > General (Window > Show View Other in plugin version). These optional views are categorized by type and are associated with distinct workbench functionality or with specific Eclipse plug-ins. For more information about working with views, see "Working with editors and views" on page 54.

You will find that several workbench views in particular are valuable aids as you develop your applications in Flex Builder. These include the Tasks, Bookmarks, and Search views.

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 keeping track of noteworthy items. Selecting a bookmark locates and displays it in the workbench.



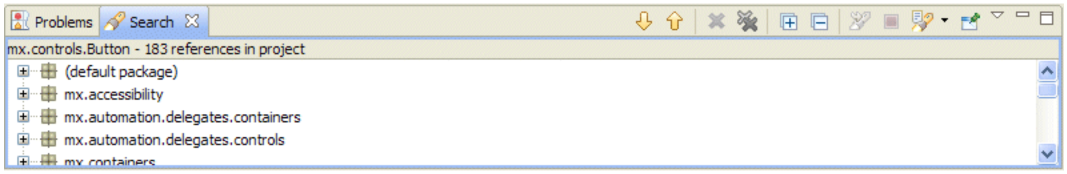
5 items

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

For more information about the Bookmarks view, see "About markers" on page 114.

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.



mx.controls.Button - 183 references in project
(default package)
mx.accessibility
mx.automation.delegates.containers
mx.automation.delegates.controls
mx.containers

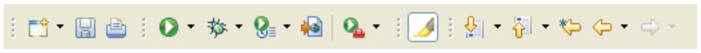
For more information about the Search view, see “Searching in the workbench” on page 60.

Workbench menus, toolbars, and shortcuts

All of the workbench commands are contained in the menu system, in right-click context menus, from toolbars, and through keyboard shortcuts.

The workbench toolbar

The workbench toolbar contains buttons for important and frequently used commands. These commands are also available from various Flex Builder menus.



The following buttons appear in the workbench toolbar (shown left to right):

New Displays a pop-up menu that displays all the types of projects and documents you can create.

Save Saves the document that is open in the editor and currently selected.

Print Source Prints the document that is open in the editor and currently selected.

Build All Appears when “Build automatically” is deselected from the Project menu.

Run Opens the main application SWF file in your default web browser or directly in stand-alone Flash Player. You can also select other application files in the project from the attached pop-up menu. For more information, see “Running your applications” on page 138.

Debug Uses the current project’s main application file to begin a debugging session. You can also select other application files in the project from the attached pop-up menu. For more information, see “Starting a debugging session” on page 141.

Profile Creates, manages, and runs configurations. For more information, see “About profiling” on page 155.

Export Release Build Launches a wizard that helps you choose the application for which you want to export an optimized release-quality version.

External Tools Selects a custom launch configuration.

Mark Occurrences Allows you to select and mark code occurrences in Source mode.

Next Annotation Allows you to select and move forward to code annotations in Source mode.

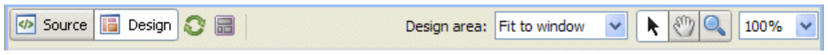
Previous Annotation Allows you to select and move backward to code annotations in Source mode.

Last Edit Location Returns you to the location of the last edit you made to a resource (for example, the exact line of code or, in Design mode, the user interface element in which you added a control or set a property).

Back and Next Go backward or forward to previously selected documents. You can also select from the list of currently open documents from the attached pop-up menu.

The MXML editor toolbar

The MXML editor toolbar contains several buttons that allow you to control the editor in Source and Design modes. To see the toolbar, open an MXML file in Design mode.



The following buttons appear in the MXML editor toolbar (shown left to right):

Source Displays the editor in Source mode, which is where you edit code.

Design Displays the editor in Design mode, which is where you visually lay out and design your Flex applications.

Refresh Reloads the visual elements (images, SWF files, or class files containing drawing API methods) that define the visual design of your application. Collectively, these elements are referred to as a *skin*. For more information, see “Creating Skins” on page 550 in the *Adobe Flex 3 Developer Guide*.

Show Surrounding Containers Expands the Design mode view so that you can see and select individual components. For more information, see “Laying out your user interface” on page 85.

State Pop-up menu displays all the defined views states. Selecting view states updates the display of the visual design. For more information, see “Adding View States and Transitions” on page 93.

Design Area Displays and allows you to select predefined design area sizes (1024 x 768 pixels and 800 x 600 pixels, for example). You can also set a custom size. For more information, see “Using the MXML editor in Design mode” on page 69.

Select Mode Engaged by default when a file is opened; it allows you to select, move, and resize items.

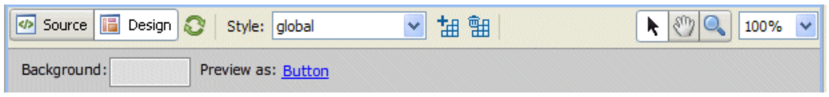
Pan Mode Allows you to pan and scroll around in design area; items cannot be selected or moved in Pan mode.

Zoom Mode Defaults to zoom-in preset magnification values. To zoom out press Alt+Click (Opt+Click on Macintosh). You can double click the Zoom Mode button to return the design view to 100%.

Magnification Pop-up menu displays specific zoom percentages, which can also be selected from the Design > Magnification menu. The default setting is 100%.

The CSS editor toolbar

The CSS editor contains several buttons that allow you to control the editor in Source and Design modes. To see the CSS editor toolbar, open a CSS file in Design mode.



The following buttons appear in the CSS toolbar (shown left to right):

Source Displays the editor in Source mode, which is where you edit code.

Design Displays the editor in Design mode, which is where you visually lay out and design your Flex applications.

Refresh Reloads the visual elements (images, SWF files, or class files containing drawing API methods) that define the visual design of your application. Collectively, these elements are referred to as a *skin*. For more information, see “Creating Skins” on page 550 in the *Adobe Flex 3 Developer Guide*.

Style Pop-up menu lists the styles contained in your CSS file.

New Style Launches the New Style dialog box which allows you to choose the selector types and components to apply the new style.

Delete Style Deletes the selected style for your CSS file.

Select Mode Engaged by default when a file is opened. It allows you to select, move, and resize items.

Pan Mode Allows you to pan and scroll around in design area. Items cannot be selected or moved in Pan mode.

Zoom Mode Defaults to zoom-in preset magnification values. To zoom out press Alt+Click (Opt+Click on Macintosh). Double click the Zoom Mode button to return the design view to 100%.

Magnification Pop-up menu displays specific zoom percentages which can also be selected from the Design > Magnification menu. The default setting is 100%.

Background Launches a color picker where you can select a background color for the preview area. Changing this color does not change the CSS file nor does it affect your Flex application when you run it.

Preview as (If applicable) Shown when the style rule is not tied to one specific MXML component.

Edit scale grids (not shown) (If applicable) Shown when the style rule uses image file skins.

Using keyboard shortcuts

Many operations that are available from the menu system in Flex Builder are also available as keyboard shortcuts.

Display the list of all keyboard shortcuts in Flex Builder

❖ Select Help > Key Assist, or enter Control+Shift+L (Command+Shift+L on Macintosh).

You can use Key Assist as a reference to all the Flex Builder keyboard shortcuts, or you can run these commands directly from the Key Assist panel by double-clicking the commands. You can also modify keyboard shortcuts or create your own. For more information, see “Changing keyboard shortcuts” on page 58.

Extending the Flex Builder workbench

Flex Builder is a collection of Eclipse plug-ins that provide the tools you need to create Flex and ActionScript 3.0 applications. The Eclipse plug-in framework allows plug-ins to expose extension points, which can be used to extend the features and capabilities of the tool. For more information, see *Adobe* Flex Builder *3 Extensibility API Reference* in Help.

Part 2: Flex Builder Basics

Topic

Chapter 4: Working with Projects

Adobe Flex Builder lets you create, manage, package, and distribute projects for building web and desktop 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 Flex Builder.

Topics

About Flex Builder projects

Flex Builder uses a traditional approach to software development: grouping the resources (folders and files) that constitute an application into a container called a *project*. A 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.

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

In addition to Flex projects, Flex Builder provides a basic project type called an *ActionScript project*. Using an ActionScript project, you can code and debug ActionScript applications that directly access the Adobe Flash Player APIs and are compiled into SWF files. ActionScript projects do not use the Flex framework or MXML language.

Flex and ActionScript applications

Using Flex Builder, you can create Flex and ActionScript applications. You compile Flex applications into standalone SWF files. For more information, see “Working with Projects” on page 27 and “About ActionScript projects” on page 44.

Adobe AIR applications

With Flex Builder you can debug, package, and manage AIR projects. Flex Builder enables you to run Flex applications in AIR. You create AIR projects by using the New Flex Project wizard. Use the Export Release Build feature to generate a release-quality, installable AIR package. For more information, see *Developing AIR Applications with Adobe Flex 3*.

The Adobe AIR Marketplace is a place where AIR developers can publish AIR applications for users to download. To find the Marketplace, go to http://www.adobe.com/go/marketplace. If you have questions on the Adobe AIR Marketplace, go to http://www.adobe.com/go/marketplace_faq.

Flex libraries

You also use Flex Builder to build custom code libraries that you 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 more information, see "About library projects" on page 47.

Applications contained in projects

To begin building a Flex or ActionScript application in Flex Builder, you must first create a project. When you create a Flex project, a main application file is created for you. Then you add other resources such as MXML application files, custom MXML component files, ActionScript files, and all of the other assets that make up your Flex application. When you create an ActionScript project, a main ActionScript file is created; then you can build an application by using ActionScript and the Flash Player API. For more information, see "Creating Flex projects" on page 32 and "Managing projects" on page 36.

Projects managed in workspaces

Projects are managed from within a *workspace*, which is a defined area of the file system that contains the resources (files and folders) that make up your applications. By default, your projects reside within the workspace. You can, however, create projects that are located outside the workspace; Flex Builder automatically links them to the workspace. To switch workspaces, you must restart Flex Builder.

More than one project in each workspace

You can add as many projects to a workspace as needed. All of your projects are displayed in the Flex Navigator view, and you can manage them as you need to—adding resources, organizing your projects into folders, and building projects in the workspace. For more information, see "Managing projects" on page 36 and "Creating folders and files in a project" on page 41.

External linked resources

In addition to the resources in your projects, you can link to resources outside a project and workspace. Linked external resources appear as part of the project but reside outside the project's location. For more information, see "Linking to resources outside the project workspace" on page 42.

More than one application in a project

Flex Builder lets you define more than one file in your project as an application. When you create a project, Flex Builder generates a main application file that serves as the entry point into your application, and the compiler uses this file to generate the application SWF file. However, if your project is complex, you can create additional application files. All application files must reside in the src folder under the root folder of your project. For more information, see "Managing project application files" on page 40.

Support for Multiple Flex SDKs

You could have projects that are in progress or an older project code base that must be maintained. With Flex Builder, you can work with different versions of the Flex SDK. To specify the installed SDKs, you configure the Flex Builder workspace, which provides a default SDK for any project. After you set up a project, you can add, remove, or edit SDK configurations in the Preferences dialog by selecting Flex > Installed SDKs. You can also modify the SDK configurations by selecting Project > Properties > Flex Compiler. For more information, see "Using multiple SDKs in Flex Builder" on page 131.

Automatic project builds

By default, your project is automatically built any time you save changes to a file. You have complete control over how and how often your applications are built. If you have no special requirements for customizing the build, it works transparently and automatically generates the application SWF files. For more information, see "Building Projects" on page 121.

Export Release Build

When your application is ready to deploy, you use the Export Release Build wizard to create a release-quality *non-debug* version of your Flex, AIR, or ActionScript application. The wizard copies required assets to a bin-release folder separate from the debug version, Export Release Build with or without source code. This version is an optimized production build that can be viewed by end users. 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 install before running an application. For more information, see "Export Release Build" on page 126.

Custom Ant scripts

Apache Ant is a Java-based build tool that you use to create custom scripts for building your Flex applications in Flex Builder. You use Ant to modify and extend the standard build process. For more information, see "Customizing builds with Apache Ant" on page 131.

Project types

You use Flex Builder to create project types in the following configurations:

Flex projects

Flex project configuration options are based on how your Flex application accesses data and if you have Adobe LiveCycle Data Services ES installed. You can create Flex projects for web (runs in Flash player) or desktop (runs in Adobe AIR) applications. Here are the options:

None If you do not have an application server, this basic configuration lets you specify the output folder for your compiled Flex application. You also set the build paths for your new project.

ASP .NET With Microsoft Windows and Microsoft Visual Web Developer installed, you can create Flex projects that use ASP .NET Development Server for deployment. Also, if you have access to Internet Information Service (IIS), you can create Flex projects with a Flex output folder under IIS.

ColdFusion This project configuration lets you create Flex projects that use ColdFusion with LiveCycle Data Services or ColdFusion Flash Remoting. If neither option is selected, a ColdFusion project is created with a Flex output folder under web root (or virtual folder).

J2EE This project configuration lets you create Flex projects that use J2EE with or without remote object access service and LiveCycle Data Services. When no option is selected, a Flex output folder is created under the Java application server root. If you select the Use Remote Object Access Service option, you can use Flex with LiveCycle Data Services and your project is deployed on a LiveCycle Data Services server. With the Eclipse Web Tools Project (WTP) plug-in installed, you select the Create Combined Java/Flex Project Using WTP option to create combined Java/Flex projects with or without remote object access service. For locally compiled projects with WTP, projects are deployed on your J2EE server.

You can use LiveCycle Data Services with or without WTP. If you use it with WTP, the project will not be deployed on the local LiveCycle Data Services server, but it will be deployed using WTP features.

PHP This project configuration lets you create Flex projects that have a Flex output folder under the Apache/IIS web root (or virtual folder). You configure the URL and run and debug your Flex application by using your PHP server or scripts.

Other If you have an application server other than those previously listed, this option lets you specify the output folder for your compiled Flex application. You can also set the build paths for your new project.

ActionScript projects

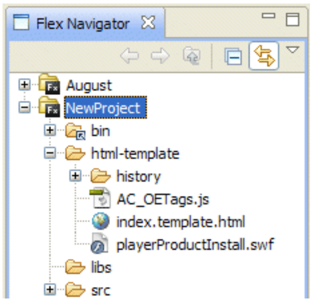
Based on the Flash API, not the Flex framework, ActionScript projects let ActionScript developers use Flex Builder to code, build, and debug ActionScript-only applications. Because these projects do not use MXML to define a user interface, you cannot view the application layout and design in Design mode. You work exclusively in the source editor, the debugging tools as necessary, and then build the project into SWF application files to preview and test your application in a web browser or stand-alone Flash Player. For more information about ActionScript projects, see "About ActionScript projects" on page 44.

Flex library projects

Library projects are used to package and distribute components and other resources. They generate SWC files that you add to other projects or distribute to other developers. For more information, see "About library projects" on page 47.

Projects in the Flex Navigator view

All projects in a workspace are displayed in the Flex Navigator view, as the following example shows. 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.



Flex Builder provides the following wizards to help you create projects:

- The New Flex Project wizard automatically generates Flex project configuration files, the output (bin) folder where your compiled application resides, and the main application file. It also lets you create an Adobe AIR project.
- The New ActionScript Project wizard generates a main ActionScript application file.
- The New Flex Library Project wizard helps you generate a Flex Library Project that you use to package and distribute components and other resources.

From the Flex Navigator view, you can open the project resources for editing. For example, you can edit MXML and ActionScript in `<mx:Script>` blocks and CSS in `<mx: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 Flex Builder editors, see "About code editing in Flex Builder" on page 102 and "Building a Flex User Interface" on page 64.

Then you add projects, files, and folders, and organize and manage them as needed (see "Creating folders and files in a project" on page 41).

You can also modify the Flex Navigator view's appearance. For example, you can expand and collapse projects and folders, limit which projects and resources are visible by creating a working set (a collection of resources), create display filters, and sort resources by name and type. For more information about modifying views, see "Navigating and Customizing the Flex Builder Workbench" on page 52.

Most menu commands that you use in the Flex Navigator view are also available from the Flex Navigator view's context menu. For example, instead of selecting File > New, you can right-click (Control-click on Macintosh) in the Flex Navigator view, and select New from the context menu.

For more information about working with projects in the Flex Navigator view, see "Managing projects" on page 36 and "Creating folders and files in a project" on page 41.

Project resources

Flex and ActionScript applications support several standard resource types (MXML, ActionScript, and CSS). The following table lists the resource types that you can add to your projects. (To add these resources, select File > New.)

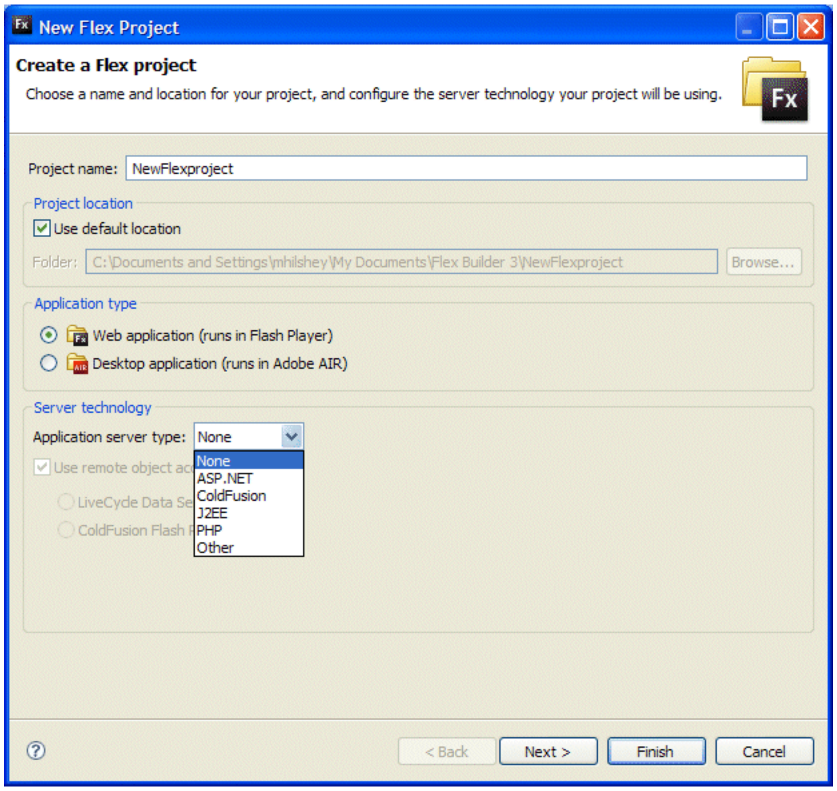
Resource type	Description
ActionScript Class	An ActionScript class file. When you add this type of resource, the New ActionScript Class wizard prompts you for class definition elements, such as the superclass, interfaces, and so on. For more information about working with ActionScript in Flex Builder, see "Creating an ActionScript class" on page 46.
ActionScript File	A text file template for creating ActionScript functions.
ActionScript Interface	An ActionScript interface file. When you add this type of resource, the New ActionScript Interface wizard prompts you for interface definition elements such as extended interfaces and the package in which they reside. For more information about working with ActionScript in Flex Builder, see "Creating an ActionScript interface" on page 46.
ActionScript Project	An ActionScript project based on the Flash API, not the Flex framework. ActionScript projects let ActionScript developers use Flex Builder to code, build, and debug ActionScript-only applications. For more information, see "Creating ActionScript projects" on page 45
CSS File	A text file template for creating a Cascading Style Sheets file.
File	An unformatted text file. For more information, see "Creating folders and files in a project" on page 41.
Flex Project	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. For more information, see "Setting Flex project properties" on page 36.
Flex Library Project	Flex Library Projects are used to package and distribute components and other resources. They generate SWC files that you add to other projects or distribute to other developers. For more information, see "About library projects" on page 47.
Folder	A standard file system folder for organizing the contents of your projects. For more information, see "Creating folders and files in a project" on page 41.
MXML Application	A standard Flex application file with the `<mx:Application>` tag as the root MXML element. A Flex project can have more than one application file. For more information, see "Managing project application files" on page 40.
MXML Component	A standard Flex component file with the `<mx:Canvas>` tag as the root MXML element. For more information, see "Creating MXML components visually" on page 227.

Resource type	Description
MXML Module	A resource that can be added to an existing application project or created separately, but always associated with one application. For more information on using modules, "Creating modules in Flex Builder" on page 147.
Other	Other file types that are registered in Flex Builder. Select File > New > Other to add any other file types. For example, if you have a Java plug-in installed in Flex Builder, you can add new Java classes, interfaces, and packages. When a file type is registered in Flex Builder, a corresponding editor is also available in the workbench. For more information, see "Associating editors with file types" on page 56. You can always add unregistered file types to your projects by importing them. For more information see "Importing projects" on page 37

For more information about adding resources to your projects, see "Creating folders and files in a project" on page 41.

Creating Flex projects

When you create a project, the New Flex Project wizard guides you through the steps, prompting you for the type of project to create, the project name, location, and other options.


New Flex Project

Create a Flex project

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: NewFlexproject

Project location

Use default location [✓]

Folder: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3\NewFlexproject

Application type

[✓ Web application (runs in Flash Player)]
Desktop application (runs in Adobe AIR)

Server technology

Application server type: None

For information about creating an ActionScript project, see "Creating ActionScript projects" on page 45. For information about creating library projects, see "About library projects" on page 47.

Creating a Flex project with no server

If you do not have a server to configure, this basic configuration lets you specify the output folder for your compiled Flex application. You can optionally set the build paths for your new project.

1 Select File > New > Flex Project.

2 Enter a project name.

3 Select the project location. The default location is the workspace, which is My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myFlexApp*). To choose a different project location, deselect the Use Default Location option.

4 Choose the application type (web or desktop).

5 For application server type, choose None.

6 Click Finish or click Next to select more configuration options.

7 (Optional) Specify the output folder for your Flex application. Click Finish or click Next to select more configuration options.

8 (Optional) Click Next to set the build paths for your new project. Click the Source Path and Library Path tabs to specify main source folder, main application file, and output folder URL.

9 Click Finish to create your project.

Creating 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 Select File > New > Flex Project.

2 Enter a project name.

3 Specify the project location. The default location is the workspace, which is My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myFlexApp*). To choose a different project location, deselect the Use Default Location option.

4 Choose the application type (web or desktop).

5 For application server type, choose ASP .NET.

6 Click Next.

7 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 Flex application.

8 Click Finish or click Next to select more configuration options.

9 (Optional) Click Next to set the build paths for your new project. Click the Source Path and Library Path tabs to specify main source folder, main application file, output folder, and output folder URL.

10 Click Finish to create the project.

Creating a Flex project with J2EE

This project configuration lets you create Flex projects that use a J2EE servlet with the remote object access service option. When no option is selected, and Java server is used, a Flex 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:** To use LiveCycle Data Services ES in your Flex projects, you must have LiveCycle Data Services ES installed.*

You have two compile options for creating a Flex project that uses J2EE. The recommended option compiles the application locally, and then saves the files (including the SWF file and HTML wrapper) on the server. The other option compiles the application source file directly on the server.

1 Select File > New > Flex Project.

2 Enter a project name.

3 Specify the project location. The default location is the workspace, which is My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myFlexApp*). To choose a different project location, deselect the Use Default Location option.

4 Choose the application type (web or desktop).

5 For application server type, choose J2EE.

6 Select the Use Remote Object Access Service option. LiveCycle Data Services ES is automatically selected. If you installed WTP, you can also choose to create a combined Java and Flex project that uses WTP (the Java source folder is selected for you).

7 Click Next.

8 Configure the J2EE server.

- If you selected the Use Remote Access Service and LiveCycle Data Services options, specify the root settings:

  Root Folder is the Flex server (web application) that serves your application (for example, C:\fds2\jrun4\servers\default\flex). If you choose not to use the default Flex development server option, you can specify a new location for the root folder, but it must be a valid folder that is mapped to the specified root URL. If you are using a remote server, specify the location; for example, *myServer**MyApplications*\jrun4\servers\default\flex.

  Root URL is a valid root URL of the Flex server (web application) that serves your application. The default root URL for local server instances is http://localhost:8700/flex/. If you use a remote server, the URL might look like this: http://*myserver.com*:8700/flex/.

  Context Root should typically match the last segment of the root URL path.

- If you selected the Create Combined Java/Flex Project Using WTP option (with or without LiveCycle Data Services):

  - Specify the names of your Java and Flex source folders and target runtime.

  When you create a Flex project with LiveCycle Data Services ES, Flex 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 LiveCycle Data Services ES, specify a flex.war file, which is located in the server installation folder.

9 Specify the location to compile your project.

- For applications that compile locally, Flex Builder creates a *projectname*-debug folder in which the SWF files and HTML wrappers are saved.
- For applications that compile on the server, the project location must be on the server.

10 (Optional) Click Next to set the build paths for your new project. Click the Source Path and Library Path tabs to specify main source folder, main application file, and output folder URL.

11 Click Finish to create your project.

***Note:** Regardless of which option you choose for a LiveCycle Data Services ES project in Flex Builder, you must specify a valid LiveCycle Data Services ES root folder and root URL. These values map the root of a LiveCycle Data Services ES web application. If you deselect the options, you must enter only your web root and root URL.*

Creating a Flex project with ColdFusion

To access data that uses ColdFusion, you must have ColdFusion MX 7 Updater 2 (7.0.2) or Adobe ColdFusion 8. You can optionally install the ColdFusion Extensions for Flex Builder. For more information, see the ColdFusion product page.

1 Select File > New > Flex Project.

2 Enter a project name.

3 Specify the project location. The default location is the workspace, which is Documents and Settings\username\\workspace\\.

4 Specify the application type (web or desktop).

5 For application server type, select ColdFusion, then choose the following options:

Use Remote ObjectAccess Service If you deselect this option, the Flex output folder is under the ColdFusion web root or virtual folder. You must enter only your web root and root URL.

If you select this option, you have the following choices:

LiveCycle Data Services The Flex output folder is under the ColdFusion web root or virtual folder. Additionally, you can choose whether your project is compiled in Flex Builder (recommended, because this option generates an HTML wrapper) or whether your project uses the web-tier compiler. Flex Builder adds fds.swc to the library path. If you use the web-tier compiler, the project must be located under the ColdFusion web root.

ColdFusion Flash Remoting The Flex output folder is under the ColdFusion web root or virtual folder.

6 Click Next to configure the ColdFusion server.

7 Choose the server location and compilation options.

Select the ColdFusion installation type: Standalone or Deployed to J2EE server.

Click Validate Configuration to ensure the setup is correct, then click Next.

8 (Optional) Click Next to set the build paths for your new project. Click the Source Path and Library Path tabs to specify main source folder, main application file, and output folder URL.

9 Click Finish.

Creating a Flex project with another server

If you have another type of server to configure, one that is not listed on the server type pop-up menu, this basic configuration lets you specify the output folder for your compiled Flex application. You can optionally set the build paths for your new project.

1 Select File > New > Flex Project.

2 Enter a project name, and then select the project location.

3 Choose the application type (web or desktop).

4 For server type, choose Other.

5 Click Finish or click Next to select more configuration options.

6 (Optional) Specify the output folder for your Flex application, and then click Finish.

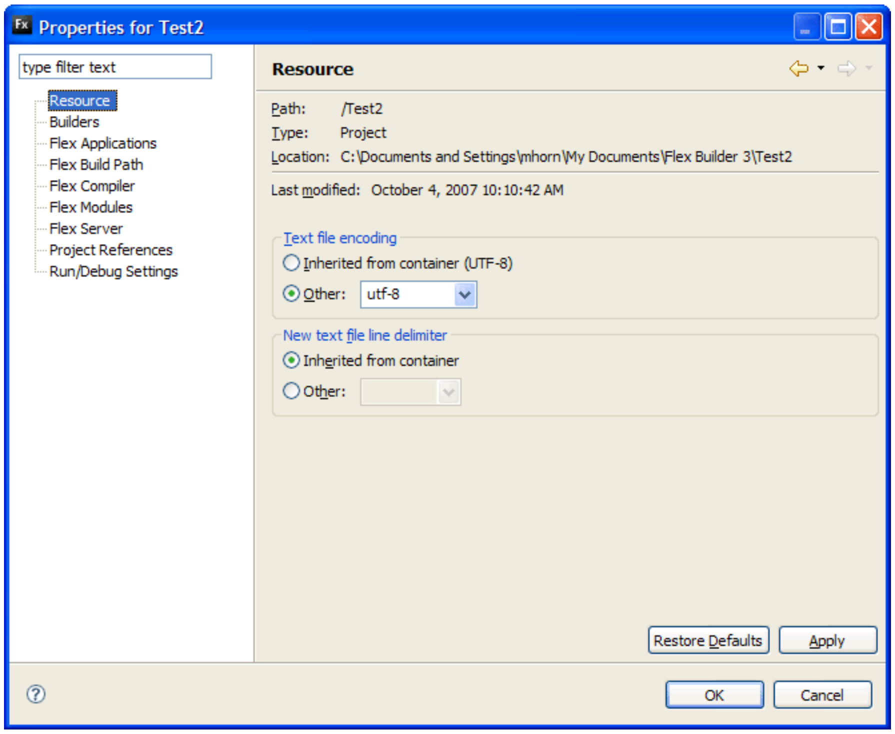
7 (Optional) Click Next to set the build paths for your new project, and then click Finish.

Managing projects

You use the Flex Navigator view to add and import resources into projects, export projects, and move and delete resources.

Setting Flex project properties

Each Flex project has its own set of properties. To set these properties, select the project in the Flex Navigator view. Then select Project > Properties from the main menu, or right-click (Control-click on Macintosh) to display the context menu and select Properties.



Path: /Test2
Type: Project
Location: C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\Test2

Last modified: October 4, 2007 10:10:42 AM

Text file encoding
( ) Inherited from container (UTF-8)
[✓ Other:] utf-8

New text file line delimiter
[✓ Inherited from container]
( ) Other:

You can set the following project-specific preferences in Flex 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 Flex Builder. You can use Apache Ant (an open-source build tool) to create build scripts or import existing Ant build scripts. (See "Customizing builds with Apache Ant" on page 131.)

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 "Managing project application files" on page 40.)

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 "Setting up a project output folder" on page 124 and "Building projects manually" on page 128.)

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 128.)

Flex Modules Specifies modules to build and optimize for the project. For more information about using modules in Flex Builder, see "Creating modules in Flex Builder" on page 147.

Flex Server Determines the location of the Flex root folder and the Flex root URL (for LiveCycle Data Services projects only), and validates location.

Project References Lists the projects that the current project references.

Run/Debug Settings Manages launch configuration settings.

Importing projects

Flex Builder provides wizards to guide you through steps to import projects. You can work with many projects simultaneously. All projects in the current workspace are displayed in the Flex Navigator view.

You can import existing projects into the workspace or create new projects. Existing projects must be valid Flex Builder projects and reside either in another workspace or, if removed from a workspace, in the file system. You can also work with a project that is not currently in your workspace. The project may be packaged in a single .zip file, or a complete project folder.

Import .zip project (stand-alone configuration)

If the .zip project was created with Export Flex Project Archive:

1 Select File > Import > Flex Project.

2 In the Import Flex Project dialog box, select the .zip file you want to import. If the project is compiled on the server (ColdFusion with LiveCycle Data Services or J2EE with/without LiveCycle Data Services), the project location must be the root of LiveCycle Data Services. If a project is not compiled in Flex Builder, you must specify a custom path, ideally under the server's web root. You can import any Flex, AIR, ActionScript, or Library project.

3 Click Finish.

Import .zip project (plug-in configuration)

If the .zip project was created with Eclipse's Export Archive File:

1 Select File > Import > General > Existing Projects into Workspace. Click Next.

2 In the Import Projects dialog box, enter the root directory or archive file paths as described above.

3 Click Finish.

Import complete project folder

If you have a project that was downloaded from source control or a different workspace:

1 Select File > Import > General > Existing Projects Into Workspace and click Next.

2 In the Import Projects dialog box, select the root directory or archive file option; then select Browse to navigate to the project location.

You can import the following archive file types: jar, zip, tar, tar.gz, and tgz.

All valid projects that are available in the specified location are listed in the dialog box.

3 Select one or more projects, and click Finish.

***Note:** Importing a project into a workspace creates a link from the workspace to the existing location of the project.*

Import folder(s) containing other source files or assets

If you want to import source files or assets not in an actual Flex Builder project, use the New Flex Project wizard:

❖ Create a new project at a specific location and set source and output folder settings to match existing folder structure.

or

❖ Create a new project in a different location and move source files to the new project folder structure.

Exporting projects

Flex Builder provides wizards to guide you through the steps to export a project to an archive file (.zip). You can easily share .zip files with other developers who use Flex Builder by posting the archive file on a web site or attaching it to a Jira bug report.

1 Select File > Export.

2 In the Export wizard, select File > Flex > Export Project, then click Next.

3 In the Export Project dialog box, select a project then enter or browse to enter the location where the .zip file will be exported.

4 Click Finish to export your project to the designated location.

For server projects, the paths to the Flex output folder and/or server root are replaced with Eclipse variables. When you export a project, Flex Builder opens readme_flex_export.txt listing which paths were replaced. This readme file is located in the in the .zip archive.

For more information about Eclipse variables and linked resources, see the Eclipse documentation.

Exporting Adobe AIR application installer

For AIR projects, a production build creates a digitally signed AIR file, which users can install before running the application. This process is similar to creating an installer .exe for a typical native application. Optionally you can create an unsigned intermediate package which you can sign later before release. Before using Export Release Build you should decide how to digitally sign your AIR application:

- Sign the application using a VeriSign or Thawte digital certificate
- Create and use a self-signed digital certificate
- Add a timestamp (a timestamp is an assertion from a timestamp authority that the digital certificate was valid when the timestamp was issued). Note that AIR disallows installation if the certificate has expired and there is no timestamp.
- Choose to package the application and sign it later

Digital certificates provided by VeriSign and Thawte give users some assurance as to your identity as a publisher and verification that the installation file has not been altered since you signed it. Self-signed digital certificates serve the same purpose but they are not validated by a third party. You can also package your AIR application without a digital signature by creating an intermediate AIR file (.airi). An intermediate AIR file is not valid because it cannot be installed. Instead, developers can use it for testing and then it can be launched using the AIR ADT command line tool. This capability is provided because in some development environments digital signing is handled by a particular developer or team, which ensures an additional level of security.

1 Select Project > Export Release Build.

If you have multiple projects and applications open in Flex Builder, select the AIR project you want to package.

2 Choose the export settings for project and application.

- If your project does not have a server web root associated with it, all assets are copied to the *project_name* folder, which is the default location.

- If your project has server web root associated with it (for example, PHP and J2EE), all assets are copied to the *web_root/project_name*-debug folder.
- If you want users to view source code, select Enable View Source.
- Click Choose Source Files to select files to you want to publish, then click OK.
- Click Next.

3 On the Digital Signature page:

Specify the digital certificate that represents the application publisher's identity. To generate a self-signed certificate, click Create to enter data in required fields.

If you want to export a file that will be signed later, you can export an intermediate AIRI file.

4 In the AIR File Contents page, select the output files to include in the AIR or AIRI file.

5 Click Finish.

For more information about Adobe AIR files, see *Developing AIR Applications with Adobe Flex 3*.

Moving 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 "Deleting projects" on page 39). After you remove a project from one workspace you can import it into another.

Deleting projects

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 "Closing and opening projects" on page 39.

1 In the Flex Navigator view, 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.

Closing and opening 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 Flex Navigator view. A closed project requires less memory than an open project, and is excluded from builds. You can easily reopen the closed project from the Flex Navigator view.

1 In the Flex Navigator view, select the project to close or reopen.

2 Right-click (Control-click on Macintosh) to display the context menu and select Close Project or Open Project.

Switching 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 into a separate SWF file, see "Managing project application files" on page 40.

1 In the Flex Navigator view, select the MXML application file that you want to make the main application file.

2 Right-click (Control-click on Macintosh) to display the context menu and select Set as Default Application.

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

Managing project application files

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

For example, you might have a complex Flex 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 Flex Navigator view, select a project.

2 Select Project > Properties from the main menu or right-click (Control-click on Macintosh) to 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.

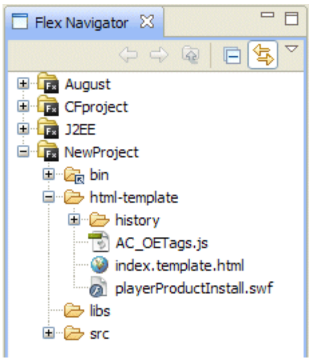
Managing project resources

Projects consist of resources (folders and files) that you can manage from the Flex Navigator view. Projects are contained within a workspace, which is a reflection of the file system. The Flex Navigator view is refreshed each time you add, delete, or modify a resource.

You can also edit project resources outside Flex Builder and the Flex Navigator view, directly in the file system.

Creating folders and files in a project

You can add folders and files to your project as needed. For example, you might create a folder to store all of your data models or to organize all the assets that make up the visual design of your application, as the following example shows:



Create a folder

1. In Flex Navigator view 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 new folder in the source path folder, it is treated like a package name and you can place source files inside that will be recognized by the compiler.

   If you create the folder outside of 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.

Create a file

1. In the Flex Navigator view, 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 "Linking to resources outside the project workspace" on page 42.

Deleting 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 "Linking to resources outside the project workspace" on page 42). 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 Flex Navigator view, 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.

Moving resources between projects in a workspace

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

1 In the Flex Navigator view, 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.

***Note:** You can move both normal resources and linked resources. For information about linking resources, see "Linking to resources outside the project workspace" on page 42.*

Refreshing resources in the workspace

As you edit, add, or delete resources in Flex Builder, 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 Flex Navigator view.

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

By default, in the stand-alone configuration of Flex Builder, the workspace is refreshed automatically. This option is available in the Flex Builder preferences dialog box, which you can access by selecting Window > Preferences > General > Workspace. You can also change the Flex Builder default behavior so that it never refreshes the workspace automatically.

Manually refresh the workspace

❖ In the Flex Navigator view, right-click (Control-click on Macintosh) and select Refresh from the context menu. All project resources in the workspace are refreshed.

Turn off the automatic refresh preference

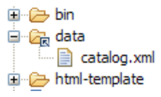
1 Open the Preferences dialog and select General > Workspace.

2 Deselect Refresh Automatically.

Linking 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 Flex Navigator view (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 "Deleting folders and files" on page 41.

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

Link to resources outside the project workspace

1 In the Flex Navigator view, 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.

Using 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 "Creating a path variable" on page 125.

1 In the Flex Navigator view, 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.

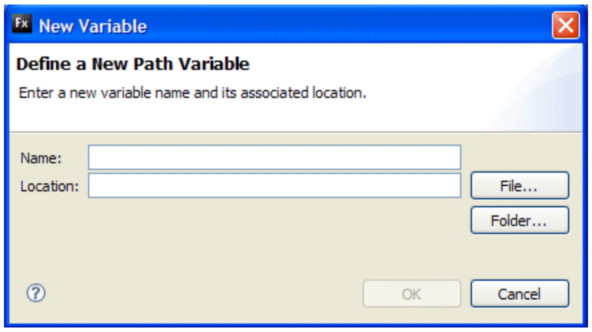
5 Select Link to folder in the file system. Click the Variables button.

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.



New Variable

Define a New Path Variable

Enter a new variable name and its associated location.

Name:
Location:

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 Flex Builder workbench preferences (Open the Preferences dialog and select Preferences > General > Workspace > Linked Resources).

Adding resource folders to the project source path

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 Flex Navigator view.
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 and 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 Flex Navigator view.

Alternatives to using project references

Project references can impact build order, so Flex Builder provides alternatives to using project references.

Flex Library projects The preferred way to create a reusable library. Flex 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 Flex 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 Flex Navigator view.

Viewing resource properties

When you work in the Flex Navigator view, you can select a resource and view its properties.

1. In the Flex Navigator view, select a resource.
2. Select File > Properties or press Alt+Enter (Option+Enter on Macintosh).

About ActionScript projects

Flex Builder lets you create ActionScript projects that use the Flash API (not the Flex framework). This leverages the Flex Builder workbench tools and the ActionScript editor, which means that you have a full-featured IDE for developing ActionScript applications.

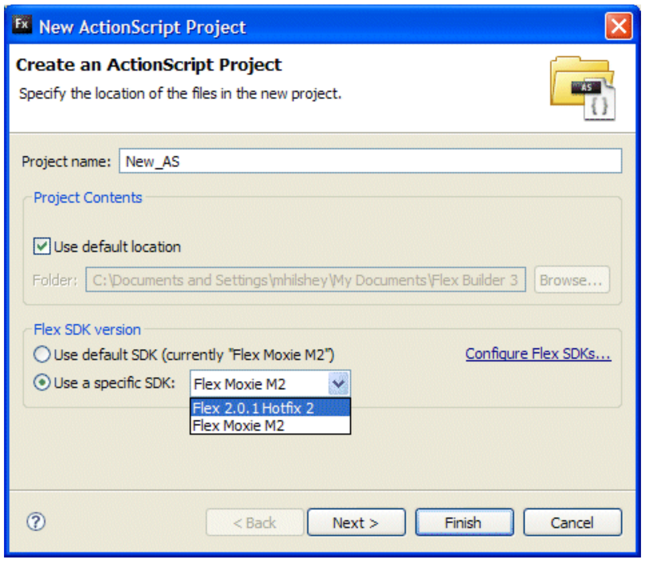
ActionScript projects do not have a visual representation in Flex Builder; in other words, there is no Design mode for ActionScript applications. You view your ActionScript applications by compiling them in Flex Builder and then running them in Flash Player. You can use all the debugging tools.

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.

Creating ActionScript projects

When you create an ActionScript project, the New ActionScript Project wizard guides you through the steps, prompting you for the type of project to create, the project's name, location, and other advanced options.

1 Select File > New > ActionScript Project.



Create an ActionScript Project

Specify the location of the files in the new project.

Project name: New_AS

Project Contents

☑ Use default location

Folder: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3

Flex SDK version

○ Use default SDK (currently "Flex Moxie M2")

● Use a specific SDK: Flex Moxie M2

Flex 2.0.1 Hotfix 2

Flex Moxie M2

2 Enter a Project name, and then specify the following:

Project Location The default location is the workspace, which is My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myASApp*). You can choose a different project location by deselecting the Use Default Location option. On the Macintosh, the default workspace location is /Users/*Flex Developer*/Flex Builder 3/*project_name*.

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.

3 Click Next to set the following advanced options (otherwise, click Finish):

Source Path Specifies the path to link external resources to your application. For example, if you have a folder of shared ActionScript classes, you can link to that folder by adding it to the source path.

Library Path Specifies the path to link external resource libraries (SWC files). By default, the library path of new ActionScript projects contains the playerglobal.swc and utilities.swc files.

Main Source Folder Specifies, by default, the root of your project. You can, however, choose a different folder within the project. You can browse the project folder structure and create a folder for the source if needed.

Main Application File Specifies the name of the ActionScript file that is the main application file. By default, Flex Builder uses the project name as the main application filename. You can change this name.

Output Folder Specifies the location of the compiled application files. By default, this is the *bin* folder, but you can change this.

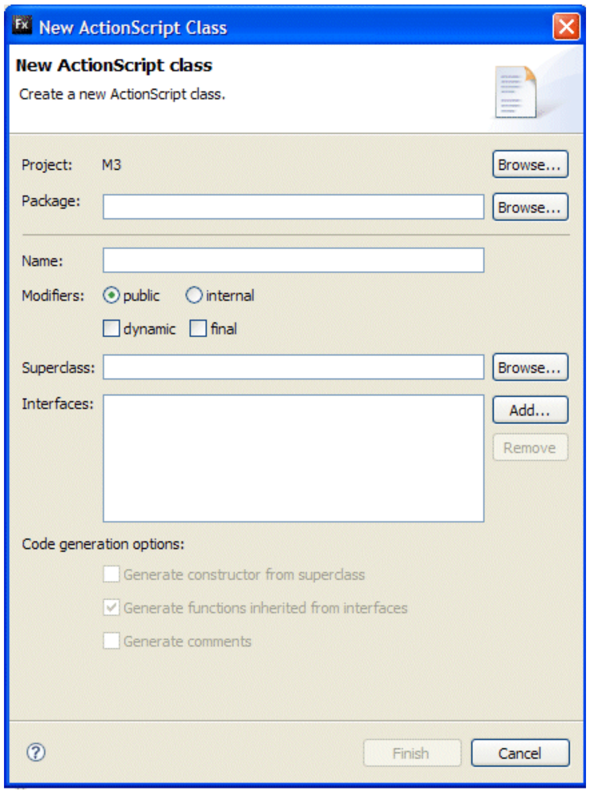
Output Folder URL Specifies the server location of the compiled application files. This is optional.

4 When you finish entering the ActionScript project settings, click Finish.

Creating an ActionScript class

You can use a wizard in Flex 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.



Project: M3
Package:
Name:
Modifiers: [✓ public] internal
dynamic final
Superclass:
Interfaces:
Code generation options:
Generate constructor from superclass
[✓ Generate functions inherited from interfaces]
Generate comments

2 Specify the basic properties of your new class in the dialog box, and then click Finish.

After clicking Finish, Flex 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, Flex 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 67.

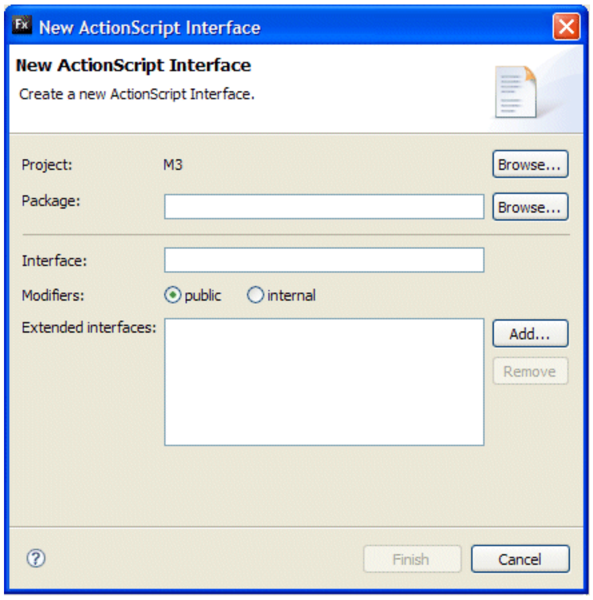
3 Write the definition of your ActionScript class.

For more information, see "Simple Visual Components in ActionScript" on page 105 in *Creating and Extending Adobe Flex 3 Components*.

Creating an ActionScript interface

You can use a wizard in Flex 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.



New ActionScript Interface

Create a new ActionScript Interface.

Project: M3

Package:

Interface:

Modifiers: [✓ public] internal

Extended interfaces:

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.

About 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 Flex applications. 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” on page 645 in the *Adobe Flex 3 Developer Guide*.

Libraries are useful if you create components entirely in ActionScript and use them in Design mode in Flex 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.

Configuring 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 run time.

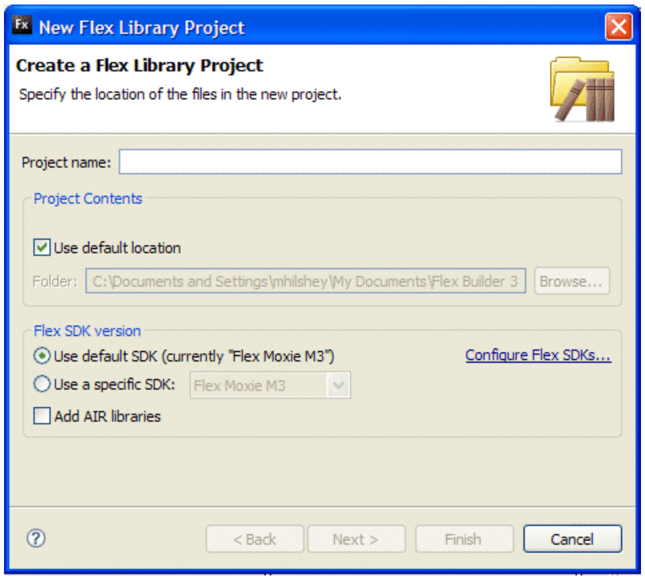
Runtime Shared Library In Flex projects only, you can also use SWC files as a Runtime Shared Library (RSL), which is similar to a dynamically linked library on other platforms. Use SWC files as an RSL when you have a collection of components that are used by more than one application.

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 start-up 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 "Using Runtime Shared Libraries" on page 195 in *Building and Deploying Adobe Flex 3 Applications*.

Creating 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. 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. The first step in creating a SWC file in Flex Builder is to create a Flex Library project.

1 Select File > New > Flex Library Project.


New Flex Library Project

Create a Flex Library Project

Specify the location of the files in the new project.

Project name:

Project Contents

[✓] Use default location

Folder: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3

Flex SDK version

(●) Use default SDK (currently "Flex Moxie M3")

( ) Use a specific SDK: Flex Moxie M3

[ ] Add AIR libraries

2 Enter a Project name, and then specify the following:

Project Location The default location is the workspace, which is My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myLibrary*). You can choose a different project location by deselecting the Use Default Location option. On Macintosh, the default workspace location is /Users/*Flex Developer*/Flex Builder 3/*project_name*.

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. Flex 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 Flex 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 "Using SWC files in your projects" on page 50.

5 When you finish entering the project settings, click Finish.

Adding 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 "Linking to resources outside the project workspace" on page 42.)
- Add a linked folder that contains components to the library project's source path. (See "Deleting folders and files" on page 41.)

***Note:** All the components you include in the library project must be associated with the library project (directly or as linked resources).*

Selecting library project elements to include in the SWC file

The next step in creating a library SWC file is to select the elements (components and resources) to include in the SWC file when it is built by the compiler.

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.

Building 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
```

Flex Builder creates a directory in the project named myOpenDir and stores the contents of the SWC file in that directory.

Using 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 run time.

Add an SWC file to the library path

1. With a project selected in the Flex Navigator view, select Project > Properties > Flex Build Path.
2. Click on 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 contain 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 Flex Navigator view, select Project > Properties > Flex Build Path.
2. Click on 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 Flex Navigator view, 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

1 With a project selected in the Flex Navigator view, 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 Run-time Shared Library (RSL) option.

5 Enter the URL where the SWC library will reside when the application is deployed.

6 (Optional) To extract the SWF file in the SWC file when it is placed in the deploy location, select the Automatically extract swf to deployment path option.

7 Click OK.

Using the SWC files as an RSL simplifies the process for using RSLs manually. To do this, 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 “Using Runtime Shared Libraries” on page 195 in *Building and Deploying Adobe Flex 3 Applications.*

Chapter 5: Navigating and Customizing the Flex Builder Workbench

The term *workbench* refers to the Flex Builder development environment. 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. The workbench is the container for all the development tools you use to develop applications.

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.*

Topics

Working with perspectives

Perspectives include combinations of views and editors that are suited to performing a particular set of tasks. For example, you normally open the Flex Debugging perspective to debug your Flex application.

For an overview of perspectives, see "About Flex Builder perspectives" on page 13.

Opening and switching perspectives

When you open a file that is associated with a particular perspective, Flex Builder automatically opens that perspective. You can also open a perspective manually. The stand-alone configuration of Flex Builder contains three perspectives:

- Flex Development
- Flex Debugging
- Flex Profiling (available on a trial basis or complete with Flex Builder Professional)

❖ Select Window > Perspective or choose Other to access all other Eclipse perspectives. (In the plug-in configuration of Flex Builder, you select Window > Open Perspective.)

You can also click the Open Perspective button in the upper-right corner of the workbench window, then select a perspective from the pop-up menu.

To see a complete list of perspectives, select Other from the Open Perspective pop-up menu.

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. To open perspectives in a new window, edit preferences.

Setting the default perspective

The default perspective is indicated by the word *default* in parentheses following the perspective name.

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.

Opening perspectives in a new window

You can change the default behavior for opening perspectives to open a perspective 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.

Customizing 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.

You can also configure several other aspects of a perspective, including the File > New submenu, the Window > Perspective > Other submenu, the Window > Other Views submenu, and action sets (buttons and menu options) that appear in the toolbar and in the main menu items. (Menu names differ slightly in the plug-in configuration of Flex Builder.)

Create a new perspective

1 Open an existing perspective.

2 Show views and editors as desired.

For more information, see "Opening views" on page 54, and "Opening files for editing" on page 56.

3 Select Window > Perspective > Save Perspective As (Window > Save Perspective As in the plug-in configuration of Flex Builder).

4 In the Save Perspective As dialog box, enter a new name for the perspective, then click OK.

Configure a perspective

1 Open the perspective to configure.

2 Select Window > Perspective > Customize Perspective (Window > Customize Perspective in the plug-in configuration of Flex Builder).

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 > Perspective > Save Perspective As (Window > Save Perspective As in the plug-in configuration of Flex Builder).

7 In the Save Perspective As dialog box, enter a new name for the perspective and click OK.

When you save a perspective, Flex Builder adds the name of the new perspective to the Window > Perspective menu (Window > Open Perspective in the plug-in configuration of Flex Builder).

Deleting 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.

Resetting perspectives

You can restore a perspective to its original layout after you made changes to 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.

Working with editors and views

Most perspectives in the workbench are composed of an editor and one or more views. An editor is a visual component in the workbench that is typically used to edit or browse a resource. Views are also visual components in the workspace that support editors, provide alternative presentations for selected items in the editor, and let you navigate the information in the workbench.

For an overview of editors and views, see “About the workbench” on page 10.

Opening views

Perspectives contain predefined combinations of views and editors. You can also open views that the current perspective might not contain.

❖ Select Window and choose a Flex Builder view or select Window > Other Views to choose other Eclipse workbench views. (In the plug-in configuration of Flex Builder, select Window > Show View.)

After you add a view to the current perspective, you might want to save that view as part of the perspective. For more information, see “Customizing a perspective” on page 53.

You can also create fast views to provide quick access to views that you use often. For more information, see “Creating and working with fast views” on page 55.

Moving and docking 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. Right-click (Control-click on Macintosh) on the view 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 > Perspectives > Save Perspective As (Window > Save Perspective As in the plug-in configuration of Flex Builder).

Rearranging tabbed views

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

Switching between views

You can switch between views to work in a different view.

❖ Click the tab of the view to switch to.

You can also press Control+F7 (Command+F7 on Macintosh), use the F7 key to select the view to switch to, and then release the Control key.

Creating and working 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 Flex Navigator view to a fast view, and then open a file from the Flex Navigator fast view, the fast view automatically is hidden to allow you to work with that file.*

Create a fast view

❖ 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.

Restore a fast view to normal view

1 Right-click (Control-click on Macintosh) the view's tab to open the view's context menu.

2 Deselect Fast View.

Filtering the Tasks and Problems views

You can filter the tasks or problems that are displayed in the Tasks or Problems views. For example, you might want to see only problems that the workbench has logged, or tasks that you logged as reminders to yourself. You can filter items according to which resource or group of resources they are associated with, 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.

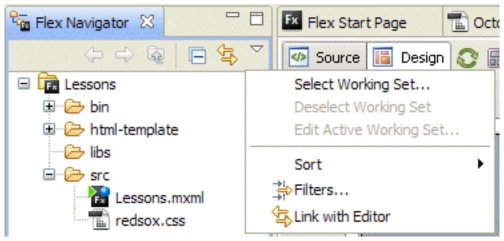
For more information about views, see "Flex Builder Workbench Basics" on page 10.

Creating 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 Flex Navigator and Task views and also search working sets rather than searching everything in the workspace.

Create a working set

1 In the Flex Navigator view, open the toolbar menu and select Select Working Set.



2 Select New.

Flex 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 Flex Navigator view and only those projects and resources contained in the set are displayed.

Display all projects in the workspace

❖ In the Flex Navigator view, open the toolbar menu and choose Deselect Working Set.

Opening files for editing

When you open a file, you launch an editor so that you can edit the file.

❖ Do one of the following:

- Right-click (Control-click on Macintosh) the file in one of the navigation views and select Open from the context menu.
- Double-click the file in one of the navigation views.
- Double-click the bookmark associated with the file in the Bookmarks view.
- Double-click an error warning or task record associated with the file in the Problems view.

This opens the file with the default editor for that particular type of file. To open the file in a different editor, right-click (Control-click on Macintosh) the file, select Open With from the context menu, and select the editor to use.

Associating 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 by right-clicking (Control-clicking on Macintosh) any resource in one of the navigation views and selecting Open With from the context menu.

Editing files outside the workbench

You can edit an MXML or ActionScript file in an external editor and then use it in Flex 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 Flex Builder.

4 In the workbench, right-click (Control-click on Macintosh) the file you edited in one of the navigation views and select Refresh from the context menu.

If you work with external editors regularly, you might want to enable auto-refresh. To do this, 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 this 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 will appear when you release the mouse button.

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

Maximizing a view or editor

You can temporarily maximize a view or editor so that it fills the workbench window.

Maximize a view or editor

❖ Right-click (Control-click on Macintosh) the view or editor’s title bar and select Maximize.

Restore a view or editor to its previous position and size

❖ Right-click (Control-click on Macintosh) the view or editor’s title bar and select Restore.

You can also maximize or restore a view or editor by double-clicking the title bar or by clicking the Maximize/Restore icons in the upper-right corner.

Switching the workspace

You can work in only one workspace at a time. When you install and run Flex 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 Flex Builder or by selecting File > Switch Workspace.

Customizing 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.

Rearranging the main toolbar

Flex 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 by right-clicking (Control-clicking on Macintosh) the toolbar and deselecting Lock the Toolbars.

2 Move the mouse pointer over the thick vertical line that is on the left side of the toolbar section you want to rearrange.

3 Click and hold the left mouse button (mouse button on Macintosh) to grab the toolbar section.

4 Move the section left, right, up, or down, and release the mouse button to place the section in the new location.

To prevent accidental changes, lock the toolbar again by right-clicking (Control-clicking on Macintosh) the toolbar and selecting Lock the Toolbars.

Changing 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.

Changing 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. The workbench lets you configure the following fonts:

Banner font Appears in the title area of many wizards. For example, the New Flex Project wizard uses the Banner font for the top title.

Dialog font Appears in widgets and dialog boxes.

Header font Appears as a section heading.

Text font Appears in text editors.

CVS Console font Appears in the CVS console.

Ignored Resource font Displays resources that CVS ignores.

Outgoing Change font Displays outgoing changes in CVS.

Console font (Defaults to text font) Appears in the Debug console.

Detail Pane Text font Defaults to text font) Appears in the detail panes of Debug views.

Memory View Table font (Defaults to text font) Appears in the table of the Memory view.

Java Editor Text font (Defaults to text font) Appears in Java editors.

Properties File Editor Text font (Defaults to text font) Appears in Properties File editors.

Compare Text font (Defaults to text font) Appears in textual compare or merge tools.

Java Compare Text font (Defaults to text font) Appears in Java compare or merge tools.

Java Properties File Compare Text font (Defaults to properties file editor text font) Appears in Java properties file compare or merge tools.

Part Title font (Defaults to properties file editor text font) Appears in view and editor titles.

View Message font (Defaults to properties file editor text font) Displays messages in the view title bar (if present).

Plug-ins that use other fonts might also provide preferences that allow for customizing. For example, the Java Development Tools plug-in provides a preference for controlling the font that the Java editor uses (In the Preferences dialog, select > General > Appearance > Colors and Fonts > Java > Java Editor Text Font).

The operating system always displays some text in the system font (for example, the font displayed in the Flex Navigator view tree). To change the font for these areas, you must use the configuration tools that the operating system provides (for example, the Display Properties control panel in Windows).

Changing fonts and colors

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 Flex Builder to use the font selected in the Windows Display Properties control panel.*

3 Set the font and color preferences as desired.

Changing colors

The workbench uses colors to distinguish different elements, such as error text and hyperlink text. The workbench uses the same colors that the operating system uses. To change these colors, you can also use the configuration tools that the system provides (for example, the Display Properties control panel in Windows).

Change colors

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 color to change.

3 Click the color bar to the right to open the color picker.

4 Select a new color.

Controlling 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.

Searching in the workbench

Flex Builder provides a search tool that lets you quickly locate resources. For more information about searching for text in a particular file, see "Finding and replacing text in the editor" on page 112.

Searching for files

Flex Builder lets you conduct complex searches for files.

❖ In the plug-in version of Flex Builder select Search > Search or Search > File.

In the stand-alone version of Flex Builder select Edit > Find in Files.

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

Searching for references and declarations

Flex 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, Flex Builder lets you find and mark references or declarations to identifiers in ActionScript and MXML files, projects, or workspaces. For more information, see "Finding references and refactoring code" on page 113.

Using 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 > Other Views > General. (Window > Show View > Other in the plug-in configuration of Flex Builder.)

2 Expand the General category, select Search, and click OK.

Working in the editor's Source and Design modes

The MXML editor in Flex Builder lets you work in either Source or Design mode. You can also use Flex Builder to create a split view so that you can work in both Source and Design modes simultaneously.

View your file in Design mode

❖ Click Design at the top of the editor area.

View your file in Source mode

❖ Click Source at the top of the editor area.

Work in both Source and Design modes simultaneously

1 Right-click (Control-click on Mac OS) the editor's tab and 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.

Switch between the Source and Design modes

❖ Press Control+`(Left Quote).

Accessing keyboard shortcuts

The keyboard shortcuts available to you while working in Flex Builder depend on many factors, including the selected view or editor, whether or not a dialog is open, installed plug-ins, and your operating system. You can obtain a list of available keyboard shortcuts at any time using Key Assist.

❖ Select Help > Key Assist.

Setting workbench preferences

You can set preferences for many aspects of the workbench. For example, you can specify that Flex Builder should prompt 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 Flex applications.

Your Flex 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 yourself, 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 Flex Builder workbench preferences

1. Select Window > Preferences.
2. Select any of the categories of workbench preferences and modify them as needed.
3. Click OK.

Part 3: Developing a Flex Application User Interface

Topics

Chapter 6: Building a Flex User Interface

The Adobe Flex framework consists of a component-based system for building rich Internet applications. You use Adobe Flex Builder to rapidly build user interfaces for Flex applications. There are several options for structuring your user interface.

Topics

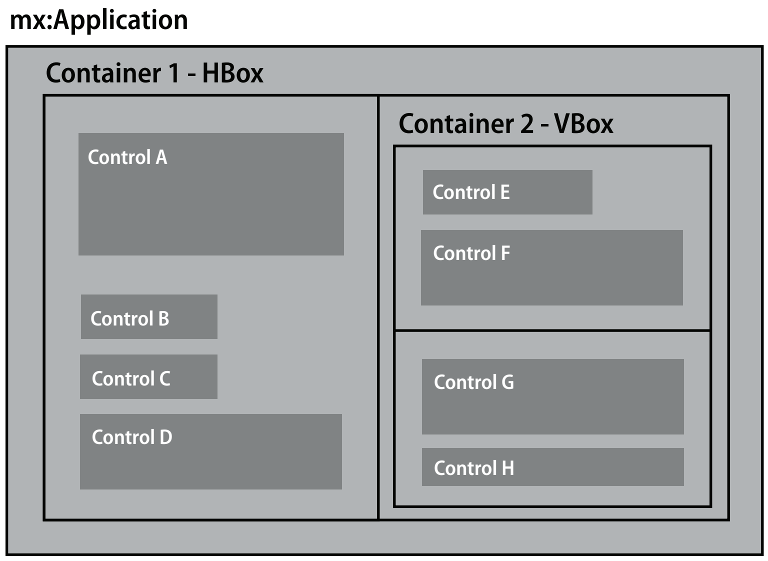
About the structure of Flex user interfaces

The building blocks of Flex user interfaces are MXML containers and controls. A control is a user interface component such as a Button, TextArea, or ComboBox. A container is a rectangular region of the Flash Player drawing surface that you use to organize and lay out controls, other containers, and custom components.

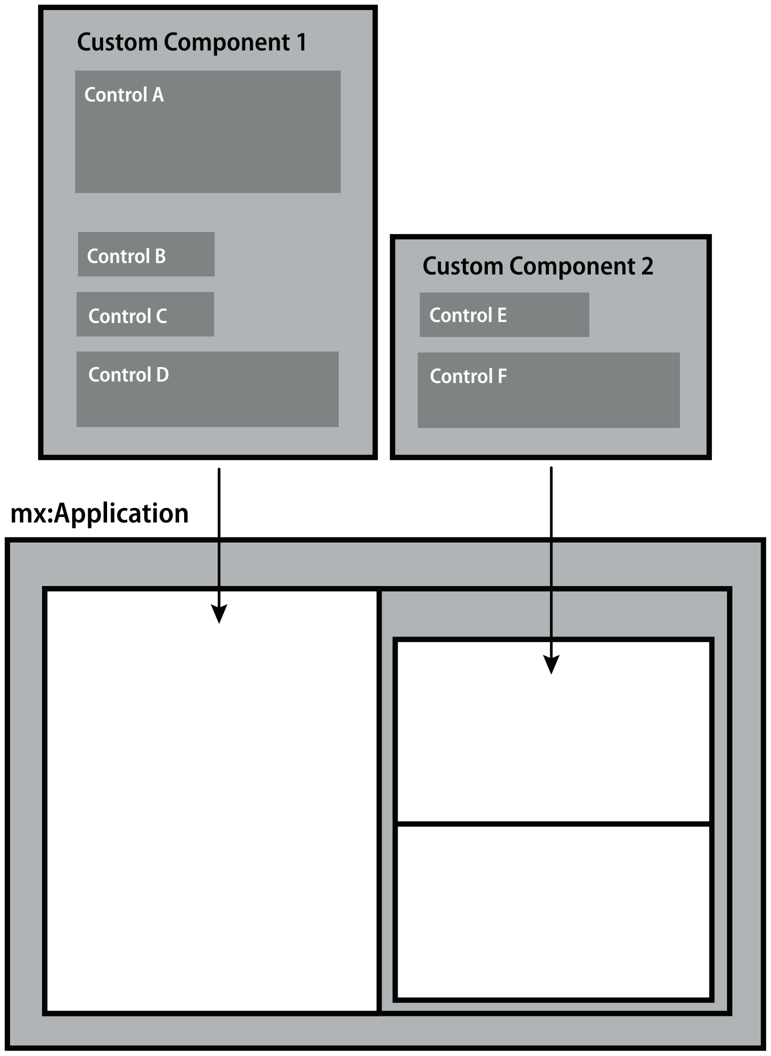
Flex applications typically consist of an MXML application file (a file with an `<mx:Application>` parent tag), and one or more components defined in separate MXML files, ActionScript files, or Flash component files (SWC files). You can insert containers and controls directly in the MXML application file, or you can insert them in separate MXML files to create custom components and then insert the custom components in the application file.

Generally, it is best to structure the main portions of your application inside Panel containers. Most of the Flex controls were not designed to be used directly on the dark application background.

The following example shows a simple structure for a Flex application. The containers and controls are inserted directly into the MXML application file.



The following example shows a component-based structure for a Flex application. You group the controls of the user-interface elements in separate custom component files, which you then insert into the MXML application file.



A component-based structure is useful when your user interface consists of distinct functional elements. For example, your layout could have an element that retrieves and displays a product catalog, another element that retrieves and displays details about any product that the user clicks in the catalog, and an element that lets the user add the selected product to a shopping cart. This user interface could be structured as three custom components, as in the previous example, or as a mixture of custom components and controls inserted directly into the layout.

For more information about components, see “Using Flex Visual Components” on page 88 in the *Adobe Flex 3 Developer Guide*.

Adding and changing components

You use Flex Builder to add, size, position, edit, or delete Flex components, as well as custom components defined in separate MXML and ActionScript files.

Add components in MXML Design mode

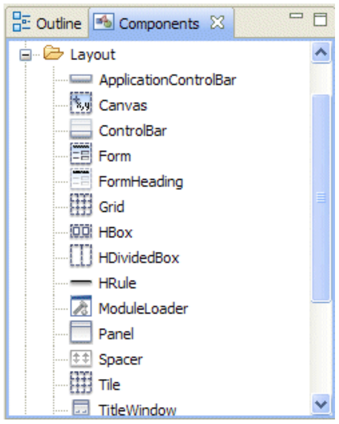
You add standard Flex containers and controls to your user interface in MXML Design mode. You drag and drop components from the Components view to the Design area of the MXML file and position them according to the layout rule of the container. You can also add custom components that you define in separate MXML and ActionScript files and save in the current project or in the source path of the current project.

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.



The components are organized by categories in the view.

The Custom category lists all the custom components that you define in separate MXML and ActionScript files and save in the current project or in the source path of the current project. For example, if you create a component file called EmployeeView.mxml and save it in your project, the EmployeeView component appears in the Custom category. For more information, see "Creating Custom MXML Components" on page 227.

***Note:** The Components view lists only visible custom components (components that inherit from the UIComponent class). For more information, see Adobe Flex Language Reference in Help.*

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.

The default layout rule of an Application, Panel, or TitleWindow container can be overridden by specifying a `layout="absolute"` property. You can then drag and position the component anywhere in the container. If you create an application or a Panel or TitleWindow component file with Flex Builder, the `layout="absolute"` property is included by default.

Add components in complex layouts

1 Drag a component over to the design area in the layout where you want to insert it, and then press the Control key.

2 Drop the component into the highlighted container, or hover over a different area and press the Control key again to see the target container.

Add components by writing code

You can use code hinting to add standard Flex containers and controls to your user interface. In Flex Builder, as in Eclipse, code hinting is called Content Assist.

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.

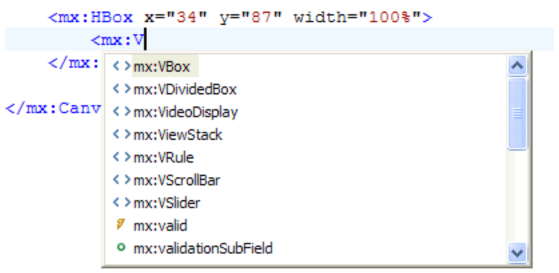
For example, to insert a VBox container inside an HBox parent container, place the insertion point after the opening `<mx:HBox>` tag:

```
<mx:HBox>
    insertion point here
</mx:HBox>
```

3 Enter the component tag.

As you enter the tag, a pop-up menu appears suggesting possible entries.

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



```
<mx:HBox x="34" y="87" width="100%">
    <mx:V
</mx:
</mx:Canv
```

In addition to the standard Flex components, the pop-up menu lists the 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. For more information, see "Creating Custom MXML Components" on page 227.

Adding Flash components (SWC files)

You can add Flash components (SWC files) to your user interface either visually or by writing code.

Note: *Adobe Flash CS3 Professional creates applications compatible with Adobe Flash Player 9. Adobe Flex applications also support Flash Player 9, which means that you can import assets from Flash CS3 Professional to use in your Flex applications. You can create Flex controls, containers, skins, and other assets in Flash CS3 Professional, and then import those assets into your Flex application as SWC files. Before you can create Flex components in Flash CS3, you must install the Flex Component Kit for Flash CS3. For more information, see the article Importing Flash CS3 Assets into 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 Flex Navigator view 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 "Building projects manually" on page 128.

The library path can also be defined in the flex-config.xml configuration file in Adobe LiveCycle Data Services ES.

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.

Working with components visually

Flex Builder lets you work with components visually in the MXML and CSS editors so you can see what your application looks like as you build it. The MXML and CSS editors have two modes: Source mode for writing code, and Design mode for developing applications visually.

Using the MXML editor in Design mode

You can set the size of the design area in Design mode. This lets you preview how the layout of your application or component will look at different sizes. You can also select, pan, move, resize, scroll, and magnify items in the design area.

View an MXML file

1 If the MXML file is not already open in the MXML editor, double-click the file in the Flex Navigator view to open it.

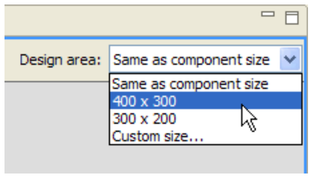
2 If the MXML editor displays source code, click Design at the top of the editor area.

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, select Window > Preferences, then Flex> Editors > Design Mode, then select the Automatically Show Design-related Views option.

Set the size of the design area

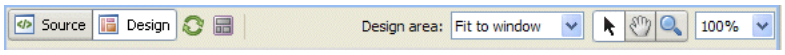
1 Select a size from the Design area pop-up menu on the editor's toolbar.



- If the size of your layout is larger than the editor window, the editor displays scrollbars to preserve the layout at the set size.
- If you select Fit to Window, the editor doesn't display scrollbars. Instead, it adjusts the layout (if possible) to fit the window size.
- If you specify a size for the Application container, the size overrides the View As settings for that dimension.

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. Click and drag a component to the desired place. You can also drag to resize and click to select.



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 on 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 on the toolbar or press Z from the keyboard. A plus symbol cursor will appear 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.

Selecting multiple components in an MXML file

You can select more than one component in an MXML file. This 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.

Deselecting multiple components

- Click the background container.
- Click an unselected component.
- Click in the gray margin around the root component.

Positioning components

You can change the position of components visually depending on the layout rules of the parent container. The properties of the parent container can also affect the position of child components. You can also dynamically position components by using a constraint-based layout. For more information, see "Setting layout constraints for components" on page 87.

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 the layout rules of the parent container. For example, if you move a VBox container in an HBox container, the VBox container is positioned into the horizontal arrangement with the other child containers (if any).

If the container has absolute positioning, you can drag and position components anywhere in the container. A container has absolute positioning if it is a Canvas container or an Application, Panel, or TitleWindow container with a `layout` property set to `absolute`. The `layout="absolute"` property overrides the container's default layout rule. For more information, see "Using Layout Containers" on page 373 in the *Adobe Flex 3 Developer Guide*.

2 In Design mode, select the component's parent container and edit the component's layout properties in the Flex 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 Tile container to set the spacing between child components and the `direction` property to specify either a row or column layout.

Sizing components

You can dynamically size components in an MXML file visually in the design area in a constraint-based layout. In the design, you can anchor one or more sides of a component to the edges of the component's container or the container's constraint regions. You can also change the size of a component in an MXML file by selecting menu options or by editing its properties in the Flex Properties view. For more information, see "About constraint-based layouts" on page 85.

Size a component visually

❖ In the MXML editor's Design mode, click on 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 Flex Properties view (Window > Flex Properties), set the `height` or `width` property of the selected component or components.

The Flex 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.

Note: *The Flex Properties view appears only when the MXML editor is in Design mode.*

3 Press Tab or Enter, or click outside the view to apply your last change.

Using snapping to position components

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

Note: *A container has absolute positioning if it is a Canvas container or if it has a* `layout` *property set to* `absolute`*. The* `layout="absolute"` *property can be used only with the Application, Panel, and TitleWindow containers. It overrides the container's layout rule and lets you drag and position components anywhere in the container.*

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

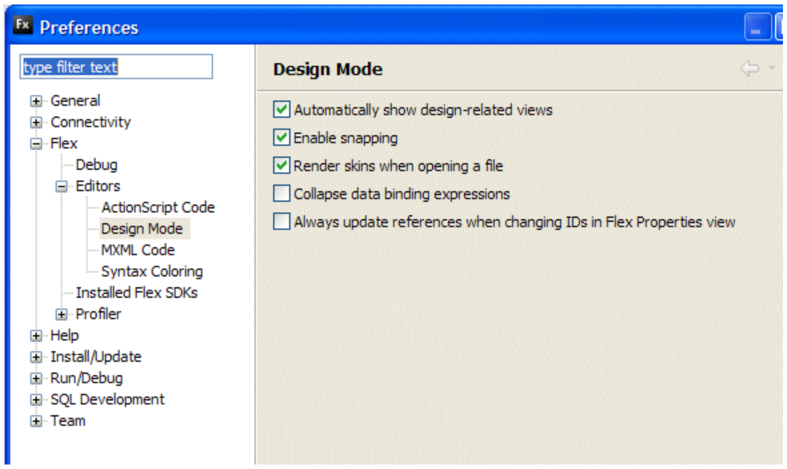
Enable or disable snapping

❖ 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 Select Window > Preferences from the main menu.

2 Select Flex> Editors > Design Mode in the sidebar of the Preferences dialog box.



☑ Automatically show design-related views
☑ Enable snapping
☑ Render skins when opening a file
☐ Collapse data binding expressions
☐ Always update references when changing IDs in Flex Properties view

3 Select or deselect the Enable Snapping option.

Aligning components

You can visually align components relative to each other in a container that has absolute positioning.

Note: *A container has absolute positioning if its* `layout` *property is set to* `absolute`*. Only Application, Canvas, and Panel containers can use the* `layout="absolute"` *property. For Canvas containers, this attribute is the default; for Application and Panel containers, you must specify* `layout="absolute"` *explicitly. This parameter overrides the container's layout rule and lets you drag and position components anywhere in the container.*

You can also center components in a container by using a constraint-based layout. For more information, see "Setting layout constraints for components" on page 87.

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 "Selecting multiple components in an MXML file" on page 70.

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 that of the first component you selected.

Align Vertical Centers Positions all selected components so that their vertical centerlines are aligned with the vertical centerline of the first component you selected.

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

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

Align Horizontal Centers Positions all selected components so their horizontal centerlines are aligned with the horizontal centerline of the first component you selected.

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

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

For objects with no layout constraints, Flex 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, Flex 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, Flex 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.

Nudging 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.

Setting component properties

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

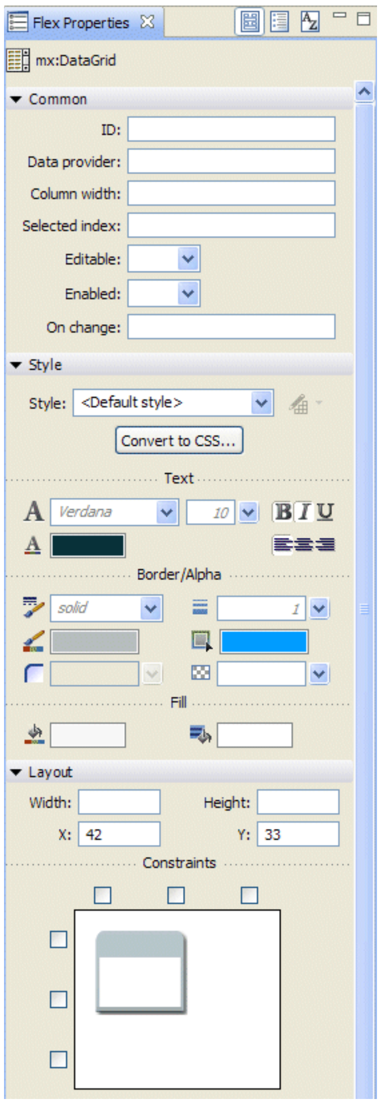
Change text in the ID field

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

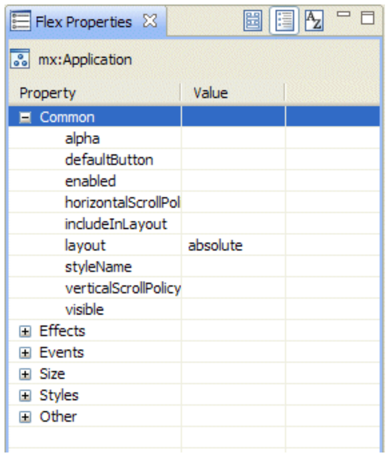
1. Select Window > Preferences from the main menu.
2. Select Flex > Editors > Design Mode.
3. Select or deselect Always Update References When Changing IDs in the Flex Properties View.

Set other properties of a component

❖ Select the component and set its properties in the Flex Properties view (Window > Flex Properties).



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



Property	Value
Common	
alpha	
defaultButton	
enabled	
horizontalScrollPol	
includeInLayout	
layout	absolute
styleName	
verticalScrollPolicy	
visible	
Effects	
Events	
Size	
Styles	
Other	

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

Showing surrounding containers

You can show surrounding containers in the MXML editor's Design mode to better visualize the containers in your layout and to more easily insert or select containers in complex layouts.

1 Select a container in the layout.

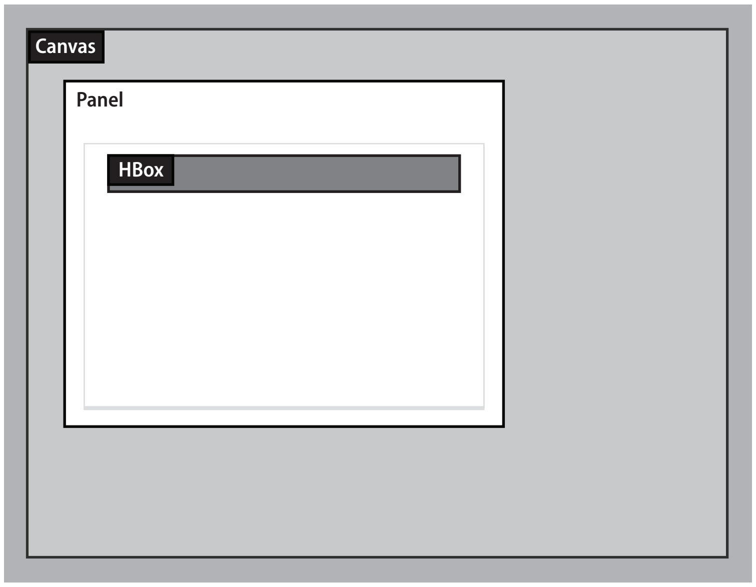
   If it is too difficult to select a container, select a control inside it instead.

2 Press the F4 key.

   If you selected a control, select the parent container now. It should be easy to see.

   Semitransparent overlays appear showing all the containers around the selected container, expanded outward slightly so there is room to insert more components into them. Overlays also appear showing all the children containers.

When you select the Panel container and press F4 in the following example, Flex Builder displays overlays for the panel's parent container (Canvas) and child component (HBox).



The overlays change if you select another container.

Show a container's parents and immediate children

❖ With surrounding containers turned on, click the container.

Move a container into another container

❖ Drag the container overlay over another container, wait for the target container to become highlighted, and then drop the container.

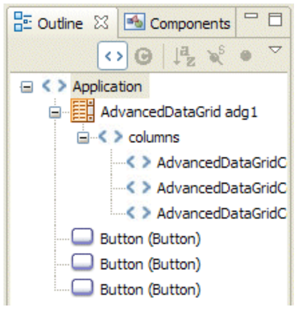
Dismiss the surrounding containers

❖ Press the F4 key again.

Inspecting the structure of your MXML

You 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.

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



❖ In Outline view, click to select a single component or Control-click (Command-click on Macintosh) to select multiple components.

Hiding container borders

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

❖ Select Design > Show Container Borders.

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

Copying components to other MXML files

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

1 Make sure the two MXML files are open in the MXML editor's Design mode.

2 Select the component in one file and press Control+C (Command+C on Macintosh) to copy it.

3 Switch to the other file, click inside the desired container, and press Control+V (Command+V on Macintosh) to paste the component or components into the container.

Deleting components

You can visually delete components from your user interface.

❖ Select the component and press the Delete key on your keyboard, or right-click (Control-click on Macintosh) the component and select Delete from the context menu.

Applying styles and skins

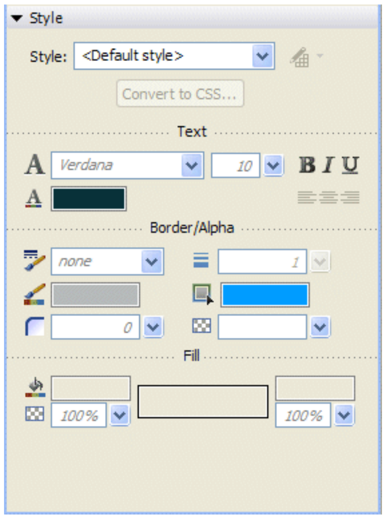
Styles affect the appearance of components. They alter visual parameters such as border thickness, or replace the entire look of a component with a new image (skin). You can set style properties inline on an MXML tag or separately using CSS code.

When you apply inline styles to components, you can convert component styles into a CSS rule in an external stylesheet. You can also move from the MXML editor to the CSS editor when external styles are used.

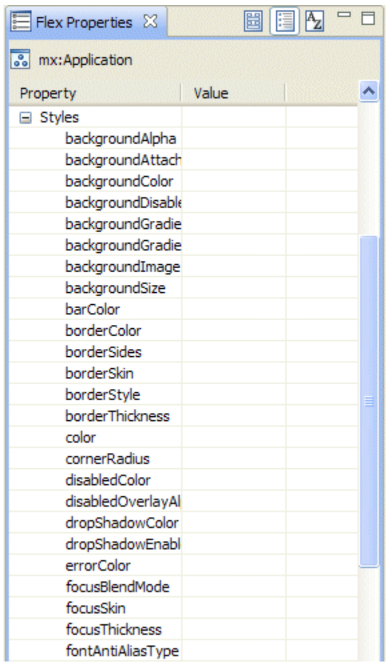
Apply inline styles to a component

1 With your MXML file open in Design mode, click the component to select it.

2 Enter the style property values in the Flex Properties view.



In Category view, the Styles category lists the styles that can be applied to the selected component.



Styles
backgroundAlpha
backgroundAttach
backgroundColor
backgroundDisable
backgroundGradie
backgroundGradie
backgroundImage
backgroundSize
barColor
borderColor
borderSides
borderSkin
borderStyle
borderThickness
color
cornerRadius
disabledColor
disabledOverlayAl
dropShadowColor
dropShadowEnabl
errorColor
focusBlendMode
focusSkin
focusThickness
fontAntiAliasType

***Note:** Multiword style names in Flex can be written either like an ActionScript identifier (for example, fontFamily) or like similar HTML styles (for example, font-family).*

Apply an external or embedded style to an application

1 Make sure you import the external style sheet or embed styles in your MXML file.

For example, the following expression imports an external style sheet called styles.css:

```
<mx:Style source="styles.css"/>
```

The following expression embeds two styles in the MXML file using CSS code:

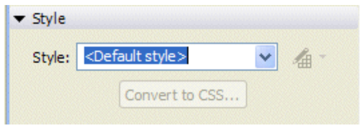
```
<mx:Style>
    .myclass { color: Red } /* class selector */
    Button { fontSize: 10pt; color: Yellow } /* type selector */
</mx:Style>
```

For styles such as `Button`, the style is automatically applied to all matching components.

For styles such as `.myclass` follow these steps:

2 Click the component in the MXML editor's Design mode to select it.

3 Apply the desired style by selecting it from the Style pop-up menu in the Flex Properties view.

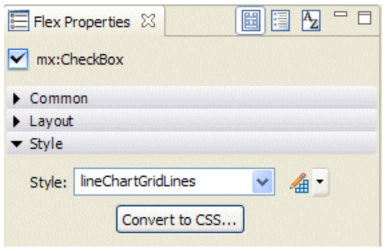


The Style pop-up menu lists the styles defined in the external style sheet or embedded in the current file.

Convert to CSS

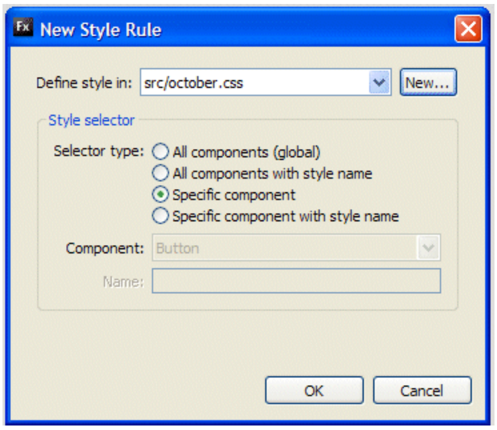
1 From your MXML file, click a component in the design area, then apply style values from the Flex Properties view.

2 Click the Convert to CSS button.



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 create one. Then select the type of the style rule you want to create, which determines the components it will affect.



5 Click OK.

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 and in the Flex Properties view.

2 Click the Edit Style Rule button next to the Style pop-up menu, then select the style you want to edit.

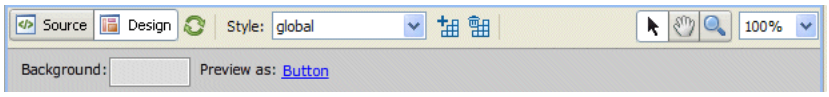
The CSS file opens in the CSS editor's Design mode. You use the Flex Properties view to make further changes. You can also modify your CSS in Source mode.

Using the CSS editor in Design mode

The CSS Design mode editor allows you to visually display and edit the contents of a CSS file. As with the MXML editor, you use the Flex Properties view to edit styles. The toolbar gives you quick access to common tasks such as creating and deleting styles, and pan/view. You can also apply skins for Flex components in CSS Design mode.

Create new style

1 Click the New Style button next to the Style pop-up menu on the CSS Design editor toolbar.



2 In the New Style dialog box choose a Selector Type option for the style to be created.

The type selected determines which components the styles will be applied to. If there is a specific component affected by the new style rule, select a component from the pop-up menu.

When the style is selected, it is previewed in the design area. Use the Flex Properties view to make further changes to your CSS.

***Note:** You can also modify your CSS in Source mode.*

For more information, see "Using Cascading Style Sheets" on page 479 in the *Adobe Flex 3 Developer Guide*.

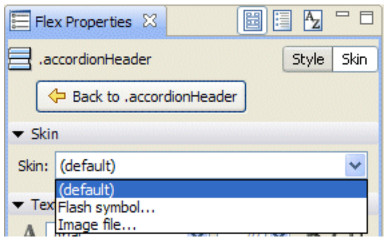
Edit styles and skins

In CSS editor's Design mode you can manipulate CSS and skin styles for Flex components.

1 In the Flex Properties Standard view, click the Style or Skin buttons to alter your CSS style or skins.

2 Choose from the controls and pop-up menus to modify your CSS or skin styles.

3 To use graphical skins, click the Skin button and then choose Image File or Flash Symbol from the pop-up menu. The supported formats are the same as the Import Skin Artwork wizard.



Edit scale grids

When an image file is applied as a skin, the Edit Scale Grid button appears in the upper-right corner of the design area. The scaling grid allows you to visually resize image skins.

1. Click the Edit Scale Grid button.

   The preview switches to editing mode with dotted lines representing the grid.
2. Drag the lines to reposition the grid.
3. Click the Edit Scale Grid button again to leave editing mode.

For more information, see “Using 9-slice scaling with embedded images” on page 784 in the *Adobe Flex 3 Developer Guide*.

Edit content area

When you apply skins to a sub-class of a container (for example, VBox, Panel, Canvas) you may need to adjust the region where the container lays out its child Flex components.

1. Select Design > Show Content Area.
2. Drag the resize handles to resize the content area.

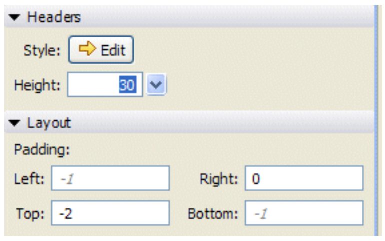
Edit sub-parts of a component

When the selected component has an external style set on it, an Edit button appears next to the Style field. Some components contain entire built-in sub-components with their own style and skin properties. For example, Accordion has section header buttons and List contain a VScrollBar.

❖ While viewing the parent component, click the Edit button.

  Flex Builder automatically generates CSS code for both the parent component and the sub-part.

  After viewing the sub-part, click the Back button to return to the parent component.



Style: Edit
Height: 30
Padding:
Left: -1 Right: 0
Top: -2 Bottom: -1

Default values

Most style properties that are not explicitly set on the selected item will still have a default value that is inherited through the CSS style chain. For example, ToggleButtonBar inherits any styles applied to ButtonBar.

The Flex Properties view displays default, inherited values for any style properties that are not explicitly set on a selected item. To distinguish default values from values that are not explicitly set, text fields use gray italic text and color swatches use a paler border.

***Note:** These default values apply to both the MXML editor and the CSS editor. However, in the MXML editor only some (not all) fields in the Flex Properties view are styles. Fields that are not styles will only display values that are explicitly set.*

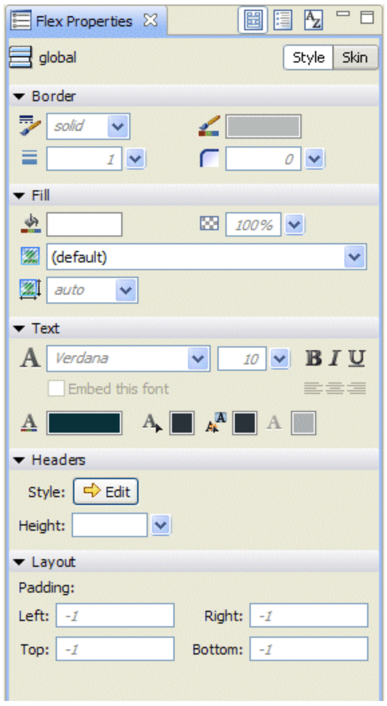
For more information, see “Using Styles and Themes” on page 470 in the *Adobe Flex 3 Developer Guide*.

Additional options in the CSS editor

Many of the features in the CSS Flex Properties view are expanded versions of similar features in the MXML editor. For example, you can also make changes to text, fill, and layout styles from this view.

Additional options in the CSS editor include an expanded fonts list, an Embed This Font option, and color swatches for choosing rollover, selected, and disabled text. The fonts list includes all web fonts, installed OS fonts, TTF files, and any additional embedded fonts in the current CSS file.

For more information, see the *Adobe Flex 3 Developer Guide*.



Importing Skin Artwork

You use the Import Skin Artwork wizard to import both vector graphics artwork and bitmap artwork from the CS3 versions of Flash, 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 Importing Skins into Flex Builder and Importing Flash CS3 Assets into Flex.*

1 Select File > Import > Skin Artwork.

In the plugin version, select File > Import > 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 CS3 for Flash Player 9
  - Vector graphic files created in Adobe illustrator CS3 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 will contain the style rules. The default name is based on the name of the artwork folder or Flash file being imported.
- Click the Delete All Existing Rules In File checkbox 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 unchecked 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 Flex 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

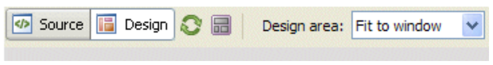
  For custom components, the item will be checked if the component has been defined somewhere within the project you are importing to.
- If necessary choose a style and skin part for the pop-up menus in each column.
- Click Finish.

  A CSS file is created and displayed in the Source view. The CSS file will be 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.

Refreshing 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. This can happen, for example, if you modify a visual element in a dependent Flash component (SWC). Styles and skins may also not be rendered properly because Flex Builder needs to rebuild the file.

❖ Click the Refresh button in the editor toolbar.



Laying out your user interface

To lay out a Flex user interface, you insert and position components in a container that has absolute positioning, and then define layout constraints for the components so they adjust automatically when a user resizes the application window. In Flex Builder you can define layout constraints in the design area or add or modify values in the Flex Properties view.

About constraint-based layouts

Flex supports constraint-based layouts. There are several approaches to layout: absolute positioning, nested H/VBox layouts, simple constraints, and advanced row/column constraints. The two scenarios outlined in the bullet points are best handled either by nested H/VBoxes or by advanced constraints.

In some situations, you could use nested H/VBoxes or advanced constraints (advanced constraints can only be edited in Source mode):

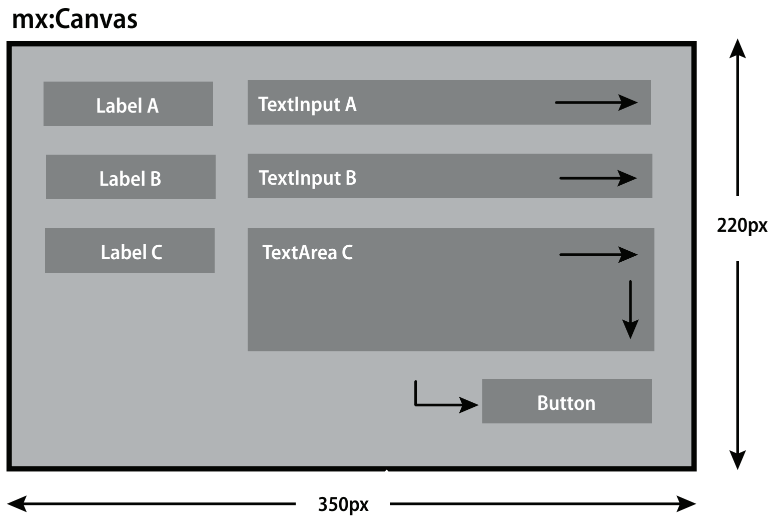
- When controls might be dynamically sized to fit their content—as in the case of localized strings, for example—and the controls need to push each other out of the way so they don't overlap.
- When you want multiple columns that are the same size or that need to remain a specific percentage width.

To create a constraint-based layout, you must use a container that has absolute positioning. The `layout="absolute"` property overrides the container's layout rule and lets you drag and position components anywhere in the container. This property can be used only with the Application, Canvas, and Panel containers. For Canvas containers, `layout="absolute"` is the default; for Application and Panel containers, you must set this property explicitly.

Absolute positioning gives you the ability to set layout constraints. For example, if you want a TextInput control to stretch when you make the application window wider, you can anchor the control to the left and right edges of the container so the width of the TextInput control is set by the width of the container.

When you create an MXML application file in Flex Builder, a `layout="absolute"` property is automatically included in the `<mx:Application>` tag.

In the following example, all the controls are absolutely positioned in a container either by dragging them or by setting the *x* and *y* coordinates in the Flex Properties view:

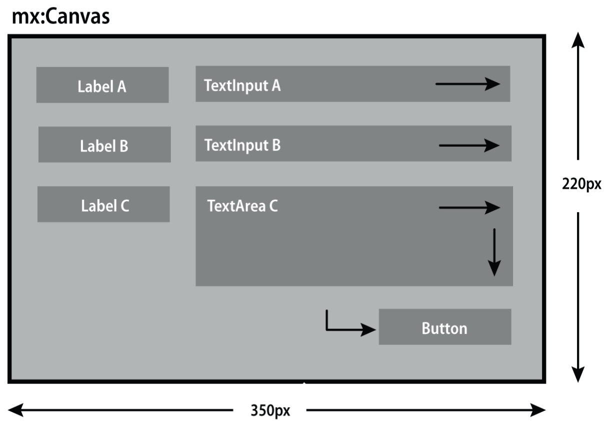


Also, a number of layout constraints are applied to the controls to ensure that the layout adjusts correctly when the user resizes the application:

- Label A, Label B, and Label C are anchored to the left and upper edges so the labels remain in place as the user resizes the layout.
- TextInput A and TextInput B are anchored to the left and right edges so the controls stretch or compress horizontally as the user resizes the layout.

- TextArea C is anchored to the left and right edges and to the upper and lower edges so that the control stretches or compresses horizontally and vertically as the user resizes the layout.
- The Button control is anchored to the right and lower edges so that the control maintains its position relative to the lower-right corner of the container as the user resizes the layout.

The following image shows how the constraints make the controls behave when the user resizes the layout:



The TextInput A and TextInput B controls stretch horizontally as the layout is enlarged. TextArea C control stretches horizontally and vertically. The Button control moves down and to the right.

For more information, see "Using Layout Containers" on page 373 in the *Adobe Flex 3 Developer Guide*.

Row and column constraints

You may also define a grid of horizontal and vertical constraint regions, ConstraintColumn regions and ConstraintRow regions. Components can be constrained to the edges or centers of these regions similarly to being constrained to the edges or centers of their parent containers. Constraint columns are laid out in the container from left to right, and constraint rows are laid out from top to bottom.

Constraint regions can be defined with fixed pixel dimensions (width or height) or as a percentage of the space in the parent container. The set of constraint columns and rows may have any combination of fixed or percentage dimensions.

Components within a parent container may be constrained to the container or to constraint regions or to any combination of container and region constraints.

For more information about row and column constraints, see the *Adobe Flex 3 Developer Guide*.

Inserting and positioning components in the layout

The first step in creating a constraint-based layout is to insert and position components in a container that has absolute positioning.

1 Ensure that the open MXML file includes a container that has absolute positioning.

2 In the MXML editor's Design mode, drag a component from the Components view into the container.

The component is inserted into your layout.

3 Position the component in the container either by moving it in the design area or by setting its `x` and `y` properties in the Flex Properties view (Window > Flex Properties).

4 Set layout constraints for the component.

Setting layout constraints for components

You use the Flex Properties view to set layout constraints relative to the edges of the parent container. Layout constraints relative to constraint columns and rows can only be set by editing the source code.

1 In the MXML editor's Design mode, select the component positioned in the container that has absolute positioning.

2 Use the Flex Properties view to specify constraints. You can expand the Layout category in this view. (You may need to scroll down to see the constraints tool.)

If you see a list of properties instead of a form, click the Standard View button in the toolbar.

3 Using the following table as a guide, select the constraint check boxes to achieve the effect you want when the user resizes the application:

Effect	**Constraints**
Maintain the component's position and size	None
Move the component horizontally	Left or Right
Move the component vertically	Top or Bottom
Move the component both horizontally and vertically	Left + Top or Right + Bottom
Resize the component horizontally	Left + Right
Resize the component vertically	Top + Bottom
Resize the component both horizontally and vertically	Left + Right and Top + Bottom
Center the component horizontally	Horizontal center
Center the component vertically	Vertical center
Center the component both horizontally and vertically	Vertical center + Horizontal center

4 Specify the distance of the constraints from the edges of the container.

For example, you can set the component to maintain its position 90 pixels from the left edge and 60 pixels from the right edge. If the user resizes the application, the component stretches or compresses to maintain these distances from the edges of the application window. Flex Builder expresses these constraints in the MXML code as follows, assuming you set a `y` property of 160:

```
<mx:TextInput y="160" left="90" right="60"/>
```

5 To set a component's column and row constraints with respect to a particular constraint region, prefix the pixel offset values with the ID of the appropriate ConstraintColumn or ConstraintRow.

Adding navigator containers

Navigator containers provide a way to organize your user interface into a group of related views. For example, you can use them to create a tabbed user interface. The containers provide built-in mechanisms for letting the user move through, or navigate, the views. For example, the TabNavigator container has tabs that let users select a different view.

Note: *You can also create multiview interfaces with Flex view states. For more information, see "Adding View States and Transitions" on page 93.*

You can use Flex Builder to insert navigator containers in your application, and then create the layout of each of the container's views. If you use a ViewStack container, you can use Flex Builder to add the means for the user to select a view.

Creating layouts in navigator containers

After you insert a navigator container in your application, you can use Flex Builder to create the layout of each of the container's views. The views are child containers of the navigator container.

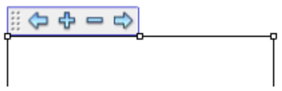
Only the ViewStack, TabNavigator, and Accordion navigator containers have child containers that you can lay out. The LinkBar, ButtonBar, and TabBar navigator containers don't have child containers. Instead, they let users control the active child container of a ViewStack container.

1 In the MXML editor's Design mode, drag a ViewStack, TabNavigator, or Accordion container from the Components view into the application.

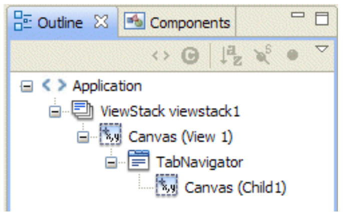
2 To add or remove a child container in the navigator container expand the top of the container.

3 For the ViewStack container, select a child container by clicking the Left or Right arrows that appear at the top of the container.

The arrows let you cycle through the child containers in the container.



You can also select views in the Outline view (Window > Outline).



4 For the TabNavigator and Accordion containers, click the child tab to select a child container.

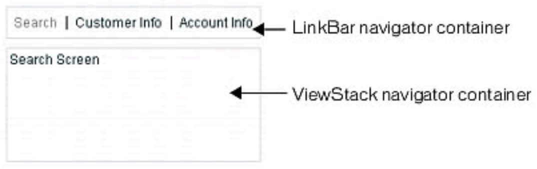
5 To set properties of the child container in the Properties view, click its background.

6 Create your layout by inserting controls or containers into the child container.

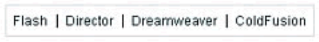
7 If you are working with a ViewStack container, add a mechanism to let users select the child containers.

Letting users select a view in a ViewStack container

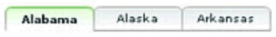
A ViewStack container consists of a collection of child containers stacked on top of each other with only one container visible, or active, at a time. The ViewStack container does not have a built-in mechanism for letting users select a child container. You must add a LinkBar, ButtonBar, or TabBar navigator container, or build the logic yourself in ActionScript. The following example shows an example of a ViewStack container with a LinkBar container:


Search | Customer Info | Account Info
LinkBar navigator container
Search Screen
ViewStack navigator container

You use Flex Builder to add a LinkBar, ButtonBar, or TabBar to a ViewStack container so that users can select the child containers in the ViewStack container. A LinkBar defines a horizontal row of Link controls that designate a series of link destinations, as the following example shows:



A ButtonBar defines a horizontal row of buttons. A TabBar defines a horizontal row of tabs, as the following example shows:



1 Ensure that you set the `label` property for each child in the ViewStack container.

The `label` property of the ViewStack children determines the text of the labels that appear on the TabBar or LinkBar. The following example shows children of a ViewStack container:

```
<mx:ViewStack id="myViewStack" borderStyle="solid" width="100%">
        <mx:Canvas id="search" label="Search">
            <mx:Label text="Enter search terms"/>
            ...
        </mx:Canvas>
        <mx:Canvas id="custInfo" label="Customer Info">
            <mx:Label text="Please enter your customer information"/>
            ...
        </mx:Canvas>
        <mx:Canvas id="accountInfo" label="Account Info">
            <mx:Label text="Please enter your account information"/>
            ...
        </mx:Canvas>
</mx:ViewStack>
```

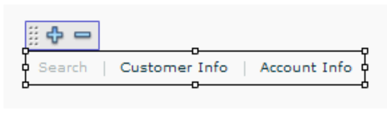
2 Drag a LinkBar, ButtonBar, or TabBar container from the Components view into your layout, above the ViewStack container.

The LinkBar, ButtonBar, and TabBar containers are listed in the Navigator category of the Components view.

3 Set the `dataProvider` property of the LinkBar, ButtonBar, or TabBar container to the ID of the target ViewStack container.

To set the `dataProvider` property, you can select the LinkBar, ButtonBar, or TabBar and set the property in the Flex Properties view. Alternatively, you can click the navigator's Plus (+) button and then select the ViewStack ID in the dialog box that appears.

Setting the `dataProvider` property to the ViewStack ID specifies the name of the ViewStack container associated with it. The text of the labels on the LinkBar, ButtonBar, or TabBar correspond to the values of the `label` property of each child of the ViewStack container.



Once the association is made between a ViewStack and a navigator, clicking the buttons or tabs of the navigator in Design mode will select the corresponding view in the ViewStack.

Adding data provider controls

You use Flex Builder to add data provider controls such as a ComboBox or a DataGrid to your application. For example, a ComboBox control might be populated with a list of countries from a database or an XML file.

Data provider controls take input from a data provider, which is a collection of objects similar to an array. After adding a data provider control, you must define a data provider for the control.

1 In the MXML editor's Design mode, drag a data provider control from the Components view into your user interface.

Data provider controls are listed in the Controls category of the Components view. They include the DataGrid, HorizontalList, List, TileList, Tree, and ComboBox controls. For more information, see "Using Data-Driven Controls" on page 295 in the *Adobe Flex 3 Developer Guide*.

2 Switch to Source mode and insert a `<mx:dataProvider>` child tag in the control's tag.

You can use Content Assist to quickly enter the tag. In the following example, you insert the `<mx:dataProvider>` tag in a Tree control tag:

```
<mx:Tree id="myTree">
    <mx:dataProvider>
    </mx:dataProvider>
</mx:Tree>
```

3 Specify the source of the data in the `<mx:dataProvider>` tag.

You can specify the data directly in the tag with an Array of Objects, as in the following example:

```
<mx:ComboBox>
    <mx:dataProvider>
        <mx:Array>
            <mx:Object label="Red" data="#FF0000"/>
            <mx:Object label="Green" data="#00FF00"/>
            <mx:Object label="Blue" data="#0000FF"/>
        </mx:Array>
    </mx:dataProvider>
</mx:ComboBox>
```

The `<mx:Object>` tag in the example defines a label property that contains the string to display in the ComboBox, and a `data` property that contains the value that you want to submit for processing.

You can define the data provider in an ActionScript variable, and then use the variable in the `<mx:dataProvider>` tag, as in the following example:

```
<mx:Script>
    <![CDATA[
        var COLOR_ARRAY:Array =
            [{label:"Red", data:"#FF0000"},
            {label:"Green", data:"#00FF00"},
            {label:"Blue", data:"#0000FF"}];
    ]]>
</mx:Script>

<mx:ComboBox>
    <mx:dataProvider>
        {COLOR_ARRAY}
    </mx:dataProvider>
</mx:ComboBox>
```

In the previous example, you could also write the `<mx:ComboBox>` tag as follows:

```
<mx:ComboBox dataProvider="{COLOR_ARRAY}">
</mx:ComboBox>
```

Adding charting components

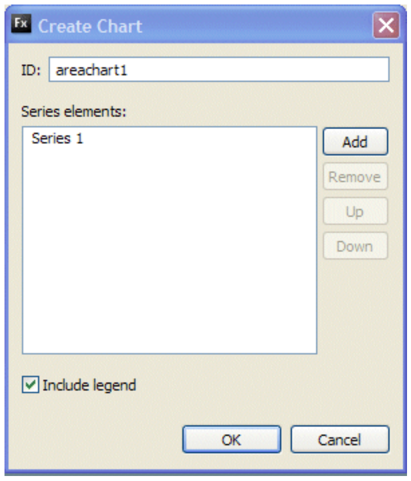
You can use Flex Builder to add charting components to display data in your user interface. The Flex charting components let you create some of the most common chart types, and also give you extensive control over the appearance of your charts. For an overview of the different charts available, see “Chart Types” on page 36 in *Adobe Flex 3 Data Visualization Developer Guide*.

The Flex charts are available in Adobe Flex Builder Professional. A trial version of the charts is included in the standard version of Flex Builder.

This section describes how to add charting components to your user interface. For information on defining chart data, formatting chart elements, and manipulating other aspects of charts, see “Introduction to Charts” on page 2 in *Adobe Flex 3 Data Visualization Developer Guide*.

Add a charting component

1 In the MXML editor’s Design mode, drag a charting component from the Components view into your user interface as shown below.


Create Chart

ID: areachart1

Series elements:

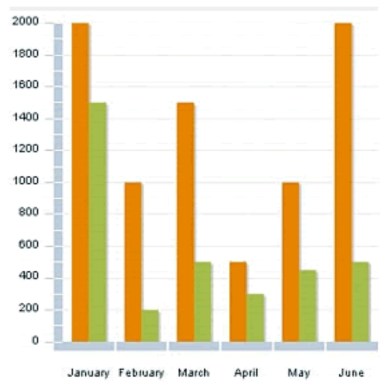
Series 1

[✓] Include legend

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.

Chapter 7: Adding View States and Transitions

You use Adobe® Flex® 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 kind of interactivity 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.

Topics

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 look and feel.

You create a view state by modifying the layout of an existing state. Modifications can include editing, moving, adding, or removing components. The modified layout is what the user sees when he or she switches state.

For a full conceptual overview of view states as well as examples, see "Using View States" on page 672 in the *Adobe Flex 3 Developer Guide*.

Generally, you do not add pages to a Flex application 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. For more information, see "Using Navigator Containers" on page 419 in the *Adobe Flex 3 Developer Guide*.

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. For more information on transitions, see “Using Transitions” on page 702 in the *Adobe Flex 3 Developer Guide*.

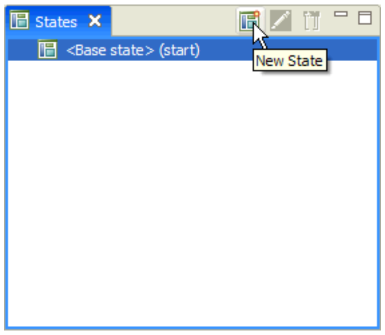
Creating a view state

You use Flex Builder to create a view state for an application or a component defined in an MXML file. First, you create a base state for the application or component, and then you create a second state based on the base state.

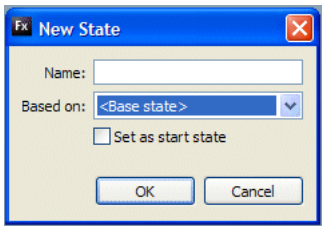
1 Using the layout tools in Flex Builder, design the layout of the base state of your application or component.

For more information, see “Building a Flex User Interface” on page 64.

2 In the States view (Window > States), click the New State button in the toolbar.



The New State dialog box appears. The newly created state is based on the selected state in the States view.



3 Enter a name for the new state, and click OK.

The name of the new state appears in the States view.

4 Use the layout tools in Flex 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.

5 Define an event handler that lets the user switch to the new state.

For more information, see “Switching states at run time” on page 95.

Creating a state based on an existing state

You use Flex Builder to create a state based on a state that is not the base state.

1 Ensure that the MXML file contains at least one state that is not the base state.

2 In the States view (Window > States), click the New State button.

3 Enter a name for the new state.

4 From the Based On pop-up menu, select the state on which to base the new state.

The pop-up menu lists the states defined in the current document.

5 Click OK.

6 Use the layout tools in Flex Builder to change the layout.

You can modify, move, add, or delete components. As you make changes to the state, they're recorded in the MXML code.

7 Define an event handler to switch to the new state.

Setting a non-base state as the starting state

By default, an application displays the base state when it starts. However, you can display a non-base state when the application starts.

1 With the MXML file opens in Design mode, double-click the view state that you want in the States view (Window > States).

2 In the Edit State Properties dialog box that appears, select the Set As Start State option and click OK.

Setting the initial state of a component

If your application has multiple states, you can set the initial state of a component independently from the rest of the application.

1 With the MXML file open in Design mode, select the component in your layout.

2 Select the initial state of the component in the State pop-up menu on the editor's toolbar.

Switching states at run time

When your Flex application is running, users need ways of switching from one view state to another. You can use Flex Builder to define event handlers for user controls so that users can switch states at run time.

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:

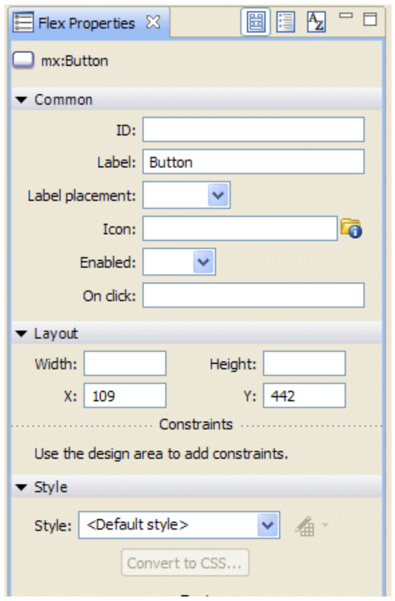
```
click="currentState='componentID.viewstatename'"
```

For more information, see “Applying view states” on page 679 in the *Adobe Flex 3 Developer Guide*.

1 Ensure that the initial state has a clickable control, such as a Button control.

In the MXML editor’s Design mode, select the control and enter the following value in the On Click field in the Flex Properties view:

currentState='*viewstatename*'



ID:
Label: Button
Label placement:
Icon:
Enabled:
On click:
Width:
Height:
X: 109
Y: 442
Use the design area to add constraints.
Style: <Default style>

2 If you want to switch to the base state, enter **currentState=''** (single-quote empty string) instead.

An empty string specifies the base state.

3 To test that the states switch correctly in the application when the button is clicked, click the Run button in the MXML editor toolbar.

You can define a transition so that the change between view states is smoother visually. For more information, see “Creating a transition” on page 97.

Modifying the appearance of existing states

After you create a state, you can use Flex Builder to modify its appearance at any time.

1 With the MXML file open in Flex Builder, select the view state that you want to modify from the States view (Window > States).

You can also select the view state from the State pop-up menu in the toolbar of the MXML editor in Design mode.

2 Use the layout tools in Flex Builder to make changes to the appearance of the state.

You can change properties, move components, add components, or delete components. As you work, the changes are recorded as part of the current state. The changes don't affect the states that the current state is based on, but they may affect states derived from the current state.

Deleting a view state

You use Flex Builder to delete a view state.

1 With the MXML file open in Design mode, select the view state that you want to delete from the States view (Window > States).

2 Click the Delete State button on the view's toolbar.

If you attempt to delete a state that other states are based on, a dialog box appears to warn you that the other states will be deleted too. You can cancel the operation or confirm that you want to delete the state.

Creating 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 you create at least one view state in addition to the base state.

2 In the MXML editor's Source mode, define a Transition object by writing a `<mx:transitions>` tag and then a `<mx:Transition>` child tag, as shown in the following example:

```
<mx:transitions>
    <mx:Transition id="myTransition">
    </mx:Transition>
</mx:transitions>
```

To define multiple transitions, insert additional `<mx:Transition>` child tags in the `<mx:transitions>` tag.

3 In the `<mx: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):

```
<mx:transitions>
    <mx:Transition id="myTransition" fromState="*" toState="checkout">
    </mx:Transition>
</mx: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">
    <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="{[myVBox1,myVBox2,myVBox3]}">
```

```
</mx:Parallel>
```

In this example, three VBox 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]}">
    <mx:Move duration="400"/>
    <mx:Resize duration="400"/>
</mx:Parallel>
```

For a list of possible effects, see “Available effects” on page 434 in the *Adobe Flex 3 Developer Guide*.

To set the properties of effects, see “Working with effects” on page 446 in the *Adobe Flex 3 Developer Guide*.

7 To test the transition, click the Run button in the MXML editor toolbar, then switch states after the application starts.

Chapter 8: Adding Interactivity with Behaviors

You use Adobe Flex Builder to create behaviors that add animation and motion to a component in response to user or programmatic action. For example, you can create a behavior for a TextInput component that causes it to bounce slightly when the user tabs to it, or you can create a behavior for a Label component that causes it to fade out when the user passes the mouse over it.

This topic describes how to visually create behaviors for components. For information on creating behaviors in MXML and ActionScript code, see “Using Behaviors” on page 427 in the *Adobe Flex 3 Developer Guide*.

Topics

About Flex behaviors

A behavior is a combination of a trigger paired with an effect. A trigger is an action, such as a mouse click on a component, a component getting focus, or a component becoming visible. An effect is a visible or audible change to a target component that occurs over a period of time, measured in milliseconds. Examples of effects are fading, resizing, or moving a component. You can define multiple effects for a single trigger.

By default, Flex components don’t play an effect when a trigger occurs. To configure a component to use an effect, you associate an effect with the trigger.

Triggers are not ActionScript events. For example, a Button control has both a mouseDownEffect trigger and a mouseDown event. The trigger initiates a Flex effect; the event calls an ActionScript function or object method.

For more information, see “Using Behaviors” on page 427 in the *Adobe Flex 3 Developer Guide*.

Creating a behavior for a component

You use Flex Builder to visually create a behavior for an MXML component.

***Note:** Though you cannot visually add an effect to an ActionScript component, you can use the editor’s Source mode to write the code that adds the effect. For more information, see “Applying behaviors in ActionScript” on page 436 in the Adobe Flex 3 Developer Guide.*

1 In the MXML editor’s Design mode, click on a component in the design area.

2 Define a trigger for the effect by selecting a trigger in the Effects category of the Flex Properties view.

The following naming conventions are used for triggers:

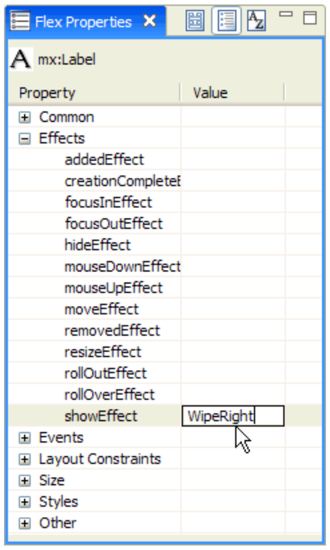
trigger`Effect`

In this convention, *`trigger`* is the trigger name. For example, to define a rollover trigger, select the `rollOverEffect` trigger.

For a list of triggers you can use, see “Available triggers” on page 432 in the *Adobe Flex 3 Developer Guide*.

3 Associate an effect with the trigger by entering the name of the effect in the field next to the trigger name.

For example, for a Label component you could enter WipeRight as the effect for the `showEffect` property. In a browser, the Label component's text becomes visible as if it were being wiped from left to right.



Property	Value
⊞ Common	
⊟ Effects	
addedEffect	
creationCompleteE	
focusInEffect	
focusOutEffect	
hideEffect	
mouseDownEffect	
mouseUpEffect	
moveEffect	
removedEffect	
resizeEffect	
rollOutEffect	
rollOverEffect	
showEffect	WipeRight
⊞ Events	
⊞ Layout Constraints	
⊞ Size	
⊞ Styles	
⊞ Other	

For a list of effects you can use, see "Available effects" on page 430 in the *Adobe Flex 3 Developer Guide*.

4 To customize the effect, you need to specify effect properties in the code.

For example, you can specify how long a wipe lasts by entering the value in milliseconds in the `duration` property of the component, as shown in the following example:

```
<mx:Button id="myButton" mouseDownEffect="WipeLeft" duration="2000"/>
```

For more information, see "Working with effects" on page 442 in the *Adobe Flex 3 Developer Guide*.

5 If you want the current component to trigger the effect, you are finished creating the behavior.

If you want another component to trigger the effect, you must write and insert an ActionScript function that triggers the effect from the other component. For more information, see "Applying behaviors in ActionScript" on page 436 in the *Adobe Flex 3 Developer Guide*.

Part 4: Programming Flex Applications

Topics

Chapter 9: Code Editing in Flex Builder

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

Topics

About code editing in Flex Builder

When designing and developing Flex and ActionScript applications, you work in the Flex Development perspective, which contains the Flex Builder editors and all the views that support code editing and design tasks. The configuration of the Flex Development perspective depends on which editor and mode you're working in. For example, when you create a Flex project, the MXML editor works in two modes (Source and Design) and each mode contains its own collection of supporting views. For an overview of the Flex Development perspective, see "The Flex Development perspective" on page 14.

MXML, ActionScript, and CSS editors

Flex Builder contains three editors for writing MXML, ActionScript, and CSS code.

The MXML editor lets you edit MXML files. The MXML editor contains two modes: Source and Design. In Source mode, you write MXML and embed ActionScript and CSS code contained within `<mx:Script>` and `<mx:Style>` tags. In Design mode, you lay out and design your Flex applications (see "Building a Flex User Interface" on page 64).

The ActionScript editor lets you edit AS files, which include main files for ActionScript projects, and class and interface files. For more information, see "About ActionScript projects" on page 44.

You use the CSS editor to write Cascading Style Sheets (CSS files). For more information, see "Applying styles and skins" on page 78 and "Using Styles and Themes" on page 470 in the *Adobe Flex 3 Developer Guide*.

MXML, ActionScript, and CSS content assistance

As you enter MXML, ActionScript, and CSS code, hints are displayed to help you complete your code. This feature is called *Content Assist*. Flex Builder also assists you in quickly developing your code by including MXML tag completion, automatic import management, integration with *Adobe Flex Language Reference*, and the capability of choosing different colors and fonts to display your code in the workspace. For more information, see "About Content Assist" on page 104 and

Custom component and ActionScript class and interface code hints

In addition to the built-in support for the Flex framework, code hints are provided for all custom components and ActionScript classes and interfaces that are included in your project. For more information, see "Using Content Assist" on page 106.

Streamlined code navigation

To easily navigate the many files and lines of code in your projects, Flex Builder provides convenient shortcuts, such as folding and unfolding code blocks, opening the source of external code definitions, browsing and opening class types, and so on. The Outline view also provides you with a convenient way to inspect the structure of and navigate to lines of code in your documents. For more information, see "Navigating and organizing code" on page 107.

Find references and code refactoring

Flex Builder lets you find all references and declarations to identifiers in a given file, project, or workspace including references found in elements linked from SWC files and other entries on a library path (for example, classes, interfaces, functions, variables, and some metadata). You use refactoring to rename identifiers in your code while updating all references to them in your entire code base. For more information, see "Finding references and refactoring code" on page 113.

Automatic syntax error checking

Flex Builder compiles your projects incrementally, giving you feedback as you work with your documents. If you introduce syntax errors while writing MXML and ActionScript code, error indicators are displayed next to the line of code in the editor and an error message is displayed in the Problems view. For more information, see "Apply syntax coloring preferences" on page 118.

Syntax coloring

You specify the set of colors to be applied throughout your code in the MXML, ActionScript, and CSS editors on the Syntax Coloring Preferences page. Font style options can also be applied and previewed from the same page. For more information, see "Getting help while writing code" on page 107 and "Using Content Assist" on page 106

Editor and debugger integration

The MXML and ActionScript editors work with the Flex Debugging perspective to assist you in debugging your code. You set breakpoints in your code to suspend your application at troublesome or otherwise crucial lines of code. When you begin a debugging session, the Flex Debugging perspective is displayed when the first breakpoint is reached. You can then inspect the state of your application and isolate and resolve the problems in your code. For more information about debugging your code, see "Running and Debugging Applications" on page 136.

About Flex Builder content assistance

The Flex Builder editors provide various ways to help you simplify and streamline code development.

Content Assist

Available in the MXML, ActionScript, and CSS editors, Content Assist provides code hints to help you complete your code expressions. For more information, see "About Content Assist" on page 104.

Auto import management

As you enter code with Content Assist, the fully qualified type names are automatically imported into the document as needed. In an ActionScript document, the fully qualified type name is added to the beginning of the document with the other import statements. In an MXML document, the import statement is added to an existing script block if one exists; if not, Flex Builder creates a script block. In an ActionScript document, you can also optionally sort import statements. For more information, see "Organizing import statements" on page 111.

MXML tag completion

As you enter MXML code, many syntax elements are automatically added, including: closing tags, indentations, new lines, and CDATA tags within `<mx:Script>` tags and when you add event attributes.

Context-sensitive language reference Help

The *Adobe Flex Language Reference* is integrated into the editor and you can easily access it by selecting an ActionScript language element, an MXML component tag or attribute, and then pressing Shift+F2. For more information, see "Getting help while writing code" on page 107.

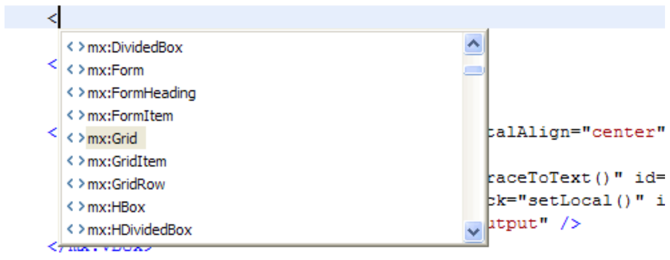
Formatting assistance

To streamline the work of coding your applications, the editors can help you reformat blocks of code and perform bulk edits. For more information, see "Formatting and editing code" on page 111.

About Content Assist

As you enter code into the Flex Builder editors, Content Assist prompts you with a list of options for completing your code expression (commonly referred to as *code hints*). For example, in an MXML document you are prompted with the list of tags that can be added at the current location.

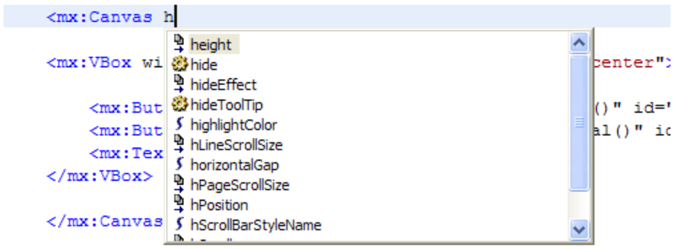
Code hints appear automatically as you enter your code. The following example shows the code hints that are displayed when you add a tag to a Canvas tag:



Only those tags that can be added to the Canvas tag are contained in the list of code hints. This is true of all uses of Content Assist: you only see relevant hints.

Code hints are categorized by type, showing you both visual and nonvisual MXML components, events, properties, and styles.

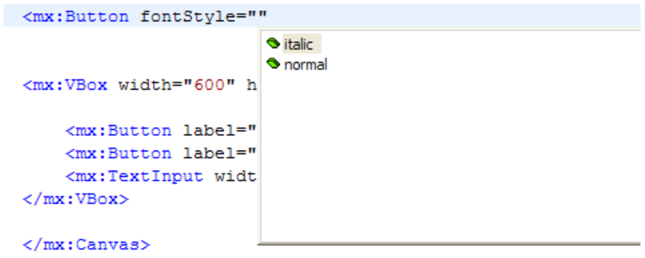
Code hints appear whenever the framework or language (MXML, ActionScript, and CSS) provides options for you to complete the current expression. For example, if you type within an MXML component, you are prompted with a list of all properties of that component. The following example shows code hints for properties of an MXML component:



```
<mx:Canvas h
    <mx:VBox wi                                      center">
        <mx:But                                      ()" id='
        <mx:But                                      al()" ic
        <mx:Tex
    </mx:VBox>
    </mx:Canvas
```

height
hide
hideEffect
hideToolTip
highlightColor
hLineScrollSize
horizontalGap
hPageScrollSize
hPosition
hScrollBarStyleName

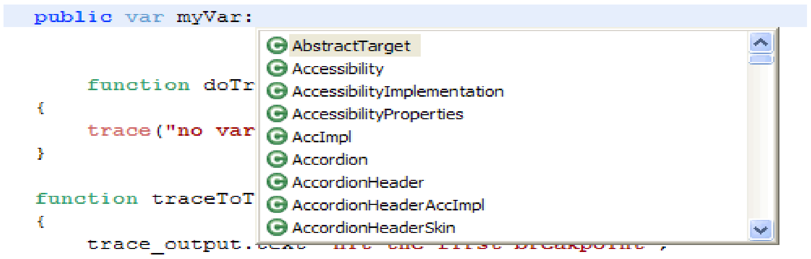
Selecting and entering properties displays possible property values (if predefined values exist). The following example shows code hints for property values:



```
<mx:Button fontStyle=""

<mx:VBox width="600" h
    <mx:Button label="
    <mx:Button label="
    <mx:TextInput widt
</mx:VBox>

</mx:Canvas>
```

italic
normal

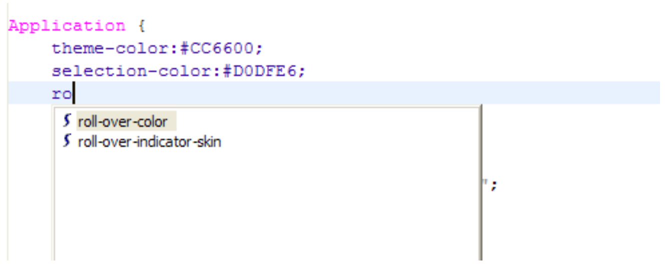
Code hints for ActionScript 3.0 are also supported. They are displayed in ActionScript documents, in `<mx:Script>` tags in MXML documents, and in event attributes. Content Assist provides hints for all ActionScript 3.0 language elements (interfaces, classes, variables, functions, return types, and so on), as the following example shows:



```
public var myVar:

    function doTr
{
    trace("no var
}
function traceToT
{
    trace_output.
```

AbstractTarget
Accessibility
AccessibilityImplementation
AccessibilityProperties
AccImpl
Accordion
AccordionHeader
AccordionHeaderAccImpl
AccordionHeaderSkin

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



```
Application {
    theme-color:#CC6600;
    selection-color:#D0DFE6;
    ro
```

In addition to the preceding examples, Content Assist provides hints for any custom MXML components or ActionScript classes that you create yourself and which are part of your project. For example, if you define a custom MXML component and add it to your project, code hints appear when you refer to the component in your MXML application.

Using Content Assist

You can use code hints to write MXML, ActionScript, and CSS more rapidly and efficiently. Code hints appear as you enter code into the editor.

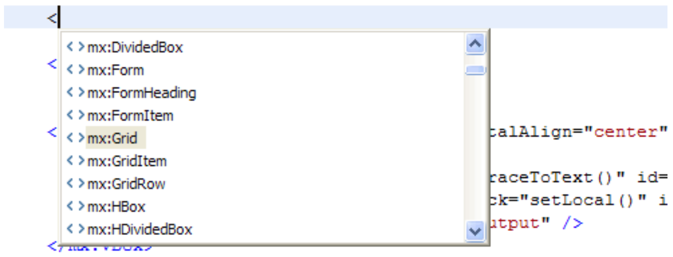
Display Content Assist and insert code hints:

1 Begin entering a line of code.

- In an MXML document, begin entering a tag:

  `<`

  Relevant code hints are displayed, as the following example shows:



- In an ActionScript document or a Script tag in an MXML document, enter a typical language construct:

  `public var myVar:`

- In a CSS document or a Style tag in an MXML document, enter a style name construct and press Control+Space to display a list of attributes that can be added.

  You can also display code hints while you enter a line of code by pressing Control+Space.

2 Navigate the list of code hints with the Up and Down Arrow keys.

3 Select a code hint, press Enter, and the code is added to the editor.

  As you continue to enter code, additional code hints are displayed.

Getting help while writing code

The *Adobe Flex Language Reference* is integrated into the MXML and ActionScript editors and you can quickly review the reference Help for an MXML tag or property, a class, or other Flex framework element.

You can also use Dynamic Help, which is docked next to the current perspective and displays reference and usage topics related to the currently selected MXML tag or ActionScript class.

Display language reference Help

1 In the MXML or ActionScript editor, select a Flex framework element (a word in your code) by highlighting or placing the mouse pointer in the word.

2 To open the Help topic directly in the Help viewer, press Shift+F2 or select Help > Find in Language Reference.

For more information about getting help while working in the Flex Builder workbench, see "Using the Flex Builder help system" on page 3.

Enable Dynamic Help

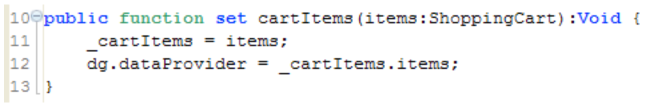
❖ Select Help > Dynamic Help.

Navigating and organizing code

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

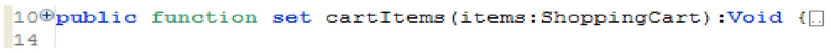
Setting, folding, and unfolding code blocks

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



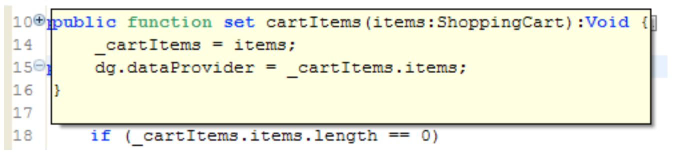
```
public function set cartItems(items:ShoppingCart):Void {
    _cartItems = items;
    dg.dataProvider = _cartItems.items;
}
```

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



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

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.



```
public function set cartItems(items:ShoppingCart):Void {
    _cartItems = items;
    dg.dataProvider = _cartItems.items;
}
    if (_cartItems.items.length == 0)
```

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

Using the Outline view to navigate and inspect code

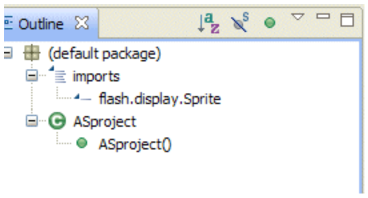
The Outline view is part of the Flex Development perspective (see "The Flex Development perspective" on page 14), and, therefore, is available when you edit code and design your application. You use the Outline view to more easily inspect and navigate the structure of your MXML, ActionScript, and CSS 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

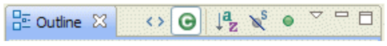
When you edit an ActionScript document (or ActionScript contained in an MXML document), the Outline view displays the structure of your code. This includes import statements, packages, classes, interfaces, variables not contained in functions, and functions. This view does not include metadata, comments, 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 was neither explicitly marked private nor public.

Outline view toolbar in Class mode

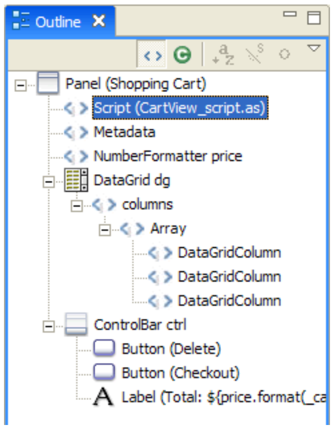
In Class mode, the Outline view toolbar contains the sort and filter commands, as the following example shows:



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.

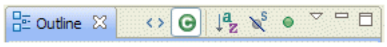
In MXML mode, each item in the Outline view represents an MXML tag and the following types of tags are displayed: components, controls, nonvisual tags such as `WebService` or `State`, component properties that are expressed as child tags (layout constraints, for example), and compiler tags such as `Model`,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

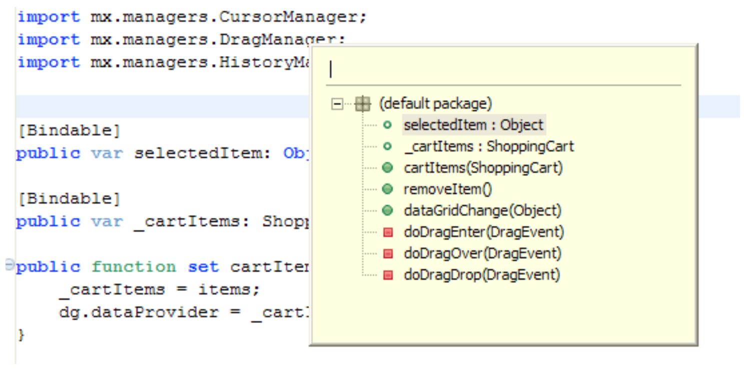
When the Outline view is in MXML mode, the toolbar contains additional commands that let you switch between the MXML and class views.



To switch between the two views, you use these toolbar commands. You can also switch the MXML editor modes (from Source to Design and vice versa) to achieve the same thing.

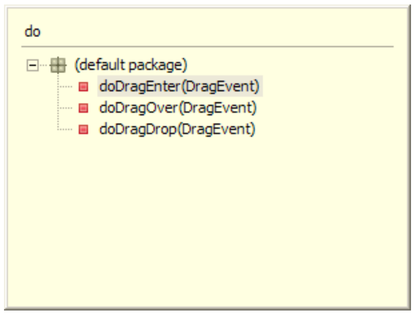
Using Quick Outline view in the editor

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.



```
import mx.managers.CursorManager;
import mx.managers.DragManager;
import mx.managers.HistoryMa

[Bindable]
public var selectedItem: Ob

[Bindable]
public var _cartItems: Shopp

public function set cartIte
    _cartItems = items;
    dg.dataProvider = _cart
}
```

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, press 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.

Opening code definitions

With applications of any complexity, your projects will contain many resources and many lines of code. To help simplify navigating and inspecting the various elements of your code, you can 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 Press F3.

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

Flex 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.

Browsing and opening class types

To quickly browse all the available types in your project (including the Flex framework), use the Open Type dialog box.

1 From anywhere in the workbench, press Control+Shift+T (Windows) or Command+Shift+T (Mac OS).

The Open Type dialog box appears.

2 To narrow the list of types, enter characters or the desired type name.

Only the types that contain the characters or name are displayed.

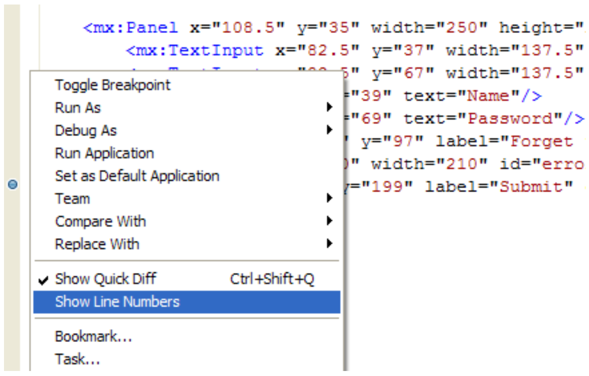
3 To open the class type source file in the editor, double-click the class type name.

Note: *You can browse, but not edit, the Flex framework classes.*

Showing line numbers

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

1 In the editor, right-click (Windows) or Control-click (Mac OS) in the editor margin, which is between the marker bar and the editor, to display the context menu.



```
    <mx:Panel x="108.5" y="35" width="250" height="
        <mx:TextInput x="82.5" y="37" width="137.5"
                               5" y="67" width="137.5"
                               ="39" text="Name"/>
                               ="69" text="Password"/>
                               ' y="97" label="Forget
                               )" width="210" id="erro
                               y="199" label="Submit"
```

2 Select the Show Line Numbers option.

Formatting and editing code

The Flex Builder editors provide shortcuts for writing and formatting your code. These include quickly adding comment blocks, indenting code blocks, finding, and replacing text.

Organizing 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 this, open the Preferences dialog, select Flex > Editors > ActionScript Code, and then select “Keep Imports Organized.”

Sort import statements

❖ With an ActionScript document that contains import statements open in the editor, press Control+Shift+O (Windows) or Command+Shift+O (Mac OS).

Adding comments and comment blocks

You can quickly add or remove comments using keyboard shortcuts. You can add comments (//) and comment blocks (/* */) to ActionScript code. You can add XML comments (<!-- -->) and CDATA blocks to MXML code. Comments in ActionScript code can be toggled on or off.

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.

Manually indenting 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. If, however, 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 Flex > Editors.

2 Select the indent type (Tabs or Spaces) and specify the IndentSize and Tab Size. (For more information, see "Setting editor preferences" on page 222.)

Finding and replacing text in the editor

To find and optionally replace text strings in your code, there are two options. You can search the document that is currently open in the editor, or you can search all resources in the projects in the workspace. For more information about searching the entire workspace, see "Finding references and refactoring code" on page 113.

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).*

Finding references and refactoring code

Flex 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, Flex Builder lets you find and mark references or declarations to identifiers in ActionScript and MXML files, projects, or workspaces. You use refactor to make changes to your code by renaming the following identifiers and then updating 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 on 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 “About markers” on page 114.

Find all references or declarations

1 In Source mode, click on 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 on an identifier in the editor.

2 Select Source > Refactor > Rename from the main menu.

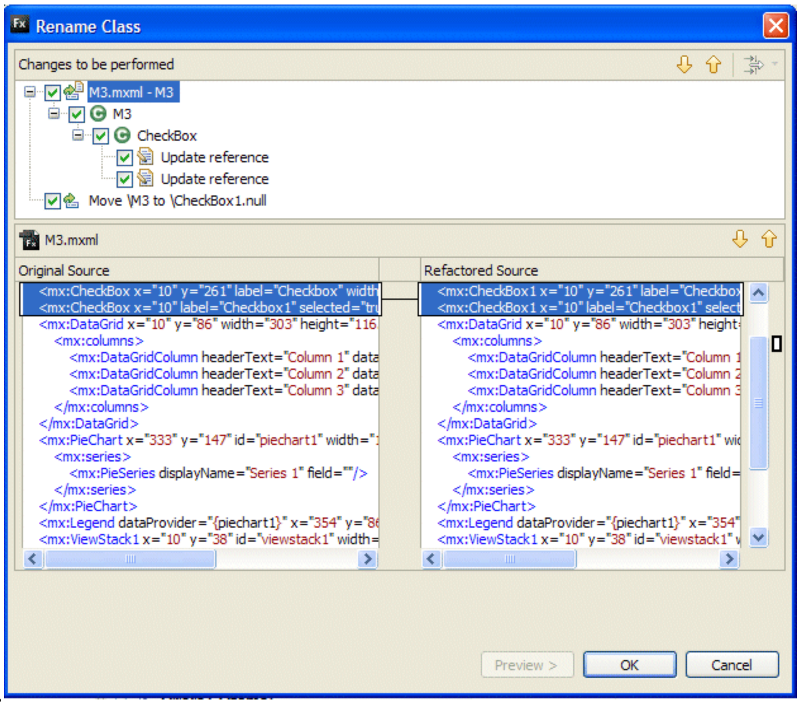
3 Enter a new name.

Flex 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.



Original Source:

```
<mx:CheckBox x="10" y="261" label="Checkbox" width
<mx:CheckBox x="10" label="Checkbox1" selected="tru
<mx:DataGrid x="10" y="86" width="303" height="116
  <mx:columns>
    <mx:DataGridColumn headerText="Column 1" data
    <mx:DataGridColumn headerText="Column 2" data
    <mx:DataGridColumn headerText="Column 3" data
  </mx:columns>
</mx:DataGrid>
<mx:PieChart x="333" y="147" id="piechart1" width="
  <mx:series>
    <mx:PieSeries displayName="Series 1" field=""/>
  </mx:series>
</mx:PieChart>
<mx:Legend dataProvider="{piechart1}" x="354" y="86
<mx:ViewStack1 x="10" y="38" id="viewstack1" width=
```

Refactored Source:

```
<mx:CheckBox1 x="10" y="261" label="Checkbox
<mx:CheckBox1 x="10" label="Checkbox1" select
<mx:DataGrid x="10" y="86" width="303" height
  <mx:columns>
    <mx:DataGridColumn headerText="Column 1
    <mx:DataGridColumn headerText="Column 2
    <mx:DataGridColumn headerText="Column 3
  </mx:columns>
</mx:DataGrid>
<mx:PieChart x="333" y="147" id="piechart1" wic
  <mx:series>
    <mx:PieSeries displayName="Series 1" field=
  </mx:series>
</mx:PieChart>
<mx:Legend dataProvider="{piechart1}" x="354"
<mx:ViewStack1 x="10" y="38" id="viewstack1" v
```

About 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 Flex 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 Flex 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 "Adding tasks" on page 115.

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 "Using the Problems view" on page 118.

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 "Adding and deleting bookmarks" on page 116.

***Note:** The Tasks and Bookmarks views are not displayed by default in the Flex Development perspective. For more information about adding these views, see "Opening views" on page 54.*

Navigating markers

Markers are descriptions of and links to items in project resources. Whether 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.

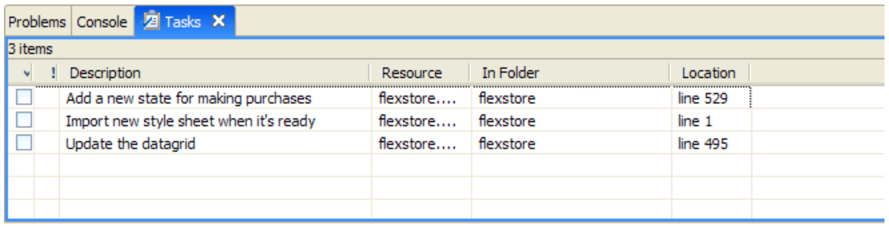
Go to a marker location

❖ 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.

Adding 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:



3 items

	!	Description	Resource	In Folder	Location
☐		Add a new state for making purchases	flexstore....	flexstore	line 529
☐		Import new style sheet when it's ready	flexstore....	flexstore	line 1
☐		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 Navigator view, 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.

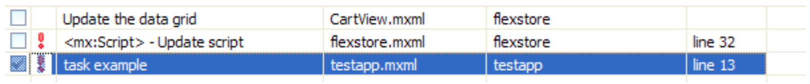
***Note:** The resource, as shown in the Flex Navigator view, does not indicate that it was marked. You can view and manage all task markers in the Task view.*

Completing and deleting 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:



☐		Update the data grid	CartView.mxml	flexstore	
☐	!	<mx:Script> - Update script	flexstore.mxml	flexstore	line 32
☑	↓	task example	testapp.mxml	testapp	line 13

Delete a task

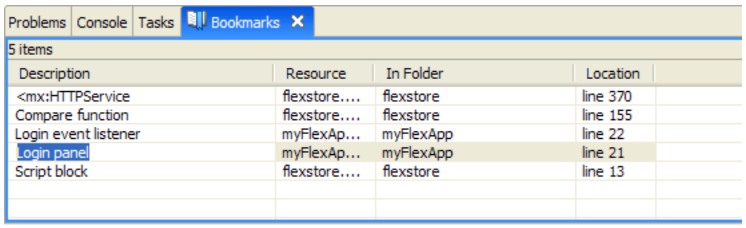
❖ In the Tasks view, right-click (Windows) or Control-click (Mac OS) the task, and select Delete.

Delete all completed tasks

❖ In the Tasks view, right-click (Windows) or Control-click (Mac OS) anywhere in the view to display the context menu, and select Delete Completed Tasks.

Adding and deleting bookmarks

You can use bookmarks to keep track of 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:



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 Navigator view, 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.

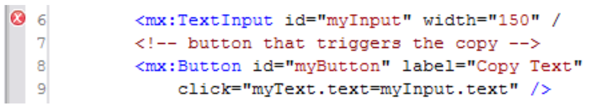
2 Right-click (Windows) or Control-click (Mac OS) the bookmark and select Delete.

About syntax error checking

The Flex Builder compiler identifies syntax errors and reports them to you so that you can correct them as you are working, before you attempt to run your application. You can easily adjust syntax coloring preferences.

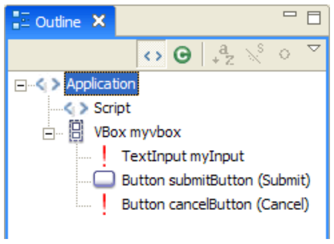
When code syntax errors are encountered, you are notified in the following ways:

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



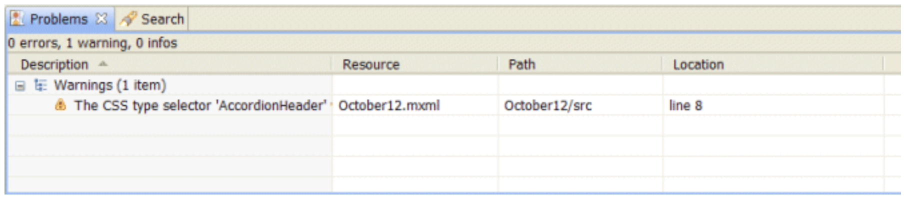
```
        <mx:TextInput id="myInput" width="150" /
        <!-- button that triggers the copy -->
        <mx:Button id="myButton" label="Copy Text"
            click="myText.text=myInput.text" />
```

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



```
Outline
Application
    Script
    VBox myvbox
        ! TextInput myInput
        Button submitButton (Submit)
        ! Button cancelButton (Cancel)
```

- 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:



Problems | Search

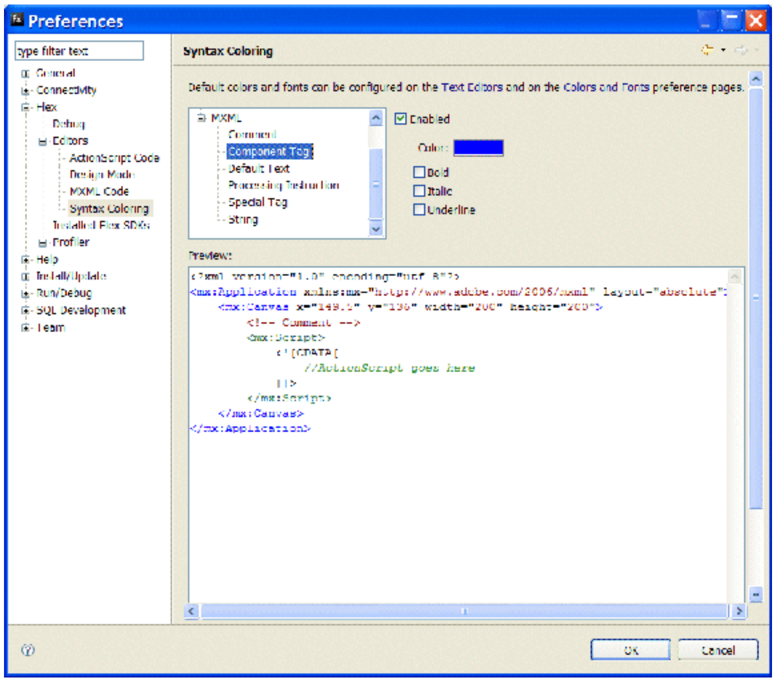
0 errors, 1 warning, 0 infos

Description	Resource	Path	Location
Warnings (1 item)			
The CSS type selector 'AccordionHeader'	October12.mxml	October12/src	line 8

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 Flex > Editors > Syntax Coloring.



Default colors and fonts can be configured on the Text Editors and on the Colors and Fonts preference pages.

☑ Enabled
Color: 🟦
☐ Bold
☐ Italic
☐ Underline

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Canvas x="149.5" y="136" width="200" height="200">
        <!-- Comment -->
        <mx:Script>
            <![CDATA[
                //ActionScript goes here
            ]]>
        </mx:Script>
    </mx:Canvas>
</mx:Application>
```

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

Using the Problems view

As you enter and save your code, Flex Builder compiles it in the background and displays syntax errors and warnings (problems) to you 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. When you double-click on a line of code, the file is opened in the editor and the line of code is highlighted.

Go to the line of code where an error or warning occurs

❖ Double-click a problem in the Problems view; or select a problem and then right-click (Windows) or Control-click (Mac OS) to display the context menu and select Go To.

Code editing keyboard shortcuts

The following table contains a list of keyboard shortcuts that are useful when editing code.

For a complete list of available keyboard shortcuts, see "Accessing keyboard shortcuts" on page 61. For information about editing existing or creating new keyboard shortcuts, see "Changing keyboard shortcuts" on page 58.

Name	Keyboard shortcut	Description
Switch between Source and Design mode	Control+`(Left Quote)	Switches between the MXML editor's Source and Design modes.
Go to Documentation (Flex Builder plug-in) Find in API Reference (Flex Builder stand-alone)	Shift+F2	When you edit MXML or ActionScript, selecting a language element and pressing Shift+F2 displays language reference Help for the selected element. For more information, see "Getting help while writing code" on page 107.
Context-sensitive Help	F1 (Windows) Command+Shift+/ (Mac OS)	Displays context-sensitive Help for the currently selected workbench element (editor, view, dialog box, and so on). For more information, see "Using the Flex Builder help system" on page 3.
Add Block Comment	Control+Shift+C (Windows) Command+Shift+C (Mac OS)	Adds block comment formatting to the currently selected lines of code or adds a comment at the insertion point. For more information, see "Adding comments and comment blocks" on page 112.
Add CDATA	Control+Shift+D (Windows) Command+Shift+D (Mac OS)	Adds a CDATA statement at the insertion point so that you can add ActionScript to an MXML document.
Find Matching Bracket	Control+Shift+P (Windows) Command+Shift+P (Mac OS)	Moves the cursor to the matching bracket of the selected code statement.
Content Assist	Control+Space (Windows) Command+Shift+Space (Mac OS)	Displays code hinting. For more information, see "Using Content Assist" on page 106.
Find All Declarations in Work-space	Control+G (Windows) Command+G (Mac OS)	Finds declarations in your code base. See "Finding refer-ences and refactoring code" on page 113.
Find All References in Workspace	Control+Shift+G (Windows) Command+Shift+G (Mac OS)	Finds references to identifiers in your code base. See "Finding references and refactoring code" on page 113
Go to Definition	F3	Open the source of an external code definition. For more information, see "Opening code definitions" on page 110.
Go to Line	Control+L (Windows) Command+L (Mac OS)	Displays the Go to Line dialog box where you can enter a line number and navigate to it in the editor.
Last Edit Location	Control+Q (Windows) Control+Q (Mac OS)	Highlights the last edited line of code.
Mark Occurrences		Marks every occurrence of the selected item in code.
Organize Imports	Control+Shift+O (Windows) Command+Shift+O (Mac OS)	When editing ActionScript, using this keyboard shortcut alphabetizes any import statements contained in the document. For more information, see "Organizing import statements" on page 111.
Open Type	Control+Shift+T (Windows) Command+Shift+T (Mac OS)	Quickly browse all class types. For more information, see "Browsing and opening class types" on page 111.

Name	Keyboard shortcut	Description
Open Resource	Control+Shift+R (Windows) Command+Shift+R (Mac OS)	Displays the Open Resource dialog box where you can quickly search for and open a resource in the editor.
Quick Outline	Control+O (Windows and Mac OS)	Displays the Outline view in Quick mode in the editor. For more information, see "Using Quick Outline view in the editor" on page 109.

Chapter 10: Building Projects

Adobe® Flex® 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 into applications. For example, you can set build preferences on individual projects or on all the projects in your workspace, 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.

Topics

Understanding 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, Flex 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 “Exporting projects” on page 38.

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 can be viewed by end users. 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 more information, see “Export Release Build” on page 126.

Adobe LiveCycle Data Services ES projects may instead be compiled on the server when accessed. For more information, see “Managing projects” on page 36.

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 “About library projects” on page 47.

Build basics

MXML and ActionScript 3.0 are *compiled* languages. Unlike *interpreted* languages such as JavaScript that can be immediately executed by their run-time environments, 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*.

Flex Builder automatically builds your projects whenever a file in your project is changed and saved. While you have the option of building your applications manually, this should not be necessary; however, understanding the build process and the output files that are generated will help you to diagnose and repair project configuration problems that may arise.

Flex **projects** Source files and embedded assets (such as images) are compiled into a single output format called SWF, which is a file that 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 (by default, this is named bin but you can name it anything you like).

LiveCycle Data Services ES projects When using LiveCycle Data Services ES you have the option of creating projects that are compiled on the server. When the MXML application file is first accessed (through a web browser), it is compiled into a SWF file.

***Note:** Even though you can configure LiveCycle Data Services ES projects to be compiled on the server, Flex Builder compiles these projects as you develop your applications so that the compiler can validate code syntax and display error messages. These projects have no output folder option and Flex Builder does not generate output files.*

ActionScript 3.0 projects Like Flex projects, ActionScript 3.0 projects compile source files and embedded assets into a SWF file.

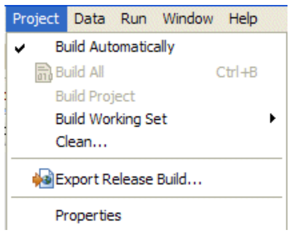
Flex **library projects** For library projects, source files are components and related resources. When library projects are built, a SWC file is generated into the output folder. A SWF file is archived into a SWC file containing components, resources, and a catalog.xml file that is the manifest of the elements contained within the SWF file.

Automatic builds

In the stand-alone configuration of Flex Builder, your applications are built automatically. In the plug-in configuration, you must select the Build Automatically option. While you have the option to build your applications manually, as mentioned above, this should not be necessary. Turning off automatic builds also prevents the compiler from identifying syntax errors and displaying warning and error messages as you enter code. In other words, you will not get any feedback in the Problems view until the project is compiled; therefore, it is best to set Flex 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. For more information, see "Advanced build options" on page 128.



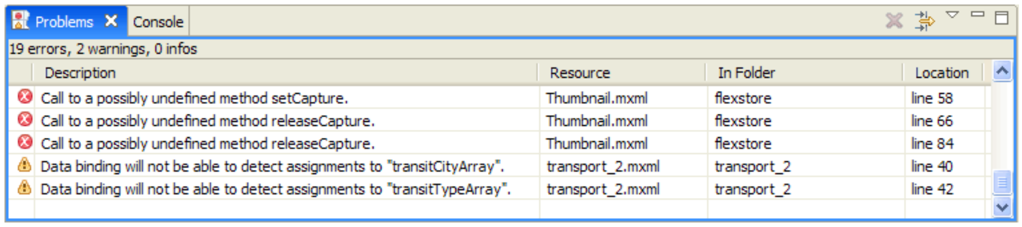
The Flex Builder compiler is incremental. It builds only those resources that have been added or affected by updates and ignores all others. This saves time and system resources. You have the option, however, to rebuild all the resources in the project. You do this by performing a *clean build*. You 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 128.

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. For more information, see "Building projects manually" on page 128.

You can also modify the build path, application list, and compiler settings for each project in the workspace. For more information, see "Building projects manually" on page 128, "Managing project application files" on page 40 and "Advanced build options" on page 128.

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:



19 errors, 2 warnings, 0 infos

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

For more information about working with the Problems view, see "Using the Problems view" on page 118.

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 run time. 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 "Customizing builds with Apache Ant" on page 131.

Command-line access to the Flex framework compilers

You have direct command-line access to use the Flex framework compilers (mxmlc and compc). When you install Flex Builder, the Flex framework prompt is available from the Windows start menu (Programs > Adobe). For more information, see "About the command-line compilers" on page 131 in *Building and Deploying Adobe Flex 3 Applications*.

Customizing project builds

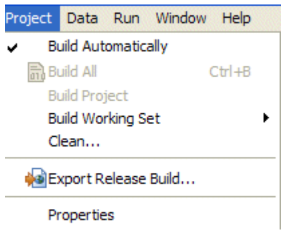
Flex Builder allows you to build your applications automatically using the default project settings. This is the recommended approach to building your applications. You can, however, customize project builds to suit your needs. For example, you may want to change the default output folder or modify the compiler options.

Enabling and disabling automatic builds

In the stand-alone configuration of Flex Builder, your projects are built automatically. In the plug-in configuration, you need to select this option yourself. Flex 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 "Building projects manually" on page 128.

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.

Setting up a project output folder

When you create a project in Flex Builder, by default, the build output is generated into the output folder. This does not apply to LiveCycle Data Services ES projects that use the server compile option because the application is compiled on the server when you run it.

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 Navigator view, 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 a LiveCycle Data Services ES application in this manner because its location is controlled by the Flex server and is accessible only through the project's Flex-config.xml file.*

5 Click OK.

The existing output folder is replaced by the new output folder.

Important: *When you change the name of the output folder, the original output folder and all of its contents will be deleted. You will need to rebuild the project to regenerate the application SWF and HTML wrapper files.*

Modifying 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 "About library projects" on page 47.) 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 Navigator view.

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 Action-Script 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 new path variable; for more information, see "Creating a path variable" on page 125.

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 "Using SWC files in your projects" on page 50).

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.

Creating 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 Navigator view.

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 Action-Script 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.

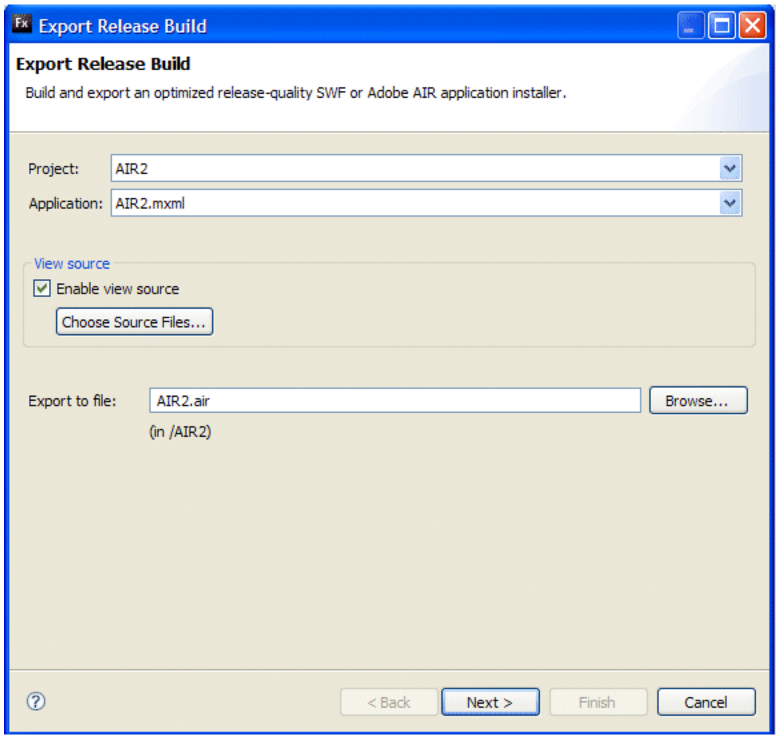
5 Enter a path variable using the following format: `${pathvariablename}`.

***Note:** If the variable does not already exist, the path entry will fail. 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.

Export Release Build

You can export an optimized release-quality version (non-debug SWF or AIR file) of your application using the Export Release Build Wizard. Required assets are copied to a folder separate from the debug version with or without source code ("View Source" in the application context menu). A .zip archive is also created automatically. You access the wizard from the Project menu or on the toolbar.


Export Release Build

Export Release Build

Build and export an optimized release-quality SWF or Adobe AIR application installer.

Project: AIR2

Application: AIR2.mxml

View source

☑ Enable view source

Export to file: AIR2.air

(in /AIR2)

Export Adobe AIR application installer

For AIR projects, a production build creates a digitally signed AIR file, which the user must install before running the application. This process is similar to creating an installer .exe for a typical native application. Optionally you can create an unsigned intermediate package which you can sign later before release. Before you begin the Export Release Build, you must decide how to digitally sign your AIR application:

- Sign the application using a VeriSign or Thawte digital certificate
- Create and use a self-signed digital certificate
- Choose to package the application and sign it later

Digital certificates provided by VeriSign and Thawte give users some assurance as to your identity as a publisher and verification that the installation file has not been altered since you signed it. Self-signed digital certificates serve the same purpose but they are not validated by a third-party.

You also have the option of packaging your AIR application without a digital signature by creating an intermediate AIR file (.airi). 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. This capability is provided because in some development environments signing is handled by a particular developer or team, which ensures an additional level of security.

1 Select Project > Export Release Build.
2 In the Export Release Build wizard, choose the export settings for project, application, and folder.
    - If your project does not have a server web root associated with it, all assets are copied to the *project_name* folder, which is the default location.
    - If your project has server web root associated with it (for example, PHP and J2EE), all assets are copied to the *web_root/project_name*-debug folder.
    - If you want users to view source code, select Enable View Source.
    - Click Choose Source Files to select files to you want to publish, then click OK.
    - Click Finish.

For more information about Adobe AIR files, see the *Adobe AIR Developer's Guide*.

Digitally signing your AIR applications

1 Select Project > Export Release Build.

    If you have multiple projects and applications open in Flex Builder, select the AIR project you want to package.

2 (Optional) Select Enable View Source if you want users to see the source code when they run the application.

    Click Choose Source Files to select individual files (all source files are selected by default).

3 On the Digital Signature page:

    Specify the digital certificate that represents the application publisher's identity. To generate a self-signed certificate, click Create to enter data in required fields.

    If you want to export a file that will be signed later, you can export an intermediate AIRI file.

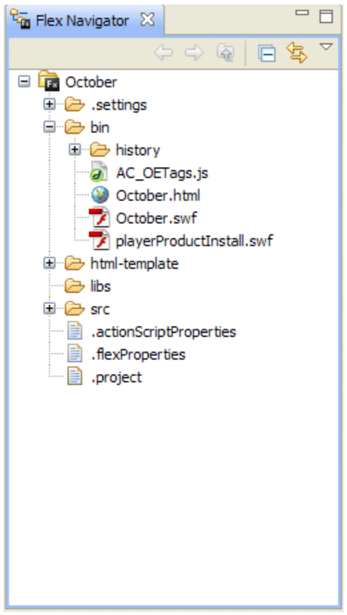
4 Click Finish.

Debug version

The debug version of your application contains debugging information and 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 will launch the development build in the bin-output folder (not the exported release build folder, bin-release.)*

In a standard Flex application, a typical output folder resembles the following example:



```
October
  .settings
  bin
    history
    AC_OETags.js
    October.html
    October.swf
    playerProductInstall.swf
  html-template
  libs
  src
  .actionScriptProperties
  .flexProperties
  .project
```

You can run or debug your Flex and ActionScript applications either in a web browser or in the stand-alone Flash Player. You control how your applications are run or debugged by modifying the project's launch configuration (see "Running your applications" on page 138). For more information about running and debugging your applications, see "Running and Debugging Applications" on page 136.

When you use LiveCycle Data Services ES you create Flex applications that leverage the Flex server technologies. When building LiveCycle Data Services ES applications, you have the option of compiling the output files locally using Flex Builder or on the server when the application is first accessed.

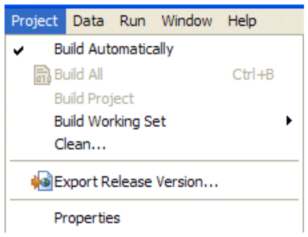
Advanced build options

Flex Builder has advanced options for customizing project builds. You can, for example, builds projects manually, change the default build order of projects in the workspace, and create custom builders using the Apache Ant utility.

Building 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 "Creating working sets" on page 56.

The build options are available in the Project menu, as shown in the following example:



Build a single project

1 In the Flex Navigator view, 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 "Creating working sets" on page 56.
- 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.

Saving 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) in the Workspace Save Interval text box.

Performing a clean build

After a project has been built, subsequent builds affect only the resources that have been added or modified. To force the Flex Builder compiler to rebuild all resources in a project, you can perform a clean build. You might 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.

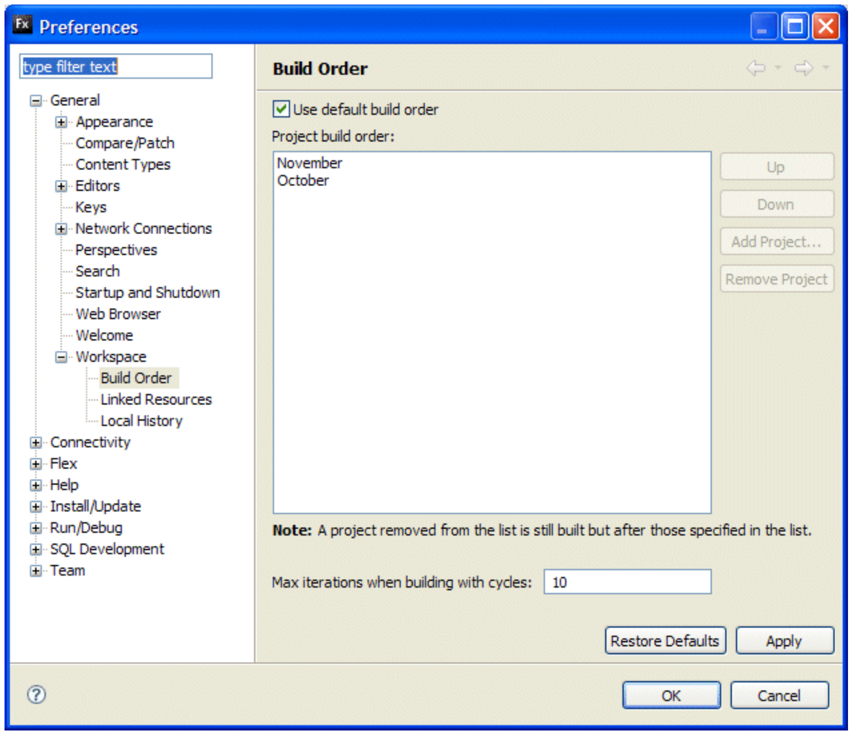
Changing the project build order

Flex 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 will be generated successfully.

You can, however, change the build order. For example, you might change the build order if you created a custom Ant builder and associated it with a project in your workspace, and you need to build that project before other projects are built. For more information about creating custom builders, see “Customizing builds with Apache Ant” on page 131.

1 Open the Preferences dialog and select General > Workspace > Build Order.



Note: A project removed from the list is still built but after those specified in the list.

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.

Customizing builds with Apache Ant

By creating a custom builder, you can modify and extend the standard build process. Flex 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.

While developing Ant build scripts is beyond the scope of this guide, this topic shows you how to create and apply a custom builder to your projects.

You can apply custom builders to all the Flex Builder project types.

Create a builder

1 In the Flex Navigator view, 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 may be more than one builder listed. Flex 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. Flex 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, which is available at http://help.eclipse.org/help31/index.jsp.

Using multiple SDKs in Flex Builder

Flex 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 2.0.1 SDK, then you are using the 2.0.1 version of the Flex framework SWC files, and the 2.0.1 version of the Flex compiler. You cannot use, for example, the Flex 3 compiler with the Flex 2.0.1 framework SWC files.

Using a different SDK can be useful if you are given a project that was developed using Flex Builder 2.0.1 (which uses the Flex 2.0.1 SDK), but you are running Flex Builder 3 (which uses the Flex 3 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 2.0.1 SDK, but want to use the Flex Builder 3 features such as code refactoring, you can upgrade your edition of Flex Builder, but then select the older SDK as the default SDK.

If you develop a project and then change the SDK, Flex Builder performs a full rebuild, not an incremental build. As a result, Flex Builder flags any differences that would throw compiler errors as if the project had been developed under the original SDK. For example, if you use an AdvancedDataGrid component in your project (which was first introduced in the Flex 3 SDK), but then change the project to use the 2.0.1 SDK, Flex Builder will notify you that the AdvancedDataGrid class is unknown.

Flex 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 2.0.1 SDK projects, Flex Builder creates the Flex 2.0.1. SDK–compatible history.swf, history.html, and history.js history management files in the html-templates directory. For Flex 3 SDK projects, Flex 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 Flex 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 2.0.1 SDK, Flex 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 3 SDK and the Flex 2.0.1 SDK, see “Backward compatibility” on page 286 in *Building and Deploying Adobe Flex 3 Applications*.

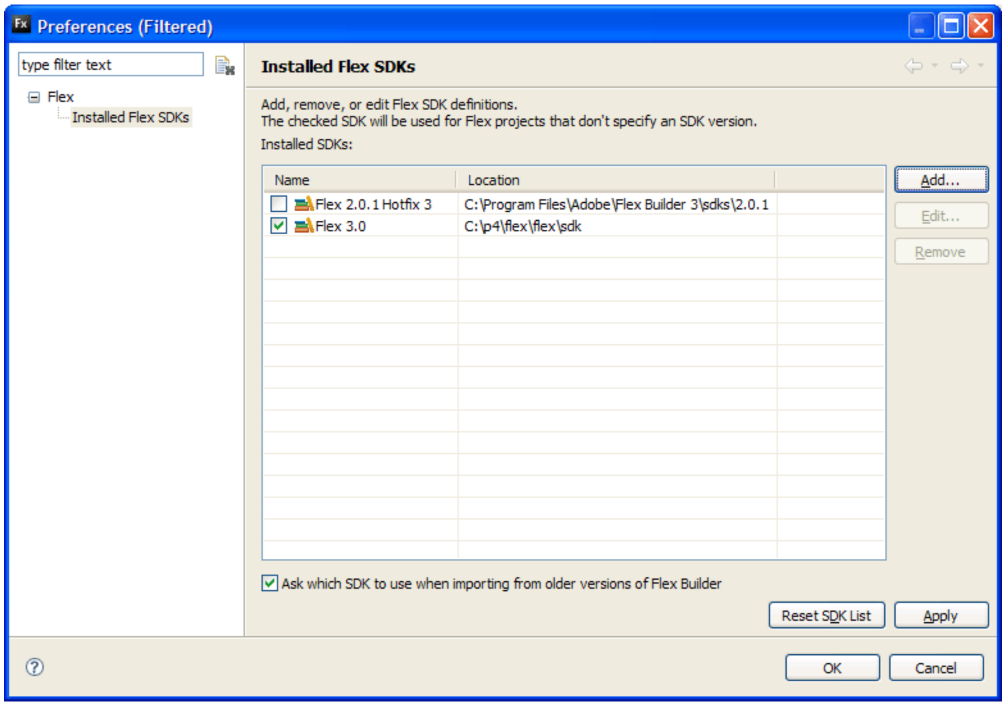
When you create a new Flex project, Flex Builder uses the default SDK. The default is the latest SDK that shipped with Flex Builder, but you can change it to any SDK that is visible in the list of available SDKs in Flex Builder.

When you create a new 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 Flex > Installed Flex SDKs.

The currently installed SDKs are listed. The default SDK has a check mark next to its name.



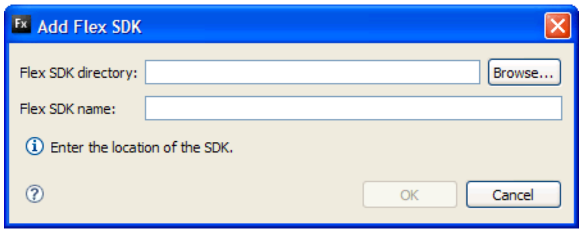
Installed Flex SDKs

Add, remove, or edit Flex SDK definitions.
The checked SDK will be used for Flex projects that don't specify an SDK version.
Installed SDKs:

	Name	Location
☐	Flex 2.0.1 Hotfix 3	C:\Program Files\Adobe\Flex Builder 3\sdks\2.0.1
✓	Flex 3.0	C:\p4\flex\flex\sdk

✓ Ask which SDK to use when importing from older versions of Flex Builder

2 Click Add.



Enter the location of the SDK.

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. You should 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 Flex Builder workspace, across Flex projects. The next time you create a new 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.

Flex Builder applies the new SDK to the current project. This can cause errors and warnings to 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 Flex > 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 will continue 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.

Using multiple SDKs in a server-based project

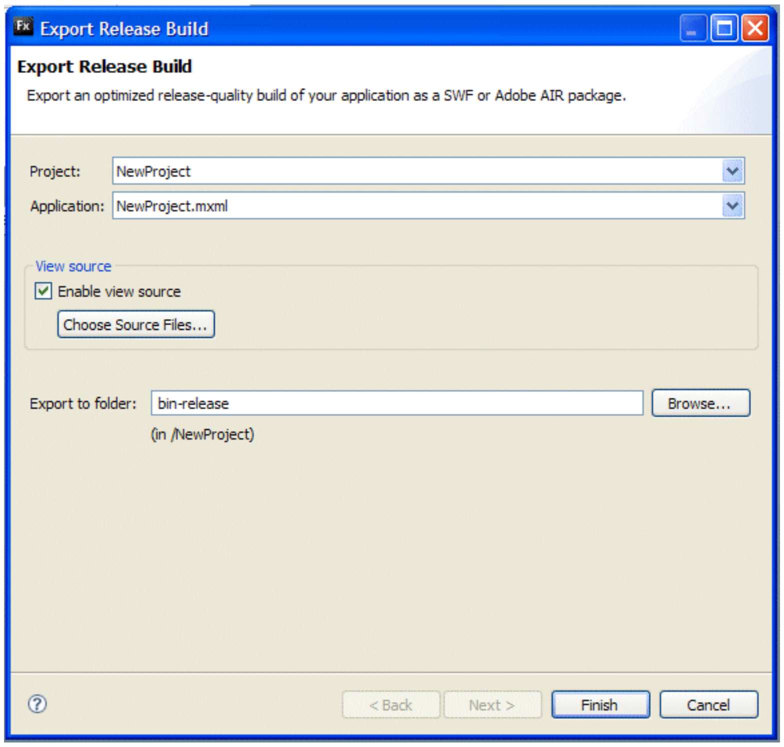
You can also change the SDK when compiling in a server-based environment. When you create a new server-based project, you are prompted either to compile the application locally or to compile the application on the server. If you choose the first option, to compile the application locally, Flex Builder uses the server's SDK. You can later change the SDK to a different one. If you choose the second option, to compile the application on the server, Flex Builder offloads the compilation to the server and uses the web-tier compiler to compile the project. In this case, you cannot change the SDK.

Publishing source code

When your applications are ready to be released, Flex 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.


Export Release Build

Export Release Build

Export an optimized release-quality build of your application as a SWF or Adobe AIR package.

Project: NewProject

Application: NewProject.mxml

View source

☑ Enable view source

Export to folder: bin-release

(in /NewProject)

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.

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 may not be able to view the source on your local development computer. If this is the case, you will need to deploy the application to a web server to view the source.*

Adding the view source menu to ActionScript projects

In Flex projects you add the View Source menu option to your application with the Export Release Build wizard. In ActionScript applications, you must 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.addMenuItem(obj:InteractiveObject, url:String,
hideBuiltins:Boolean = true)
```

You can use this 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.addMenuItem(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, your code should use the correct location.

Chapter 11: Running and Debugging Applications

When you test your applications in Adobe® Flex® Builder™, you run the application SWF files in a web browser or the stand-alone Flash Player. If you encounter errors in your applications, you use the debugging tools to set and manage breakpoints in your code; control application execution by suspending, resuming, and terminating the application; step into and over the code statements, select critical variables to watch, and evaluate watch expressions while the application runs.

Topics

About running and debugging applications

After your projects are built into applications (see “Building Projects” on page 121), you can run and debug them in Flex Builder.

Use debugger version of Flash Player

To use the Flex Builder debugging feature, you must run 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 is the version that is installed with Flex Builder, but it is also available as a download from the Adobe web site.

The installers for the debugger version of Flash Player are located in the *flex_builder_install*/Player directory.

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” on page 167 in *Building and Deploying Adobe Flex 3 Applications*.

Use launch configurations to run and debug applications

A launch configuration defines the project name, main application file, and the path to the run and debug versions of the application. Flex Builder contains a default Flex application launch configuration that is used to create launch configurations automatically for each of your projects. For more information, see “Managing launch configurations” on page 139.

***Note:** In the plug-in configuration of Flex Builder, a launch configuration is not automatically created for you. You need to create one the first time you run your application. See “Running your applications” on page 138.*

Customize launch configurations

You can customize the launch configurations that Flex Builder creates automatically for you. For example, you can switch the main application file or modify the path to point to the SWF file rather than the HTML wrapper file so that you can run and debug your applications directly in the stand-alone Flash Player instead of a web browser. You can also create separate launch configurations for each of the application files in your project. For more information, see "Creating or editing a launch configuration" on page 139.

Run and debug applications in a browser or the stand-alone Flash Player

By default, the launch configuration 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. Instead you can run and debug your applications in the stand-alone Flash Player (see "Running the application SWF file in the stand-alone Flash Player" on page 140). You can also override the system default web browser setting and run your applications in any browser you have installed (see "Changing the default web browser" on page 140).

Run debugging tools from the code editor

The Flex Debugging perspective provides all the debugging tools you expect from a robust, full-featured development tool. 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

For more information about the Flex Builder debugging tools, see "Debugging your applications" on page 141.

For more information about code editing, see "About code editing in Flex Builder" on page 102.

Activate the Debugging perspective at a breakpoint

You add breakpoints to executable lines of code in the code editor. When you begin a debugging session, the application runs until the first breakpoint is hit. The Flex Debugging perspective is then activated and you can inspect the state of and manage the application by using the debugging tools. For more information, see "Starting a debugging session" on page 141.

Compare debug and non-debug versions of your Flex application

By default, Flex Builder generates debug versions of your Flex 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 Flex application, you can do one of the following:

- Select Project > Export Release Build. This creates a non-debug SWF file or AIR file in the bin-release directory.
- Add `-debug=false` to the Additional Compiler Arguments field. This generates a non-debug SWF file no matter where you export it to.

For more information about Export Release Build, see "Export Release Build" on page 126.

Running your 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 "Managing launch configurations" on page 139.

Running your projects opens the main application SWF file in your default web browser or directly in the stand-alone Flash Player. For information about changing the default web browser or running and debugging with the stand-alone Flash Player, see "Changing the default web browser" on page 140 and "Running the application SWF file in the stand-alone Flash Player" on page 140.

You can run your projects in a number of ways in Flex Builder. For example, you can use the Run command, which is available from the workbench main menu and toolbar, from the Flex Navigator view, 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.*

Run project with default Flex application launch configuration

1. In the Flex Navigator view, select the project to run.
2. On the main workbench toolbar, click the Run button.

   If your project is not built yet, Flex Builder builds and runs it.

   Your application appears in your default web browser or the stand-alone Flash Player.

You can run and debug any project files that have been set as application files. For more information see "Managing project application files" on page 40.

Create custom launch configuration

1. In the Flex Navigator view, select the project you want to run and open the main application file in the editor.
2. From the main menu, select Run > Run > Other.
3. In the Create, Manage, and Run Configurations dialog box, select Flex Application.
4. Click the New button on the dialog box toolbar.
5. Enter the launch configuration name.
6. (Optional) Modify the configuration properties as needed.
7. (Optional) Click Run to run the application.

Run other application files in your project

1. In the Flex Navigator view, select the project to run.
2. From the main menu, select Run > Run > Other.
3. Select the application you want to run.

Run a custom launch configuration

❖ From the main menu, select Run > Run > Other.

In the Create, Manage, and Run Configurations dialog box, select an existing custom launch configuration or create a new one. See "Managing launch configurations" on page 139.

The Run command works a bit differently in the plug-in configuration of Flex Builder. Instead of running the currently selected project, it runs the most recently launched configuration. You can also select from a list of recently launched configurations.

Run the last launched configuration

❖ Click the Run button on the main toolbar.

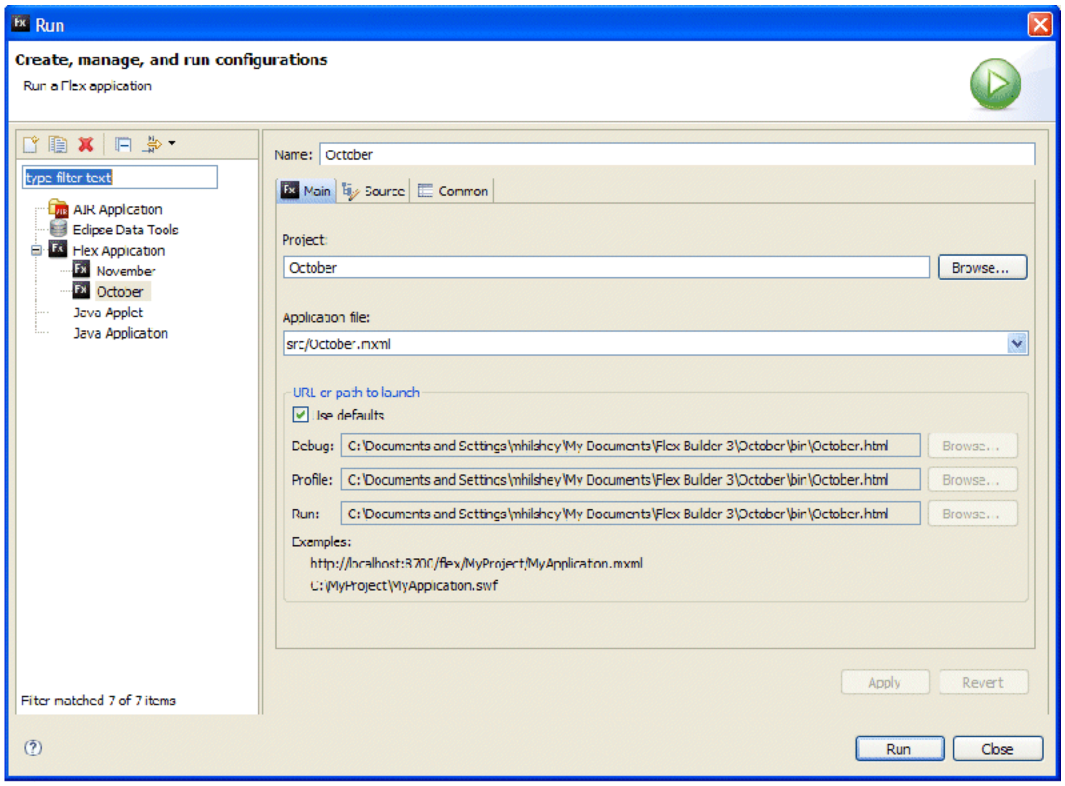
Managing launch configurations

Launch configurations are used both to run and to debug applications. Flex Builder provides a default launch configuration for Flex and ActionScript applications. When you first run or debug a project, a project-specific launch configuration is created. You 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.

Creating or editing a launch configuration

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, Flex 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 Flex application configuration, and you can edit them as necessary.

Launch configurations are managed in the Create, Manage, and Run Configurations dialog box.



Run

Create, manage, and run configurations

Run a Flex application

type filter text

AIR Application
Eclipse Data Tools
Flex Application
November
October
Java Applet
Java Application

Filter matched 7 of 7 items

Name: October

Main | Source | Common

Project: October

Application file: src/October.mxml

URL or path to launch

[✓] Use defaults

Debug: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3\October\bin\October.html

Profile: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3\October\bin\October.html

Run: C:\Documents and Settings\mhilshey\My Documents\Flex Builder 3\October\bin\October.html

Examples:
http://localhost:8700/flex/MyProject/MyApplication.mxml
C:\MyProject\MyApplication.swf

Access and edit a launch configuration

1 In the Flex Navigator view, select a project.

2 With a project file open in the code editor, open the Create, Manage, and Run Configurations dialog box. The way to do this depends on your Flex Builder installation.

- Stand-alone Flex Builder: Select the project and then select Run > Run > Other or Run > Debug > Other.
- Flex Builder plug-in with Eclipse 3.2: Right-click (Control-click on Macintosh) to display the context menu and select Run As > Run or Debug As > Debug.
- Flex Builder plug-in with Eclipse 3.3: Right-click (Control-click on Macintosh) to display the context menu and select Run As > Open Run Dialog or Debug As > Open Debug Dialog.

3 Select the launch configuration to edit.

A number of configurations may be listed if you have other projects in the workspace, if you have set other project files as application files, or if other Eclipse plug-ins are installed.

By default, the first time you run a project, Flex Builder creates a project-specific launch configuration, which is based on the default Flex application launch configuration.

You can edit application-specific configurations. You can also create new a launch configuration or base a new configuration on an existing configuration.

4 Modify the configuration preferences as needed, and click Run or Debug.

Running the application SWF file in the stand-alone Flash Player

Your applications are run or debugged in the default web browser. You can change the default web browser to use run and debug applications (see "Changing the default web browser" on page 140). You can also choose to run and debug your applications in the stand-alone Flash Player by making a simple change to the launch configuration.

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.

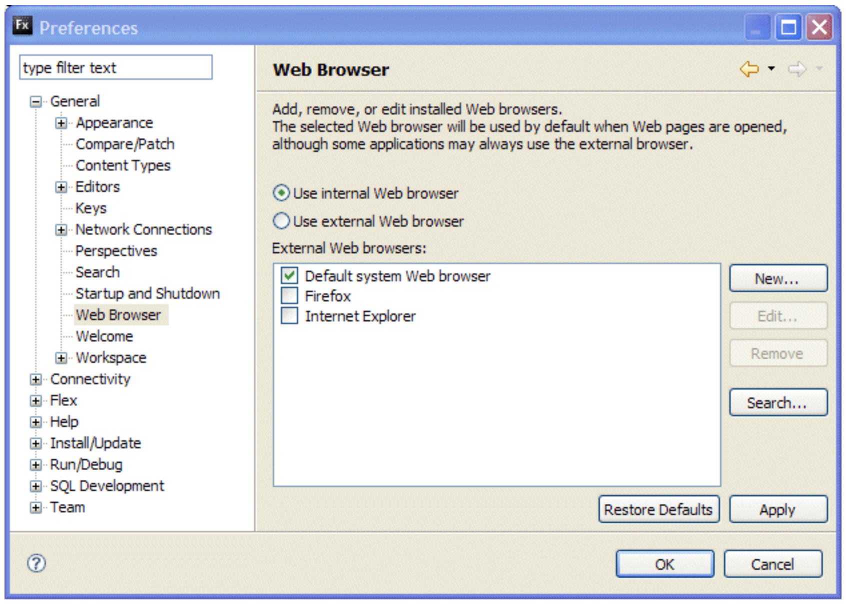
6 Apply your changes and use the modified configuration to run or debug the application.

Changing the default web browser

Flex 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.

Change the default web browser

1 Open the Preferences dialog and select General > Web Browser.



Web Browser

Add, remove, or edit installed Web browsers.
The selected Web browser will be used by default when Web pages are opened, although some applications may always use the external browser.

● Use internal Web browser
○ Use external Web browser

External Web browsers:

- [✓] Default system Web browser
- [ ] Firefox
- [ ] Internet Explorer

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.

Debugging your applications

Debugging is similar to running your applications. However, when you debug you control when the application stops at specific points in the code and 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.

For an overview of the debugging tools available in the Flex Debugging perspective, see "The Flex Debugging perspective" on page 19.

In some cases, you will be prompted to look at the Eclipse log file. For more information, see "Eclipse environment errors in the log file" on page 123.

Starting a debugging session

To begin a debugging session, you run the application launch configuration in the Flex Debugging perspective.

Debug an application

1 In the Flex Navigator view, select the project to debug.

2 Select the Debug button from the main workbench toolbar.

***Note:** The Debug button has two elements: the main action button and a drop-down list that 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.*

If your project has not been built yet, Flex 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 Flex Builder debugger to interact with it.

4 When a breakpoint is reached, the Flex Debugging 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 Flex 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.

Adding and removing breakpoints

You use breakpoints to suspend the execution of your application so you can inspect your code and use the Flex Builder debugging tools to explore options to fix 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 Flex Debugging perspective.

When the debugger encounters a breakpoint, the application is suspended, the Flex Debugging perspective is displayed, and the line of code is marked with a breakpoint that is highlighted in the code editor. You then use the debugging commands to interact with the code. (See "Managing the debugging session in the Debug view" on page 143).

Remove a breakpoint in the code editor

❖ In the marker bar, double-click an existing breakpoint.

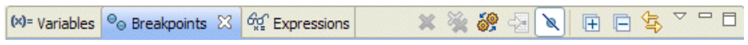
The breakpoint is removed from the marker bar and the Breakpoints view of the Flex Debugging perspective.

You manage breakpoints in the Breakpoints view. You can remove one or all breakpoints in the list or disable them and re-enable them at a later time (see "Managing breakpoints in the Breakpoints view" on page 143).

Managing breakpoints in the Breakpoints view

In the Breakpoints view you manage breakpoints during a debugging session. You can remove, disable and enable, or skip them.

The following commands are available from the Breakpoints view toolbar (as shown left to right):



Button/Command	Description
Remove Selected Breakpoints	Removes the selected breakpoints.
Remove All Breakpoints	Removes all breakpoints.
Show Breakpoints Supported by Selected Target	Displays breakpoints that are applicable to the select debug target.
Go to File for Breakpoint	Opens the file (if it is not already open) that contains the breakpoint in the code editor and highlights the line of code on which the breakpoint was set. You can also simply double-click the breakpoint to display it in the code editor.
Skip All Breakpoints	Skips all breakpoints.
Expand All	Expands all breakpoints.
Collapse All	Collapses all breakpoints.
Link With Debug View	Links to Debug view.

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.

You can also remove all the breakpoints in the Breakpoints view in a single action.

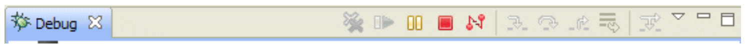
Remove all breakpoints from the Breakpoints view

❖ In the Breakpoints view, click Remove All Breakpoints.

Managing the debugging session in the Debug view

The Debug view is the control center of the Flex Debugging 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 provides the following debugging commands, which are available from the Debug view toolbar (as shown left to right):



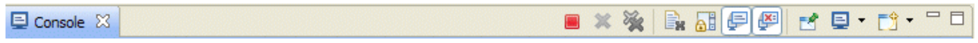
Button/Command	Description
Remove All Terminated Launches	Clears all terminated debugging sessions.
Resume	Resumes the suspended application.
Suspend	Suspends the application so that you can inspect, step into the code, and so on.
Terminate	Terminates the debugging session.
Disconnect	Disconnects the debugger when debugging remotely.
Step Into	Steps into the called function and stops at the first line of the function.

Button/Command	Description
Step Over	Executes the current line of the function and then stops at the next line of the function.
Step Return	Continues execution until the current function has returned to its caller.
Drop to Frame	This command is not supported in Flex Builder.
Use Step Filters	This command is not supported in Flex Builder.

Using the Console view

The Console view displays the output from trace statements placed in your ActionScript code and also feedback from the debugger itself (status, warnings, errors, and so on).

The Console view provides the following commands, which are available from the Console view toolbar (as shown left to right):



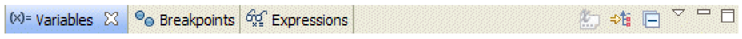
Button/Command	Description
Terminate	Terminates the debugging session.
Remove Launch	Clears all launched debugging sessions.
Remove All Terminated Launches	Clears all terminated debugging sessions.
Clear Console	Clears all content from the Console view.
Scroll Lock	Prevents the Console view from scrolling.
Show Console When Standard Out Changes	
Show Console When Standard Error Changes	
Pin Console	Prevents the console from refreshing its contents when another process is selected.
Display Selected Console	
Open Console	Opens new console and displays pop-up menu to select other console views.

Managing variables in the Variables view

The Variables view displays the variables that the currently selected stack frame defines (in the Debug view). Simple variables (name and value) are displayed on a single line. 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.

All superclass members are grouped in a separate tree node; by default you see only the members of the current class. This helps reduce excess numbers of variables that are visible at one time in Variables view.

The Variables view provides the following actions, which are available from the Variables view toolbar (as shown left to right):



Command	Description
Show Type Names	Displays the type names of variables.
Show Logical Structure	This command is not supported in Flex Builder.
Collapse All	Collapses 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

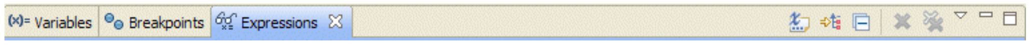
❖ To locate a variable or variable member in the Variables view, with the Variables view selected, begin entering the name of the variable you're looking for. You can also use the wildcard character (*) to search for words that occur anywhere within a variable name (for example, "*color").

Using the Expressions view

You use the Expressions view to watch variables you selected in the Variables view and to add and evaluate watch expressions while debugging your applications.

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.

The Expressions view provides the following commands, which are available from the Variables view toolbar (as shown left to right):



Command	Description
Show Type Names	Shows the object types for items in the Expressions view.
Show Logical Structure	This command is not supported in Flex Builder.
Collapse All	Collapses all expressions in view.
Remove Selected Expressions	Removes the selected variable or watch expression.
Remove All Expressions	Removes all variables and watch expressions from the Expressions view.

You can also hover the mouse 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.

Chapter 12: Creating Modules

You can create, add, optimize, and debug modules in Adobe® Flex® Builder™. For information on writing module code, see "Creating Modular Applications" on page 780 in the *Adobe Flex 3 Developer Guide*.

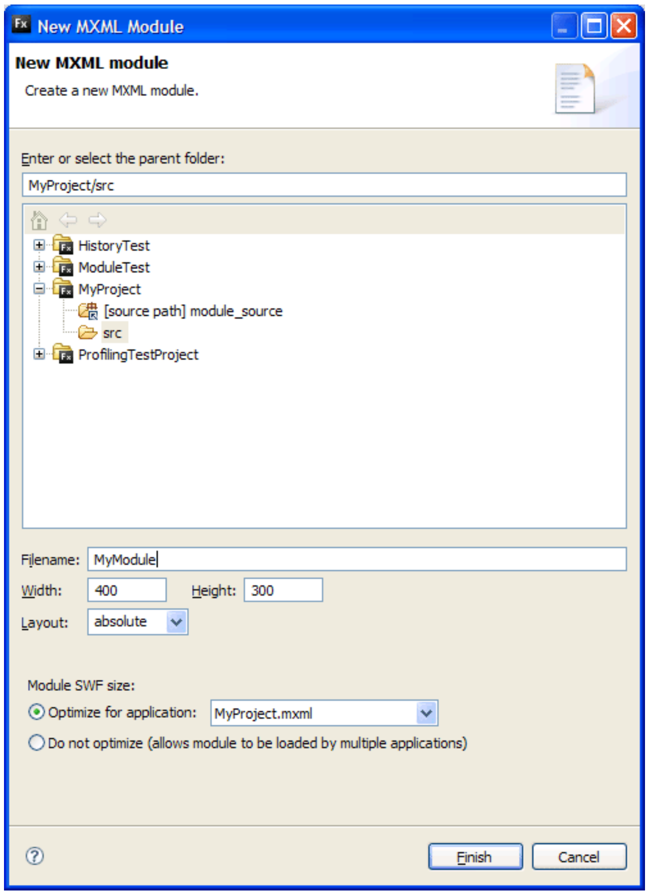
Topics

Creating modules in Flex Builder

The following steps describe how to create a new module in Flex Builder. After you create a new module, you can compile it.

Create modules in Flex Builder

1 In Flex Builder, select File > New > MXML Module. The New MXML Module dialog box appears.



Enter or select the parent folder: MyProject/src

Filename: MyModule

Width: 400 Height: 300

Layout: absolute

Module SWF size:

(●) Optimize for application: MyProject.mxml

( ) Do not optimize (allows module to be loaded by multiple applications)

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.

3 Enter a filename for the module; for example, MyModule.

4 Enter the Width, Height, and Layout properties for the module.

5 (Optional) Select the Optimize for Application radio button to excludes from the module classes that are used by the application. (This can result in smaller download sizes for your SWF files.) From the pop-up menu, select the application that will be used to optimize this module. For more information, see "Optimizing modules in Flex Builder" on page 153.

Select Do not Optimize to include all classes in the module, whether or not 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, not just the application that you select here, because it has all of its dependencies compiled into it.

6 Click Finish. Flex Builder adds a new MXML module file in your project.

The following example shows the default contents of this new application:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="400"
height="300">
</mx:Module>
```

Compiling modules in Flex Builder

In Flex 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 Flex application, then when you run your application, Flex Builder compiles the modules' SWF files for you. The SWF files are then loaded into the application at run time.

You cannot run the module-based SWF file as a stand-alone Flash application or load it into a browser window. It must be loaded by an application as a module. If you run the module in Flex Builder to compile it, you should close Adobe Flash Player or 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 Flex Builder compiles the modules at the same time as your application, regardless of their location (they 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, unless you compile all open projects in Flex Builder at one time.

If you compile modules separately from the main application, you must be sure to include or exclude debugging information, based on whether you want to debug your application and modules. For more information, see "Debugging modules in Flex Builder" on page 154.

The Flex 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.

Using 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.

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.

Using one project for each module has the following drawbacks:

- Having many projects uses more memory.
- Having 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, you must manually apply the `load-externs` and `link-report` compiler options.

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, you must manually apply the `load-externs` and `link-report` compiler options.

Creating projects for modules

When creating a separate project for modules, you change the module project's output folder to a directory that is used by the application. You also suppress the generation of wrapper files.

Create a separate project for modules in Flex Builder

1. Create a main project.
2. Create a new project for your module or modules.
3. Right click the module's project and 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.

Compiling projects for modules

Compiling multiple projects in Flex Builder is a common operation. First you choose the order in which you want the projects to be compiled and then you compile all projects at the same time.

Compile all projects at the same time in Flex 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, you use the Build Order dialog box. This 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 was just generated by the shell application. For more information on reducing module size, see “Optimizing modules in Flex Builder” on page 153.

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 will still be 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 run time. Modules are not meant to be run in the Player or in a browser directly.

Adding modules to your project

In some cases, you use modules that are not in your main application's project. You might have the module in a separate project so that you can use custom configuration options, or you might want to share the module across multiple applications. You must 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

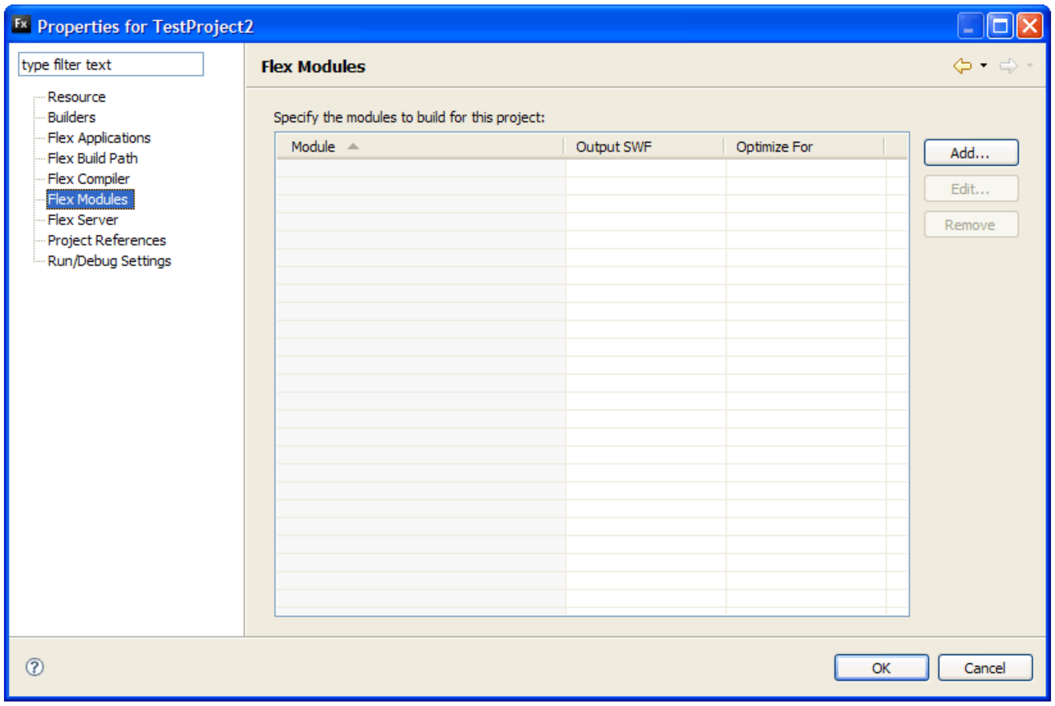
1 Select the main application in the navigator.

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. You must do this for each external module that you add to your application's project.

4 Click OK again to save your changes.

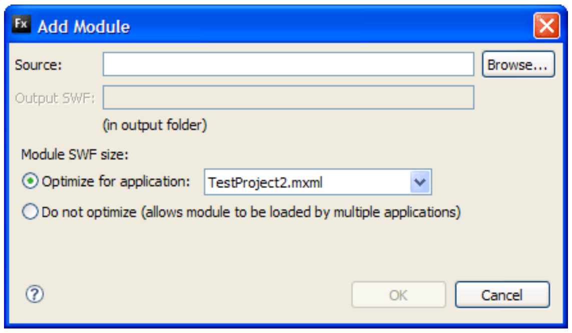
5 Click Project > Properties > Flex Modules to add the module to the application's module list. The Flex Modules dialog box lists all modules that have been added to the current project or that are in the current project. When you first create a project, this dialog box is empty.



Specify the modules to build for this project:

Module	Output SWF	Optimize For

6 Click the Add button. The Add Module dialog box appears.



Source:
Output SWF:
(in output folder)
Module SWF size:
[✓ Optimize for application:] TestProject2.mxml
Do not optimize (allows module to be loaded by multiple applications)

7 Use the Browse button or enter the location of the module's MXML file in the Source text box. All modules that are in the project's source path are available to add by using this dialog box.

8 Select one of the radio buttons under Module SWF Size to enable or disable module optimization. If you select Optimize for Application, Flex 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 "Optimizing modules in Flex Builder" on page 153.

9 Click OK to save your changes. Flex Builder adds the module to the list of available modules in your application's project.

Optimizing modules in Flex Builder

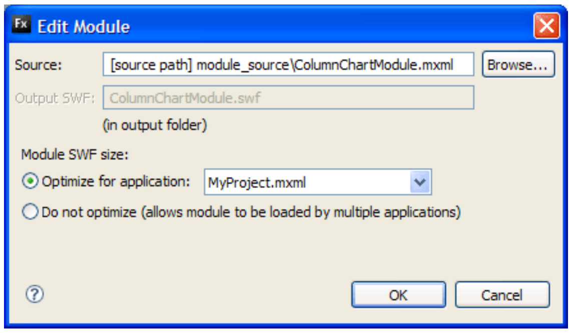
In Flex Builder, you typically select a single application to optimize the module against when you first create the module or add it to a project. If you later decide to change the application that you optimize the module against, or if do not want to optimize the module, you can edit the module's properties within the project. For more information, see "Reducing module size" on page 785 in the *Adobe Flex 3 Developer Guide*.

Optimize modules with Flex Builder

1 Right-click the application's project in the Navigator 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.



Source: [source path] module_source\ColumnChartModule.mxml
Output SWF: ColumnChartModule.swf
(in output folder)
Module SWF size:
(●) Optimize for application: MyProject.mxml
( ) Do not optimize (allows module to be loaded by multiple applications)

4 To remove optimization, select the Do Not Optimize radio button 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 Flex 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 "Debugging modules in Flex Builder" on page 154.

Debugging modules in Flex Builder

To debug modules and applications, you must include debug information in the SWF files when they are compiled. To do this in Flex Builder, you just 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, you must be sure to override it.

By default, Flex 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 Flex Builder. However, including debug symbols in a module's SWF file makes the SWF file larger. To exclude debug symbols prior to deployment, you must disable debugging for the application's modules. To do this, you export the release version of the modules by selecting Project > Export Release Build.

To exclude debugging information from SWF files in Flex 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 nondebug 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 will 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're 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.

Chapter 13: Profiling Flex applications

As you interact with your application, identify performance bottlenecks and memory leaks in your applications by using the Adobe Flex profiler in Adobe Flex Builder. The profiler is only available for Flex Builder Professional.

This topic contains the following sections:

About profiling

The Adobe Flex profiler helps you identify performance bottlenecks and memory leaks in your applications. You launch it from within Adobe Flex Builder, and 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.

Profiling an application can help you understand the following about your application:

- **Call frequency** In some cases, you might discover that 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 will 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. This might lead you to discover that methods are being called unnecessarily.
- **Number of instances (object allocation)** You might discover that the same object is being created many times, when only a specific number of instances are required. In these cases, you might consider implementing a Singleton pattern if you really require only one of those objects, or applying other techniques that reduce excessive object allocation. If the number of objects is large, but necessary, you might 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. This is especially helpful if you optimize objects that are created many times in the application.
- **Garbage collection** When comparing profiling snapshots, you might discover that some objects that are no longer required by the application 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.

You should not look at profiling as only a single, discrete step in the process of developing an application. Rather, profiling should be an integral part of each step of application development. If possible, you should profile an application early and often during application development so that you can quickly identify problem areas. Profiling is an iterative process, and you gain the most benefit by profiling as often as possible.

About types of profiling

Before you use the profiler, you should decide what kind of 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 hot spots 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. You should only do memory profiling 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 kind of 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, look at the following resources in the Flex documentation for help in improving your application:

- “Optimizing Flex Applications” on page 42 in *Building and Deploying Adobe Flex 3 Applications*
- “Improving Startup Performance” on page 67 in *Building and Deploying Adobe Flex 3 Applications*

How the Flex 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 very 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.

About the 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.

About internal player actions

Typically, the profiler records that Flash Player was executing methods of a particular class during the sampling snapshot, but sometimes it 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.
`[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).
`[sweep]`	Flash Player reclaims memory of unmarked objects.
`[verify]`	The JIT compiler performs ActionScript 3.0 bytecode verification.
`[`*`event_type`*`Event]`	Flash Player dispatches the specified event.

You can use this information to help you identify performance issues. For example, if you see a large number of entries for `[mark]` and `[sweep]`, you can assume that there are a large number of 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 have any effect.

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 “Using the Performance Profile view” on page 174 and “Using the Allocation Trace view” on page 171.

Using the profiler

The profiler requires Flash Player version 9.0.115 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 Adobe® AIR™ applications.

The profiler requires debugging information in the application that you are profiling. When you compile an application and launch the profiler, Flex 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.

Starting, stopping, and resuming the profiler

You can profile applications that you are currently developing in Flex Builder. Flex 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 Flex 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 "Profiling external applications" on page 164.

Start profiling a Flex application

1 Close all instances of your browser.

2 Open your Flex application in Flex Builder.

3 Click the Profile *application_name* button in the main toolbar. Flex Builder informs you that you should close all instances of your browsers if you have not already done so.

4 Click the OK button. Flex Builder compiles the application and launches it in a separate browser window. Flex 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, you must select the Enable Memory Profiling option or the Enable Performance Profiling option.

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.
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. You use this option to detect memory leaks or find excessive object creation.
	If you are doing performance profiling, you can deselect this option.
	The default value is that the option is selected.
Watch Live Memory Data	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.
	The default value is that the option is selected.

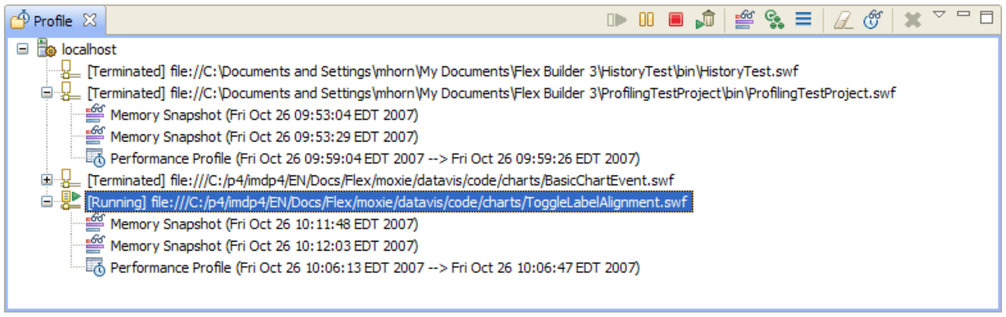
Setting	Description
Generate Object Allocation Stack Traces	Instructs the profiler to capture a stack trace each time a new object is created. Enabling this option can slow down the profiling experience, so you should only do it when absolutely necessary. You can select this option only if you selected Enable Memory Profiling. The default value is that the option is unselected. If you do not select this option, you cannot view allocation trace information on the Object References view or on the Allocation Trace view.
Enable Performance Profiling	Instructs the profiler to collect stack trace data at the sampling intervals. You use these samples to determine where the bulk of the execution time in your application is spent. If you are doing memory profiling, you can deselect this option. The default value is that the option is selected.

You can change the default values of these options by changing the profiling preferences. For more information, see "Setting profiler preferences" on page 163.

6 You can now start interacting with your application and examining the profiler data.

Pause and resume profiling a Flex 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, and all other applications have been terminated.



```
localhost
  [Terminated] file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\HistoryTest\bin\HistoryTest.swf
  [Terminated] file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\ProfilingTestProject\bin\ProfilingTestProject.swf
    Memory Snapshot (Fri Oct 26 09:53:04 EDT 2007)
    Memory Snapshot (Fri Oct 26 09:53:29 EDT 2007)
    Performance Profile (Fri Oct 26 09:59:04 EDT 2007 --> Fri Oct 26 09:59:26 EDT 2007)
  [Terminated] file:///C:/p4/imdp4/EN/Docs/Flex/moxie/datavis/code/charts/BasicChartEvent.swf
  [Running] file:///C:/p4/imdp4/EN/Docs/Flex/moxie/datavis/code/charts/ToggleLabelAlignment.swf
    Memory Snapshot (Fri Oct 26 10:11:48 EDT 2007)
    Memory Snapshot (Fri Oct 26 10:12:03 EDT 2007)
    Performance Profile (Fri Oct 26 10:06:13 EDT 2007 --> Fri Oct 26 10:06:47 EDT 2007)
```

1 Select the application in the Profile view.

2 Click the Suspend button. Flex marks the application in the Profile view with `[Suspended]`.

3 To resume profiling the application, select the application and click the Resume button. Flex marks the application in the Profile view with `[Running]`.

Stop profiling a Flex application

1 Select the application in the Profile view.

2 Click the Terminate button to end the profiling session. This does not close the browser or kill the Player process. You must do that manually.

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.

About the profiler buttons

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 terminated already.
	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 "About garbage collection" on page 179.
	Take Memory Snapshot	Stores the memory usage of an application so that you can examine it or compare it to other snapshots. 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. To open the new memory snapshot in the Memory Snapshot view, double-click the memory snapshot entry. 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. For more information about memory snapshots, see "Using the Memory Snapshot view" on page 168.
	Find Loitering Objects	Compares two memory snapshots in the Loitering Objects view. This option is enabled only when two memory snapshots are selected and when you selected Enable Memory Profiling in the launch dialog box. For more information about the Loitering Objects view, see "Using the Loitering Objects view" on page 177.
≡	View Allocation Trace	Compares the methods between two memory snapshots in the Allocation Trace view. This option is enabled only when two memory snapshots are selected, and when you select Enable Memory Profiling in the launch dialog box. For more information about the Allocation Trace view, see "Using the Allocation Trace view" on page 171.
	Reset Performance Data	Clears the performance profiling data. 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. 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. For more information about the Performance Profile view, see "Using the Performance Profile view" on page 174.

Button	Name	Description
⏱	Capture Performance Profile	Stores a new performance snapshot as a child of the selected application. 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. To open the Performance Profile view, double-click the performance snapshot entry. For more information about the Performance Profile view, see "Using the Performance Profile view" on page 174.
✖	Delete	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. This option is enabled only when a performance or memory snapshot is selected.

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 "About profiler filters" on page 183.
⏱	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.

Saving and loading profiling data

After you run the profiler, you can save the data so that you can compare a snapshot from the current profiling session with a snapshot you take after you make changes to your code. This 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 includes all performance profiles, memory snapshots, and allocation traces. Flex 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. You should create a new folder for each set of profiling data that you want to save. Otherwise, the new data will overwrite the old data if you choose the same folder.

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. Flex 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 Flex Builder.

Delete profiling data

1 Select the snapshot from the application in the Profile view.

2 Click the Delete button.

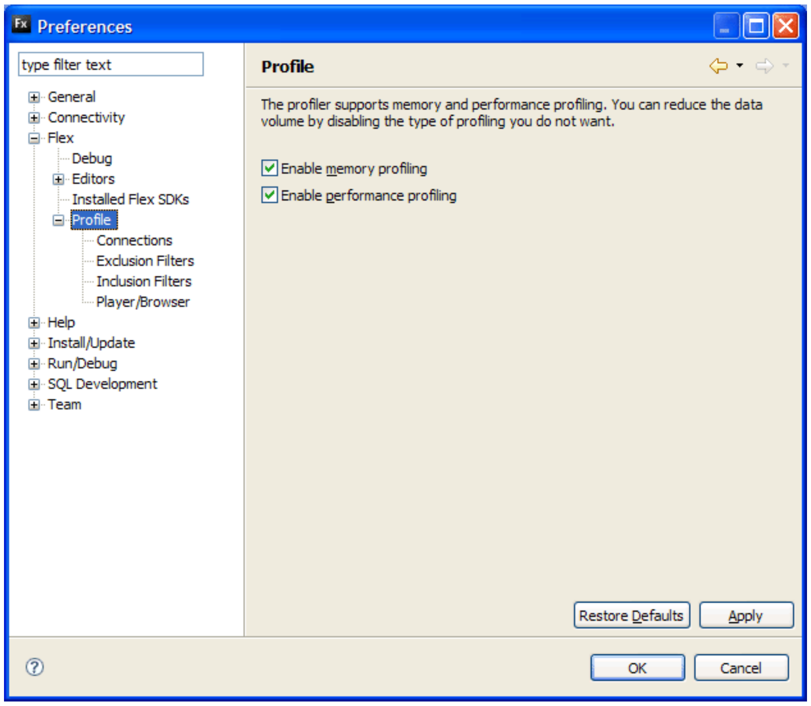
Setting profiler preferences

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 Flex Builder preferences for multiple profiling sessions

❖ Open the Preferences dialog and select Flex > Profiler.

The following example shows the profiler preferences:



Profile

The profiler supports memory and performance profiling. You can reduce the data volume by disabling the type of profiling you do not want.

[✓] Enable memory profiling
[✓] Enable performance profiling

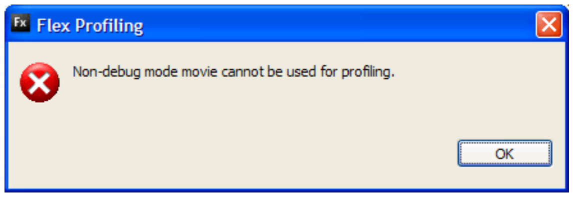
Select the options under the Profiler menu to navigate to the various options. The following table describes the preferences you can set:

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 Flex 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 "About profiler filters" on page 183.
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 "About profiler filters" on page 183.
Player/Browser	Lets you define the location of the Flash Player executable and browser executable that Flex Builder uses to run your profiled application.

Profiling external applications

In addition to profiling applications that you are developing in Flex 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.

The external application must be compiled with debugging information in it before you can profile it. To compile an application with debugging information in it, add `debug=true` to the compiler arguments. Flex Builder also returns this error if you specify an unknown file type, such as HTML or JSP. You can only specify SWF files as external applications to profile. If the application has no debugging information, Flex Builder returns the following error:



Non-debug mode movie cannot be used for profiling.

If you get this error, recompile the application with the `debug` compiler option set to `true` and launch it again.

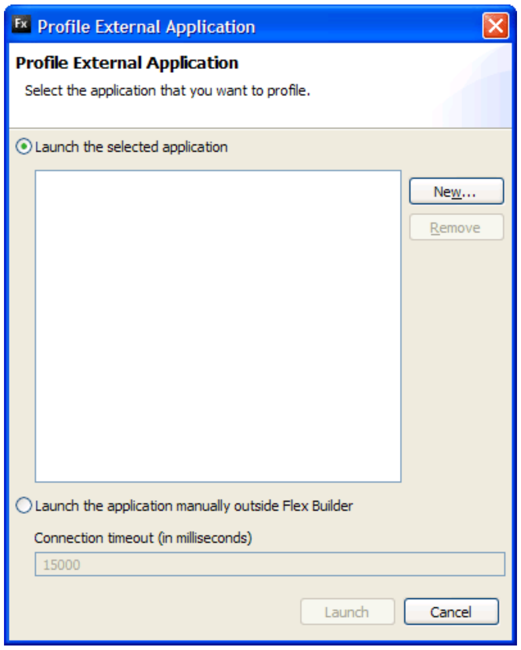
For the SWF file, you can specify either a URL or a file system location. If you specify a URL, Flex Builder launches the application's SWF file within the default browser. The browser must be using the debugger version of Flash Player to successfully profile the application.

If you specify a file system location for the SWF file, Flex Builder opens the application within the debugger version of the stand-alone Flash Player. In general, you should request the file by using a URL. Running applications in the stand-alone version of Flash Player can produce unexpected results, especially if your application uses remote services or network calls.

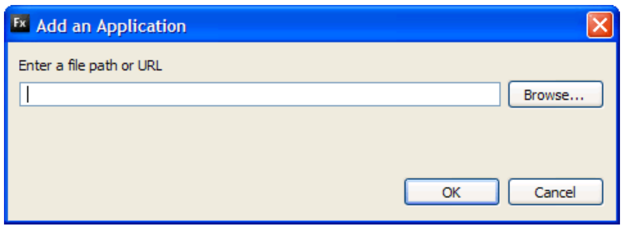
Profile an external application

1 Change to the Flex Profiling perspective.

2 Select Profile > Profile External Application. The Profile External Application dialog box appears.


Profile External Application
Profile External Application
Select the application that you want to profile.
Launch the selected application
New...
Remove
Launch the application manually outside Flex Builder
Connection timeout (in milliseconds)
15000
Launch
Cancel

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 Flex 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, Flex Builder launches the application within the default browser. If you specified a file system location for the application, Flex 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, Flex Builder returns an error. Recompile the application with the `debug` compiler option set to `true` and launch it again.

About the profiler views

The Flex 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 "Saving and loading profiling data" on page 162.
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 "Viewing information in the Live Objects view" on page 167.
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 "Using the Memory Snapshot view" on page 168.
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 "Using the Loitering Objects view" on page 177.
Allocation Trace	Displays method statistics when comparing two memory snapshots. You view the Allocation Trace view by selecting two memory snapshots and then clicking the View Allocation Trace button. For more information, see "Using the Allocation Trace view" on page 171.
Object References	Displays objects and the objects that reference them. You view the Object References view by double-clicking a class name in the Memory Snapshot or Loitering Objects views. For more information, see "Using the Object References view" on page 169.
Object Statistics	Displays details about the caller and callee of the selected group of objects. You view the Object Statistics view by double-clicking an entry in the Allocation Trace view. For more information, see "Using the Object Statistics view" on page 172.
Performance Profile	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. You view the Performance Profile view by double-clicking one of the performance snapshots in the Profile view. For more information, see "Using the Performance Profile view" on page 174.

View	Description
Method Statistics	Displays the performance statistics of the selected group of methods.
	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.
	For more information, see "Identifying method performance characteristics" on page 176.
Memory Usage	Graphically displays peak memory usage and current memory usage over time.
	For more information, see "Using the Memory Usage graph" on page 178.

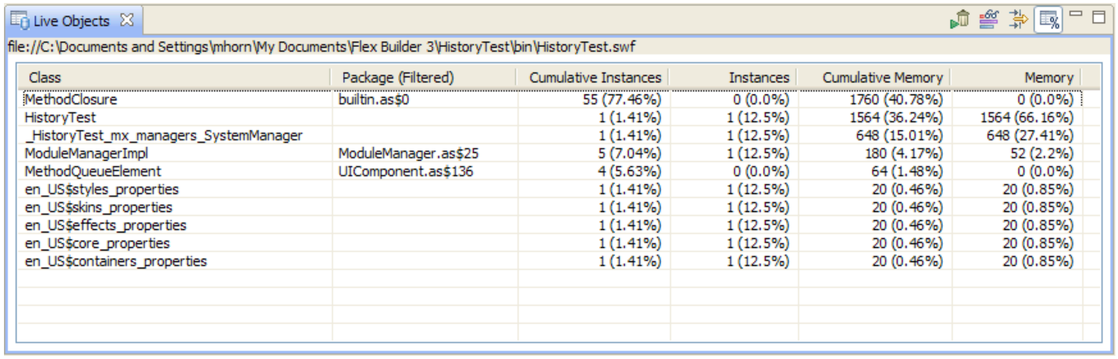
Viewing information in the Live Objects view

The Live Objects view displays information about the classes that the current application uses. This view shows which classes 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, you must 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:



file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\HistoryTest\bin\HistoryTest.swf

Class	Package (Filtered)	Cumulative Instances	Instances	Cumulative Memory	Memory
MethodClosure	builtin.as$0	55 (77.46%)	0 (0.0%)	1760 (40.78%)	0 (0.0%)
HistoryTest		1 (1.41%)	1 (12.5%)	1564 (36.24%)	1564 (66.16%)
_HistoryTest_mx_managers_SystemManager		1 (1.41%)	1 (12.5%)	648 (15.01%)	648 (27.41%)
ModuleManagerImpl	ModuleManager.as$25	5 (7.04%)	1 (12.5%)	180 (4.17%)	52 (2.2%)
MethodQueueElement	UIComponent.as$136	4 (5.63%)	0 (0.0%)	64 (1.48%)	0 (0.0%)
en_US$styles_properties		1 (1.41%)	1 (12.5%)	20 (0.46%)	20 (0.85%)
en_US$skins_properties		1 (1.41%)	1 (12.5%)	20 (0.46%)	20 (0.85%)
en_US$effects_properties		1 (1.41%)	1 (12.5%)	20 (0.46%)	20 (0.85%)
en_US$core_properties		1 (1.41%)	1 (12.5%)	20 (0.46%)	20 (0.85%)
en_US$containers_properties		1 (1.41%)	1 (12.5%)	20 (0.46%)	20 (0.85%)

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 file name 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 is being used by objects. 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 "About garbage collection" on page 179.

You limit the data in the Live Objects view by using the profiler filters. For more information, see "About profiler filters" on page 183.

Using the 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, you must enable memory profiling when you start the profiler. This is the default setting.

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.

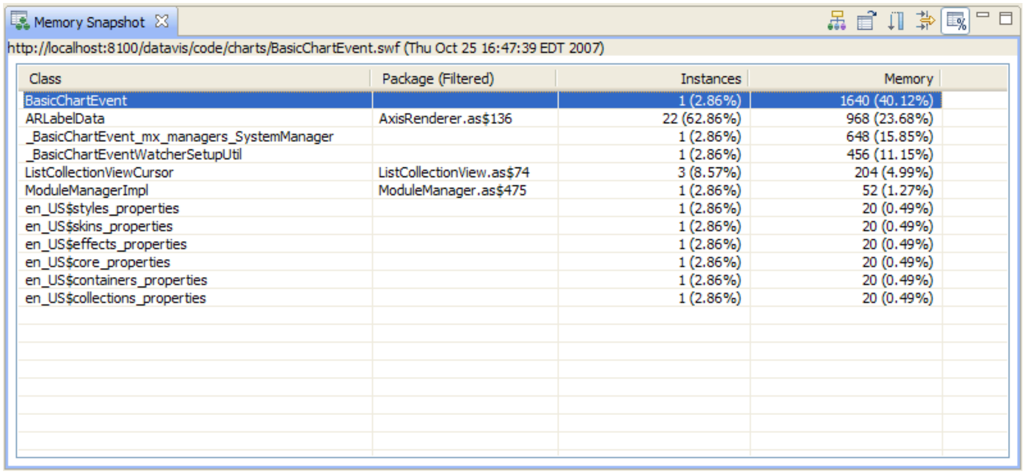
3 Select the application in the Profile view.

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:



http://localhost:8100/datavis/code/charts/BasicChartEvent.swf (Thu Oct 25 16:47:39 EDT 2007)

Class	Package (Filtered)	Instances	Memory
BasicChartEvent		1 (2.86%)	1640 (40.12%)
ARLabelData	AxisRenderer.as$136	22 (62.86%)	968 (23.68%)
_BasicChartEvent_mx_managers_SystemManager		1 (2.86%)	648 (15.85%)
_BasicChartEventWatcherSetupUtil		1 (2.86%)	456 (11.15%)
ListCollectionViewCursor	ListCollectionView.as$74	3 (8.57%)	204 (4.99%)
ModuleManagerImpl	ModuleManager.as$475	1 (2.86%)	52 (1.27%)
en_US$styles_properties		1 (2.86%)	20 (0.49%)
en_US$skins_properties		1 (2.86%)	20 (0.49%)
en_US$effects_properties		1 (2.86%)	20 (0.49%)
en_US$core_properties		1 (2.86%)	20 (0.49%)
en_US$containers_properties		1 (2.86%)	20 (0.49%)
en_US$collections_properties		1 (2.86%)	20 (0.49%)

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 file name 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 you should focus on for memory optimizations or to find memory leaks. You do this by creating multiple memory snapshots at different points in time and then comparing 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 "Saving and loading profiling data" on page 162.

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 "Using the Object References view" on page 169.

You can also limit the data in the Memory Snapshot view by using the profiler filters. For more information, see "About profiler filters" on page 183.

Using the Object References view

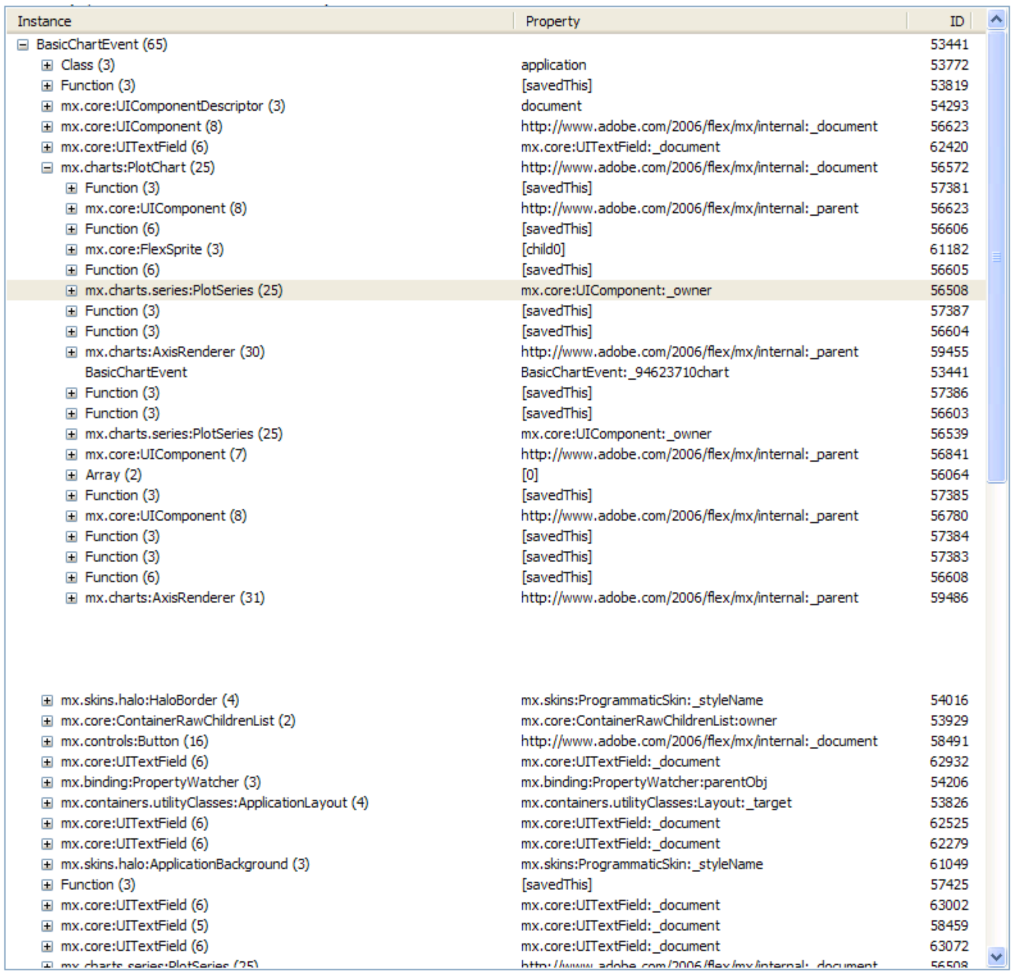
The Object References view displays stack traces for classes that were instantiated in the application.

To open the Object References view, double-click a class name in the Memory Snapshot or Loitering Objects views. The Object References view displays information about the selected class's instances.

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 the class name is the total number of references to the current object. You cannot view the number of forward references for an object. If no objects hold references to the specified object, then there will be no objects 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:



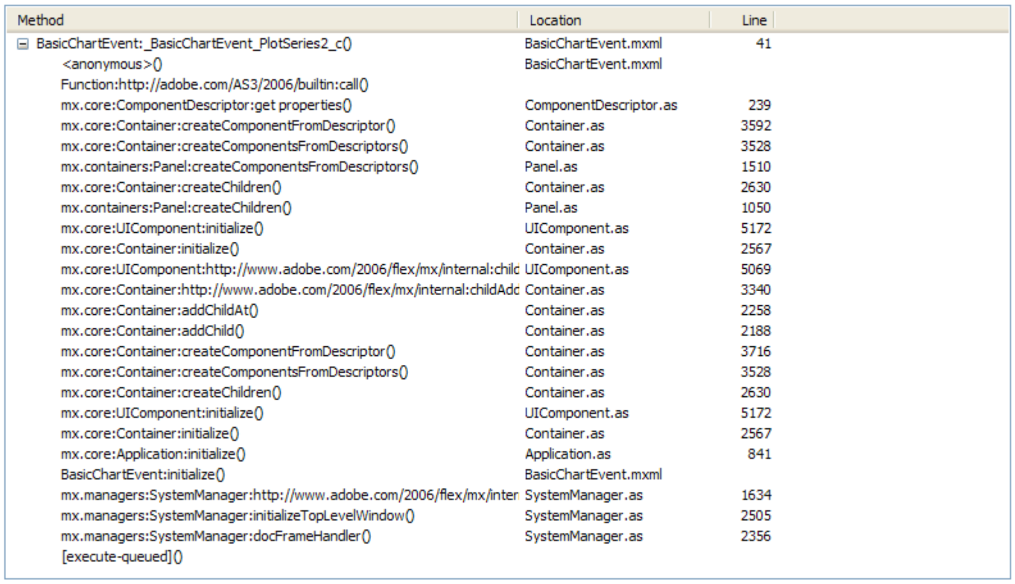
Instance	Property	ID
⊟ BasicChartEvent (65)		53441
⊞ Class (3)	application	53772
⊞ Function (3)	[savedThis]	53819
⊞ mx.core:UIComponentDescriptor (3)	document	54293
⊞ mx.core:UIComponent (8)	http://www.adobe.com/2006/flex/mx/internal:_document	56623
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	62420
⊟ mx.charts:PlotChart (25)	http://www.adobe.com/2006/flex/mx/internal:_document	56572
⊞ Function (3)	[savedThis]	57381
⊞ mx.core:UIComponent (8)	http://www.adobe.com/2006/flex/mx/internal:_parent	56623
⊞ Function (6)	[savedThis]	56606
⊞ mx.core:FlexSprite (3)	[child0]	61182
⊞ Function (6)	[savedThis]	56605
⊞ mx.charts.series:PlotSeries (25)	mx.core:UIComponent:_owner	56508
⊞ Function (3)	[savedThis]	57387
⊞ Function (3)	[savedThis]	56604
⊞ mx.charts:AxisRenderer (30)	http://www.adobe.com/2006/flex/mx/internal:_parent	59455
BasicChartEvent	BasicChartEvent:_94623710chart	53441
⊞ Function (3)	[savedThis]	57386
⊞ Function (3)	[savedThis]	56603
⊞ mx.charts.series:PlotSeries (25)	mx.core:UIComponent:_owner	56539
⊞ mx.core:UIComponent (7)	http://www.adobe.com/2006/flex/mx/internal:_parent	56841
⊞ Array (2)	[0]	56064
⊞ Function (3)	[savedThis]	57385
⊞ mx.core:UIComponent (8)	http://www.adobe.com/2006/flex/mx/internal:_parent	56780
⊞ Function (3)	[savedThis]	57384
⊞ Function (3)	[savedThis]	57383
⊞ Function (6)	[savedThis]	56608
⊞ mx.charts:AxisRenderer (31)	http://www.adobe.com/2006/flex/mx/internal:_parent	59486
⊞ mx.skins.halo:HaloBorder (4)	mx.skins:ProgrammaticSkin:_styleName	54016
⊞ mx.core:ContainerRawChildrenList (2)	mx.core:ContainerRawChildrenList:owner	53929
⊞ mx.controls:Button (16)	http://www.adobe.com/2006/flex/mx/internal:_document	58491
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	62932
⊞ mx.binding:PropertyWatcher (3)	mx.binding:PropertyWatcher:parentObj	54206
⊞ mx.containers.utilityClasses:ApplicationLayout (4)	mx.containers.utilityClasses:Layout:_target	53826
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	62525
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	62279
⊞ mx.skins.halo:ApplicationBackground (3)	mx.skins:ProgrammaticSkin:_styleName	61049
⊞ Function (3)	[savedThis]	57425
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	63002
⊞ mx.core:UITextField (5)	mx.core:UITextField:_document	58459
⊞ mx.core:UITextField (6)	mx.core:UITextField:_document	63072
⊞ mx.charts.series:PlotSeries (25)	http://www.adobe.com/2006/flex/mx/internal:_document	56508

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.
Property	The property of the object that holds a reference to the specified object. For example, if you have object o with a property `i`, and assign that property to point to your button's label: `o.i = myButton.label;` That creates a reference to `myButton.label` from property `i`.
id	The reference ID of the object that holds the reference to the selected 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:get properties()	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:chilc	UIComponent.as	5069
mx.core:Container:http://www.adobe.com/2006/flex/mx/internal:childAdc	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:docFrameHandler()	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.

Using the Allocation Trace view

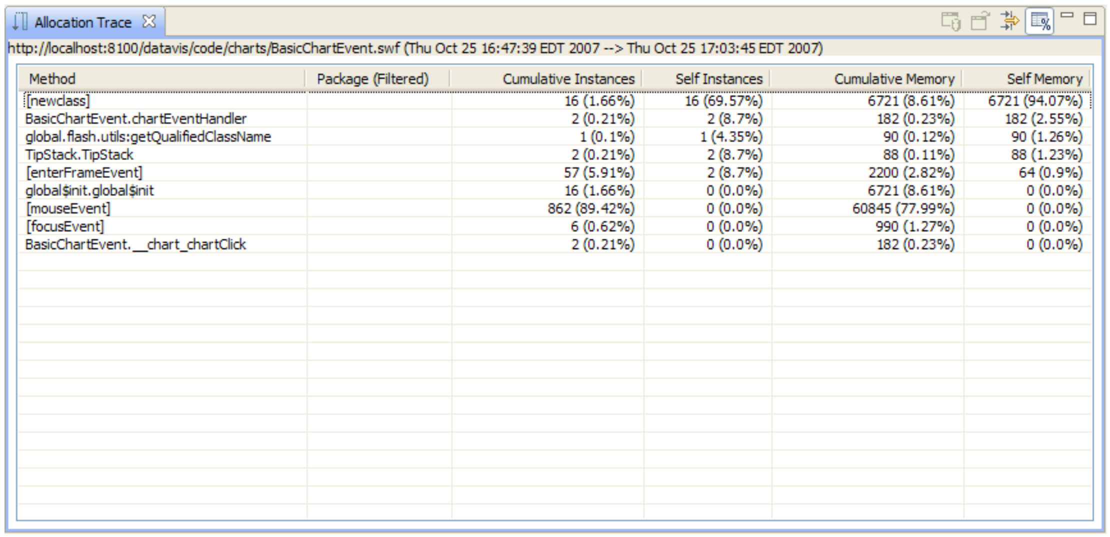
The Allocation Trace view shows which methods were called between two memory snapshots and how much memory was consumed during those method calls. To open the Allocation Trace view, you select two memory snapshots, and then click the View Allocation Trace button. For information on recording a memory snapshot, see "Using the Memory Snapshot view" on page 168.

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 hot spots.

To use the Allocation Trace view, you must enable allocation traces when you start the profiler. The default is disabled.

The following example shows the Allocation Trace view:



http://localhost:8100/datavis/code/charts/BasicChartEvent.swf (Thu Oct 25 16:47:39 EDT 2007 --> Thu Oct 25 17:03:45 EDT 2007)

Method	Package (Filtered)	Cumulative Instances	Self Instances	Cumulative Memory	Self Memory
[newclass]		16 (1.66%)	16 (69.57%)	6721 (8.61%)	6721 (94.07%)
BasicChartEvent.chartEventHandler		2 (0.21%)	2 (8.7%)	182 (0.23%)	182 (2.55%)
global.flash.utils:getQualifiedClassName		1 (0.1%)	1 (4.35%)	90 (0.12%)	90 (1.26%)
TipStack.TipStack		2 (0.21%)	2 (8.7%)	88 (0.11%)	88 (1.23%)
[enterFrameEvent]		57 (5.91%)	2 (8.7%)	2200 (2.82%)	64 (0.9%)
global$init.global$init		16 (1.66%)	0 (0.0%)	6721 (8.61%)	0 (0.0%)
[mouseEvent]		862 (89.42%)	0 (0.0%)	60845 (77.99%)	0 (0.0%)
[focusEvent]		6 (0.62%)	0 (0.0%)	990 (1.27%)	0 (0.0%)
BasicChartEvent.__chart_chartClick		2 (0.21%)	0 (0.0%)	182 (0.23%)	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 file name 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 Flex profiler works" on page 157.

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 "Using the Object Statistics view" on page 172.

You limit the data in the Allocation Trace view by using the profiler filters. For more information, see "About profiler filters" on page 183.

Using the Object Statistics view

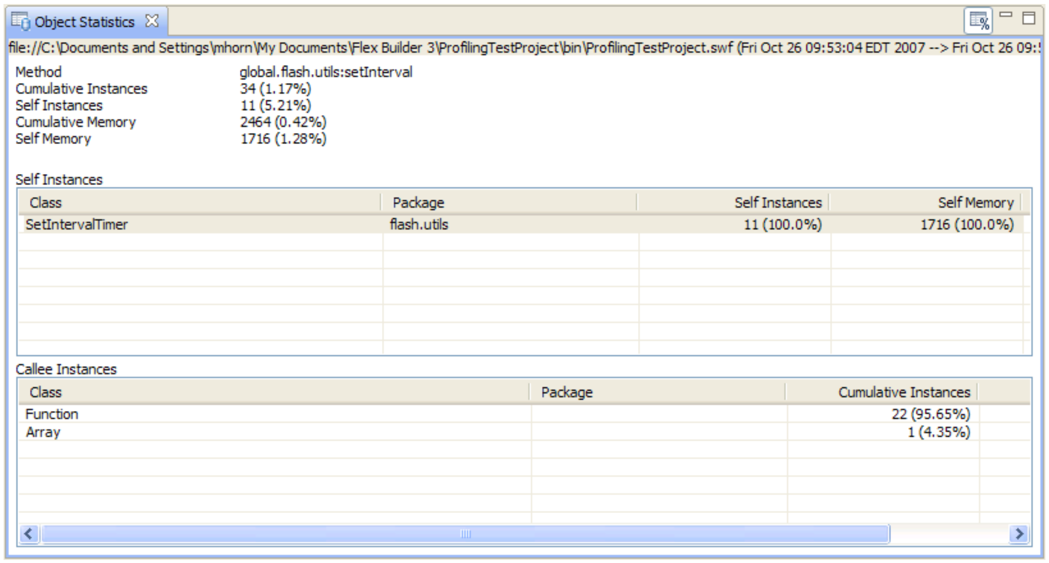
The Object Statistics view shows the performance statistics of the selected group of objects. This view helps you identify which methods call a disproportionate number of other methods. It also shows you how much memory the objects instantiated in those method calls consume. You use the Object Statistics 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:



file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\ProfilingTestProject\bin\ProfilingTestProject.swf (Fri Oct 26 09:53:04 EDT 2007 --> Fri Oct 26 09:

Method	global.flash.utils:setInterval
Cumulative Instances	34 (1.17%)
Self Instances	11 (5.21%)
Cumulative Memory	2464 (0.42%)
Self Memory	1716 (1.28%)

Self Instances

Class	Package	Self Instances	Self Memory
SetIntervalTimer	flash.utils	11 (100.0%)	1716 (100.0%)

Callee Instances

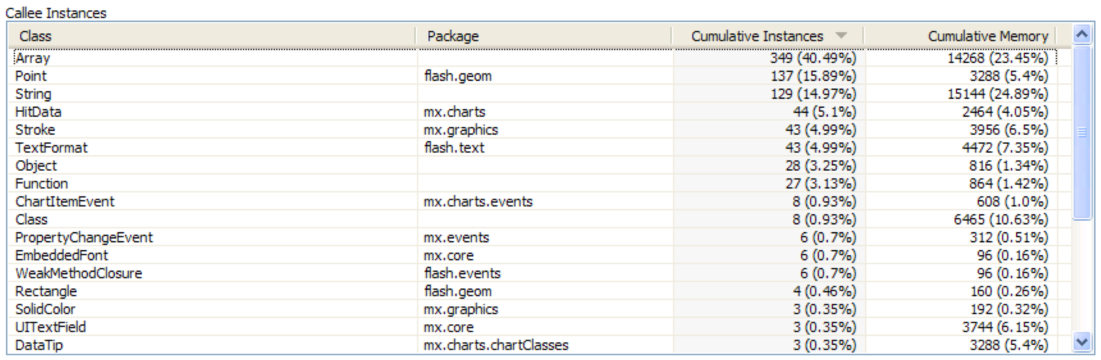
Class	Package	Cumulative Instances
Function		22 (95.65%)
Array		1 (4.35%)

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 file name 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.

Column	Description
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
Array		349 (40.49%)	14268 (23.45%)
Point	flash.geom	137 (15.89%)	3288 (5.4%)
String		129 (14.97%)	15144 (24.89%)
HitData	mx.charts	44 (5.1%)	2464 (4.05%)
Stroke	mx.graphics	43 (4.99%)	3956 (6.5%)
TextFormat	flash.text	43 (4.99%)	4472 (7.35%)
Object		28 (3.25%)	816 (1.34%)
Function		27 (3.13%)	864 (1.42%)
ChartItemEvent	mx.charts.events	8 (0.93%)	608 (1.0%)
Class		8 (0.93%)	6465 (10.63%)
PropertyChangeEvent	mx.events	6 (0.7%)	312 (0.51%)
EmbeddedFont	mx.core	6 (0.7%)	96 (0.16%)
WeakMethodClosure	flash.events	6 (0.7%)	96 (0.16%)
Rectangle	flash.geom	4 (0.46%)	160 (0.26%)
SolidColor	mx.graphics	3 (0.35%)	192 (0.32%)
UITextField	mx.core	3 (0.35%)	3744 (6.15%)
DataTip	mx.charts.chartClasses	3 (0.35%)	3288 (5.4%)

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 file name 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.

Using the 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, you must enable performance profiling when you start the profiler. This is the default setting.

Generate a performance profile

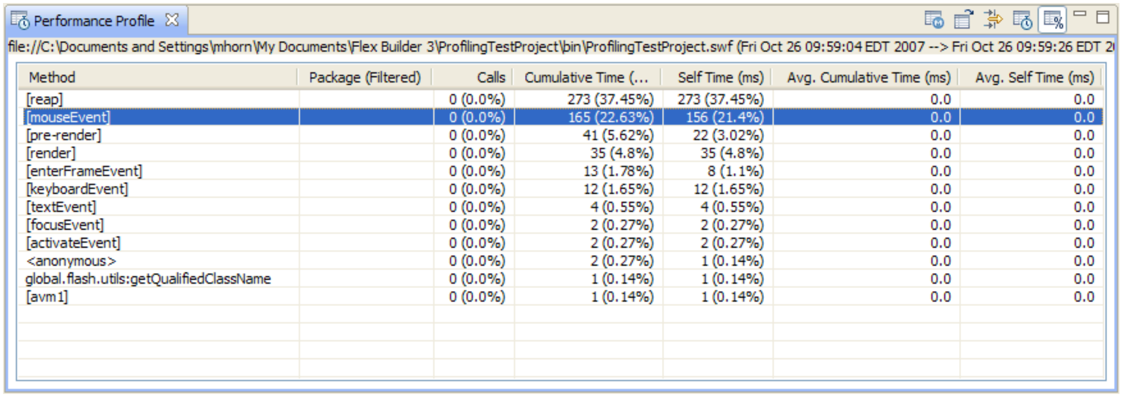
1. Start a profiling session with performance profiling enabled.
2. Interact with your application until you reach the point where you want to start profiling.
3. Click the Reset Performance Data button.
4. Interact with your application and perform the actions to profile.

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:



file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\ProfilingTestProject\bin\ProfilingTestProject.swf (Fri Oct 26 09:59:04 EDT 2007 --> Fri Oct 26 09:59:26 EDT 2

Method	Package (Filtered)	Calls	Cumulative Time (...	Self Time (ms)	Avg. Cumulative Time (ms)	Avg. Self Time (ms)
[reap]		0 (0.0%)	273 (37.45%)	273 (37.45%)	0.0	0.0
[mouseEvent]		0 (0.0%)	165 (22.63%)	156 (21.4%)	0.0	0.0
[pre-render]		0 (0.0%)	41 (5.62%)	22 (3.02%)	0.0	0.0
[render]		0 (0.0%)	35 (4.8%)	35 (4.8%)	0.0	0.0
[enterFrameEvent]		0 (0.0%)	13 (1.78%)	8 (1.1%)	0.0	0.0
[keyboardEvent]		0 (0.0%)	12 (1.65%)	12 (1.65%)	0.0	0.0
[textEvent]		0 (0.0%)	4 (0.55%)	4 (0.55%)	0.0	0.0
[focusEvent]		0 (0.0%)	2 (0.27%)	2 (0.27%)	0.0	0.0
[activateEvent]		0 (0.0%)	2 (0.27%)	2 (0.27%)	0.0	0.0
<anonymous>		0 (0.0%)	2 (0.27%)	1 (0.14%)	0.0	0.0
global.flash.utils:getQualifiedClassName		0 (0.0%)	1 (0.14%)	1 (0.14%)	0.0	0.0
[avm1]		0 (0.0%)	1 (0.14%)	1 (0.14%)	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 "How the Flex profiler works" on page 157.
Package	The package that the class is in. If the class is not in a package, then the value of this field is the file name 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.
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 "Identifying method performance characteristics" on page 176.

You limit the data in the Performance Profile view by using the profiler filters. For more information, see "About profiler filters" on page 183.

You can save performance profiles for later use. For more information, see "Saving and loading profiling data" on page 162.

Identifying 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 "Using the Performance Profile view" on page 174.

View method details in the Method Statistics view

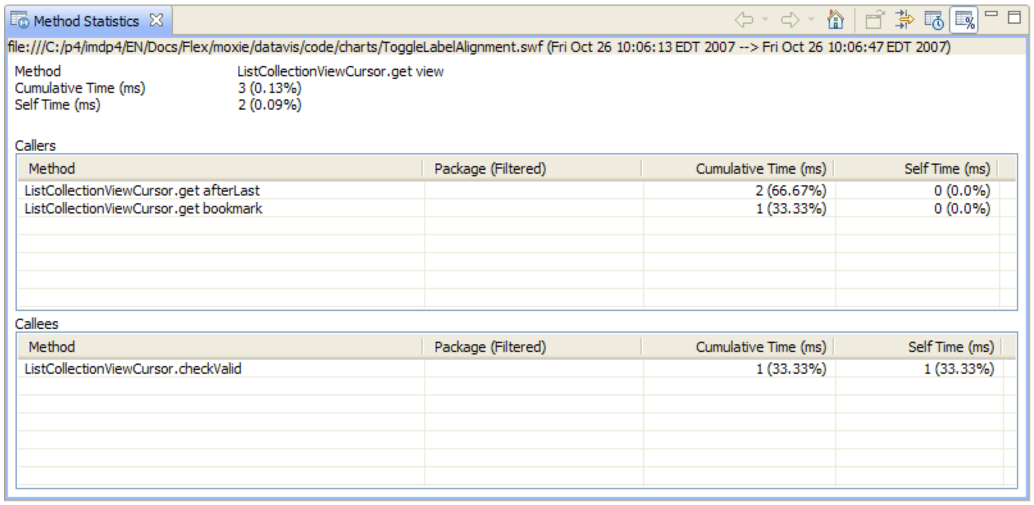
1 Double-click a row in the Performance Profile view or select a method in the Performance Profile view.

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.
- Callees table—Details about the methods that were called by the selected method.

The following example shows the method summary and the Callers and Callees tables of the Method Statistics view:



file:///C:/p4/imdp4/EN/Docs/Flex/moxie/datavis/code/charts/ToggleLabelAlignment.swf (Fri Oct 26 10:06:13 EDT 2007 --> Fri Oct 26 10:06:47 EDT 2007)

Method	ListCollectionViewCursor.get view
Cumulative Time (ms)	3 (0.13%)
Self Time (ms)	2 (0.09%)

Callers

Method	Package (Filtered)	Cumulative Time (ms)	Self Time (ms)
ListCollectionViewCursor.get afterLast		2 (66.67%)	0 (0.0%)
ListCollectionViewCursor.get bookmark		1 (33.33%)	0 (0.0%)

Callees

Method	Package (Filtered)	Cumulative Time (ms)	Self Time (ms)
ListCollectionViewCursor.checkValid		1 (33.33%)	1 (33.33%)

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 file name 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 file name 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.

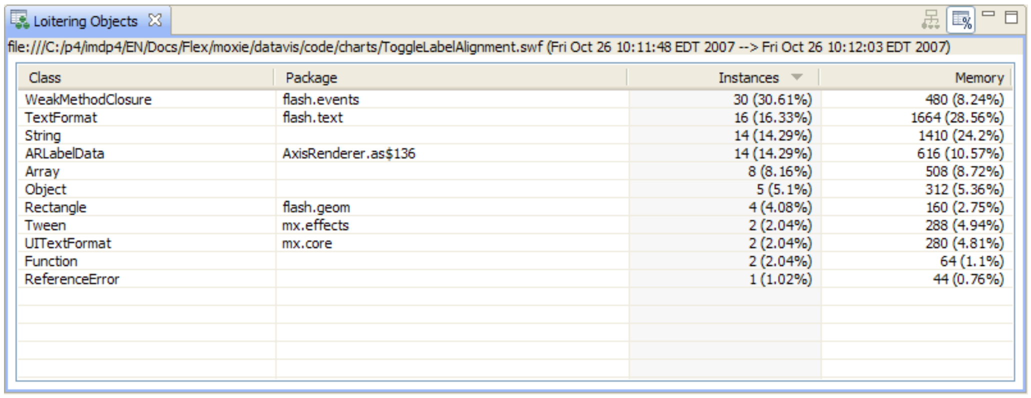
You can limit the data in the Method Statistics view by using the profiler filters. For more information, see "About profiler filters" on page 183.

Using the 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 "Using the Memory Snapshot view" on page 168.

The following example shows the Loitering Objects view:



file:///C:/p4/imdp4/EN/Docs/Flex/moxie/datavis/code/charts/ToggleLabelAlignment.swf (Fri Oct 26 10:11:48 EDT 2007 --> Fri Oct 26 10:12:03 EDT 2007)

Class	Package	Instances	Memory
WeakMethodClosure	flash.events	30 (30.61%)	480 (8.24%)
TextFormat	flash.text	16 (16.33%)	1664 (28.56%)
String		14 (14.29%)	1410 (24.2%)
ARLabelData	AxisRenderer.as$136	14 (14.29%)	616 (10.57%)
Array		8 (8.16%)	508 (8.72%)
Object		5 (5.1%)	312 (5.36%)
Rectangle	flash.geom	4 (4.08%)	160 (2.75%)
Tween	mx.effects	2 (2.04%)	288 (4.94%)
UITextFormat	mx.core	2 (2.04%)	280 (4.81%)
Function		2 (2.04%)	64 (1.1%)
ReferenceError		1 (1.02%)	44 (0.76%)

The following table describes the columns in the Loitering Objects view:

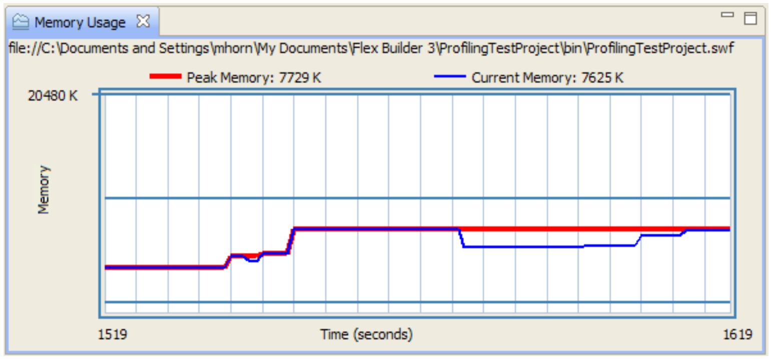
Column	Description
Class	The classes that were created but not destroyed during the snapshot interval.
Package	The package that each class is in. If the class is not in a package, then the value of this field is the filename that this class is in. The number following the dollar sign is a unique ID for 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 “Locating memory leaks” on page 180.

Using the 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 the 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 will get very different results.

The following image shows the graph in the Memory Usage view:


Memory Usage
file://C:\Documents and Settings\mhorn\My Documents\Flex Builder 3\ProfilingTestProject\bin\ProfilingTestProject.swf
Peak Memory: 7729 K
Current Memory: 7625 K
20480 K
Memory
1519
Time (seconds)
1619

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 menu button in the Profile view and select Memory Usage.

About 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 will be 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 may 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.

Identifying problem areas

You can use a variety of techniques to identify problem areas in your applications by using the profiler.

Locating 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 of time but not garbage collected. By comparing two memory snapshots in the Loitering Objects view, you can determine which objects are still in memory after a particular series of events.

Find loitering objects

1 Create two memory snapshots. For more information, see "Create and view a memory snapshot" on page 168.

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 of time 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” on page 54 in *Building and Deploying Adobe Flex 3 Applications*.

Analyzing 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, you should 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.

Locating 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 "Using the Memory Snapshot view" on page 168.

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.

About 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 should appear in the profiler views. You do this 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.

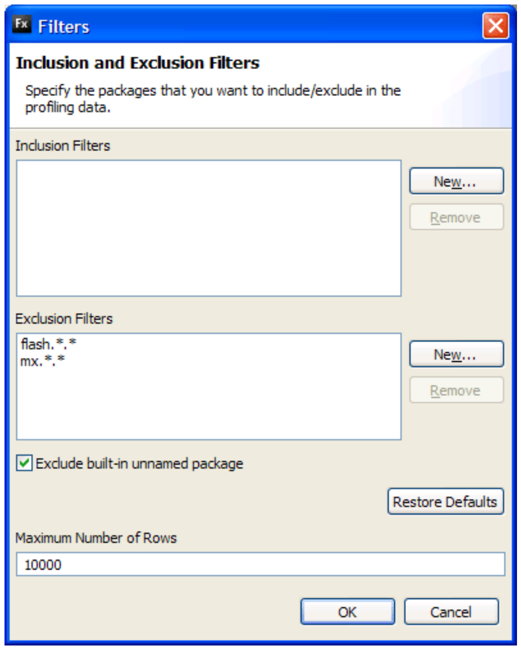
Inclusion filters The inclusion filters instruct the profiler to include in the profiler views only those packages that match the patterns in its pattern list. If you have a custom package named com.mycompany.*, for example, you can view details about only classes in this package by adding it to the inclusion filter.

The default exclusions are flash.*.* and 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.

Set default filter preferences

❖ Open the Preferences dialog and select Flex > Profiler > Inclusion Filters or Exclusion Filters.



Exclusion Filters:
flash.*.*
mx.*.*

☑ Exclude built-in unnamed package

Maximum Number of Rows: 10000

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 nonfiltered 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 14: Working with Data in Flex Builder

In Adobe® Flex® Builder™, you interact with data and the data-driven controls directly in your MXML and Action-Script 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 Flex Builder with a proxy service.

Topics

About working with data in Flex Builder

You work with data in Flex 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 Flex 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 the TileList or HorizontalList controls to present product data that includes images.
- You can provide standard navigation options by using containers such as the TabBar, LinkBar, 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 “Using Data-Driven Controls” on page 293 in the *Adobe Flex 3 Developer Guide*.

Data providers and collections

A *collection* object contains a data object, such as an Array or an XMList 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:

```
<?xml version="1.0"?>
```

```
<!-- dpcontrols\ArrayCollectionInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initData()">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;
            [Bindable]
            public var stateArray:ArrayCollection;

            public function initData():void {
                stateArray=new ArrayCollection(
                [{label:"AL", data:"Montgomery"},
                {label:"AK", data:"Juneau"},
                {label:"AR", data:"Little Rock"}]);
            }
        ]]>
    </mx:Script>

    <mx:ComboBox id="myComboBox" dataProvider="{stateArray}"/>
</mx:Application>
```

For more information about data providers and collections, see "Using Data Providers and Collections" on page 106 in the *Adobe Flex 3 Developer Guide*.

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 Flex 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 a Flex 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 Flex Builder to develop applications that access server-side data, you must 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 "Managing Flash Player security" on page 208.*

You can also use Flex Builder to build applications that use LiveCycle Data Services ES, a separate product that provides advanced data service features. LiveCycle Data Services ES provides proxying for remote procedure call (RPC) service applications as well as advanced security configuration. LiveCycle Data Services ES also provides the following data services:

Data Management Service Allows you to create applications that work with distributed data and also to 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 Flex applications and Java Message Service (JMS) applications.

Flex Builder provides a set of data wizards that automatically generate database applications that use PHP, J2EE, or ASP.NET. For more information, see "Automatically generating database applications" on page 186.

Data binding

In the code example in "Data providers and collections" on page 183, you may have noticed that the value of the ComboBox control's `dataProvider` property is `"{stateArray}"`. 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:

```
<mx:TextInput id="LNameInput"></mx:TextInput>
...
<mx:Label text="{LNameInput.text}"></mx:Label>
```

To bind data from one object to another, you use either the curly braces ({ }) syntax (as shown in the example) or the `<mx:Binding>` tag. For more information, see "Using data binding with data models" on page 981 and "Binding Data" on page 977 in the *Adobe Flex 3 Developer Guide*.

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 `<mx: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:

```
<mx:Model id="Employee">
<Employee>
    <name>
        <first>Jennifer</first>
        <last>Nadeau</last>
    </name>
    <age>30</age>
    <work_tel>555-555-5555</work_tel>
</Employee>
</mx:Model>
```

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 `<mx:Model>` tag's `source` property, as the following example shows:

```
<mx:Model source="content.xml" id="Contacts"/>
<mx:Model source="http://www.somesite.com/companyinfo.xml" id="myCompany"/>
```

For more information about data models, see "Storing Data" on page 1000 in the *Adobe Flex 3 Developer Guide*.

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, e-mail, 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. -->
<mx: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 run time, when the user enters the phone number into the TextInput control, the number is immediately validated.

You can, of course, 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” on page 1008 in the *Adobe Flex 3 Developer Guide*.

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 might 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” on page 1039 in the *Adobe Flex 3 Developer Guide*.

Automatically generating database applications

You can use Flex Builder data wizards to automatically generate create, read, update, delete (CRUD) database applications for existing PHP, J2EE, and ASP.NET server projects. From the Create Application from Database option on the Data menu, you can select an SQL database table and generate an entire application that includes both client-side and server-side code.

For information about creating server projects in which you can create database applications, see “Working with Projects” on page 27.

Generating a PHP database application

You can generate an application that uses PHP and a MySQL database. To generate this type of application, you must have a local PHP server and access to a target MySQL database table.

Generate PHP application code

1 If you have not already done so, create a Flex project that specifies PHP as its application server type.

2 Select Data > Create Application from Database.

3 From the Project list, select the PHP project to use.

4 From the Connection list, select a database connection profile. To create and use a new connection profile, click the New button and complete the text boxes. For more information, see “Directory structure and deployment considerations for PHP” on page 187.

5 From the Table list, select the database table to use.

6 From the Primary Key list, select the primary key of the database table if it is not already selected.

7 Click Next.

8 Specify a Main Source Folder and a Main Application File, or accept the default values. You can generate the PHP files in the bin-debug folder, which is the default location, or in the src folder. If you want to export the project after using the data wizard, you must generate the PHP files in the src folder; otherwise, the project is exported without the PHP files and does not function.

9 Click Next.

10 Deselect any database columns that you do not want to appear in the DataGrid that the wizard generates.

11 Choose a database filter column, or accept the default value.

12 Click Finish.

13 Select Run > External Tools > Run As > Run on Server.

14 Select Run > Run to run the application on the server.

Directory structure and deployment considerations for PHP

When you use the data wizard to generate a PHP application, the default location of the generated files is the [web_server_root]\[project_name]_debug directory. The directory includes a PEAR subdirectory that contains the PEAR XML parsing utility, and a history subdirectory that contains files for Flex history management.

The main deployment directory contains *tablename*.php, *databasename*conn.php, functionsinc.php, and XmlSerializer.class.php PHP files. It also contains a SWF file named *tablename*.swf and an HTML wrapper named *tablename*.html.

The project src directory contains the following files: *tablename*.mxml, *tablename*config.as, and *tablename*script.as.

Deploy an application to a remote server

1 Set up the host:

a Create a new database. Typically, the host has a naming convention like *sitename_dbname*.

b Create a web user with a password (for example, name: bigryans_webuser, pw: webuser). Give the web user the following privileges: insert, update, delete, and select.

c Create the database table or tables (you can export the SQL and run it in your MySQL administrator application, or you can create it manually); for example:

```
CREATE TABLE users (
userid int(10) unsigned NOT NULL auto_increment,
username varchar(255) collate latin1_general_ci NOT NULL,
emailaddress varchar(255) collate latin1_general_ci NOT NULL,
PRIMARY KEY (userid)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
```

2 Edit the Flexphpconn.php file. This file creates the connection to the database for your Flex application users:

```
$hostname_conn = "localhost"; // If your database and Flex app are on the same server
$database_conn = "bigryans_flexphp"; // name you gave it in step 1 $username_conn =
"bigryans_webuser"; // name you created in step 1 $password_conn = "webuser"; // password
you gave it in step 1
```

3 Upload the files. You need the following subset of the files that the wizard outputs:

- /history directory (if you want to support history management/deep linking)
- /PEAR directory
- AC_OETags.js
- Flexphpconn.php
- functions.inc.php
- playerProdInstall.swf (if you want to support express installation)

- Flex application files:
  - users.html (HTML wrapper)
  - users.swf (Flex application)
  - users.php (PHP functions for the Flex application)
  - XMLSerializer.class.php

You do not need to upload the *xxx*-profile.swf files, the *projectname*.swf file, or the *projectname*.html file.

Generating a J2EE database application

You can generate an application that uses Java and a SQL database. To generate this type of application, you must have a local Java application server and access to a target SQL database table. You can generate an application that uses a Java servlet to access a database and return data to the Flex application; or if you have LiveCycle Data Services ES, you can choose to generate an application that uses the data management service to access a database and dynamically distribute data to and from clients.

Generate J2EE application code

1 If you have not already done so, create a new Flex project that specifies Java 2 Enterprise Edition (J2EE) as the application server type.

2 Select Data > Create Application from Database.

3 From the Project list, select the J2EE project to use.

4 From the Connection list, select a database connection profile. To create and use a new connection profile, click the New button and complete the text boxes.

5 From the Table list, select the database table to use.

6 From the Primary Key list, select the primary key of the database table.

7 Click Next.

8 To use a specific Java package for the Java classes that will be generated, enter the package name in the Java Package text box.

9 By default, the names of the generated Java classes are prefixed with the name of the selected database table. To use a different prefix, enter it in the Class Name Prefix text box.

10 If your project uses LiveCycle Data Services ES, select whether to use LiveCycle Data Services ES or XML over HTTP for communication between the client and the server. If you select LiveCycle Data Services ES, the Data Management Service is used. If you select XML over HTTP, a servlet passes XML between the client and the server.

11 Click Next.

12 (Optional) Deselect any of the listed table fields to exclude them from the generated Flex DataGrid control.

13 Click Finish.

14 Select Run > External Tools > Run As > Run on Server.

15 Select Run > Run to run the application on the server.

Directory structure and deployment considerations for J2EE

The Flex Builder project's WebContent directory contains directories to be deployed into the root directory of the web application. Depending on the type of Java application server, the web application is deployed as a WAR file or, as is the case with Tomcat, as a linked web application directory. The Flex Builder project's src directory contains Java source files for classes that are deployed to the WEB-INF/classes directory of the web application. For projects that are compiled with Flex Builder, the contents of the Flex Builder project's bin and flex_src directories are included in the list of files that are deployed to the web application; however, the compiled SWF files are deployed rather than the corresponding MXML files that are in the flex_src directory. All of the files in those folders are deployed to the root of the web application.

Generating an ASP .NET database application

You can generate an application that uses ASP .NET and a Microsoft SQL Server database. To generate this type of application, you must have a local .NET installation and access to a target Microsoft SQL Server database table. You can generate an application that uses an ASP .NET page to access a database and return data to the Flex application.

Generate ASP .NET application code

1 If you have not already done so, create a new Flex project that specifies ASP .NET as the application server type. You can deploy the application to the Visual Studio Development Server or the Internet Information Server (IIS) web server that comes with Windows. If you use IIS, the deployment directory specified in the New Flex Project wizard must be defined in the IIS administration console as an ASP Application.

2 Select Data > Create Application from Database.

3 From the Project list, select the ASP .NET project to use.

4 From the Connection list, select a database connection profile. To create and use a new connection profile, click the New button and complete the text boxes.

5 From the Table list, select the database table to use.

6 From the Primary Key list, select the primary key of the database table.

7 Click Next.

8 Specify a .NET class name, or accept the default value.

9 Choose either C# with Web Services or VB with Web Services as the server language.

10 Click Next.

11 (Optional) Deselect any of the listed table fields to exclude them from the generated Flex DataGrid control.

12 Click Finish.

13 Select Run > External Tools > Run As > Run on Server.

14 Select Run > Run to run the application on the server.

Directory structure and deployment considerations for ASP .NET

When you run Project > Export Release Build, files are saved in the project bin_release directory and the optimized application runs from that location.

Creating a database connection profile

To generate a database application, you must have a database connection profile that provides information about the database that you want to use.

Create a connection profile

1 If the Create Application from Database dialog box is not already open, select Data > Create Application from Database.

2 Click the New button to the right of the Connection text box.

3 In the Create Connection Profile dialog box, enter a connection name in the Name text box. You can also complete the optional Description text box.

4 (Optional) When the wizard is finished or when the Data Source Explorer is opened, select the Autoconnect option to connect to the database.

5 Click New.

6 Complete the database connection text boxes. The fields vary depending on the type of database connection you choose. For PHP projects, you use MySQL. For ASP.NET projects, you use Microsoft SQL Server. For J2EE projects, you can select from a list of choices, but MySQL and Microsoft SQL Server are supported.

7 Click Test Connection to make sure that the connection is valid.

8 Click Next.

9 Click Finish.

Automatically generating web service proxies

You can use the Flex Builder Import Web Service feature to automatically generate connection code for invoking SOAP-based web service operations. The generated code includes client-side (ActionScript) web service proxy classes that return strongly typed objects. You can use the generated code to call web service operations directly and to pass strongly typed objects as operation parameters.

Generate client code for a WSDL document

1 Select Data > Import Web Services (WSDL).

2 Select the project source folder in which to save the generated code.

3 Click Next.

4 Specify the URI of the WSDL to use. If your project uses LiveCycle Data Services ES, you can optionally select a LiveCycle Data Services ES web service destination. For a LiveCycle Data Services ES destination to appear in the Import Web Service menu, it must be defined in the proxy-config.xml file and be configured in the following way:

- The `adapter="soap-proxy"` attribute must be specified on the destination element. This implies that you have an adapter definition with the `id="soap-proxy"` attribute defined somewhere else in your configuration files.
- The destination must also have an `id` attribute (the value of the id attribute is displayed in the pop-up menu).
- The destination must have a `wsdl` child element. Its text value is prefilled in the WSDL URI text box in the wizard.

5 Click Next.

6 Deselect any of the listed web service operations for which you do not want to generate code.

7 (Optional) Select a different service and or port from the Service and Port pop-up menus.

8 (Optional) Change the default package name and the main class name.

9 Click Finish to generate the ActionScript proxy classes.

Generated web service proxy files

The Import Web Service feature introspects a WSDL file and generates the following types of ActionScript class files:

Generated file	Description
Base*ServiceName*.as	A base implementation of the web service. This class contains the internal code that maps the operations from the WSDL file to Flex WSDLOperation instances and sets the corresponding parameters.
I*Servicename*.as	The service interface that defines all of the methods that users can access.
Servicename.as	The concrete web service implementation.
Base*Servicename*Schema.as	A file containing the XSD schema of the web service as an ActionScript custom type.
*Operationname*ResultEvent.as	For each web service operation, Flex Builder generates a strongly typed event type class. You can use these classes to benefit from strongly typed results.
*Operationname*_request.as	For each web service operation that passes parameters to the server operation, Flex Builder generates a request wrapper class with the parameters as members. The request object is intended for use with the MXML tag syntax.
Type.as	For each complex type defined in the WSDL file, Flex Builder generates a class with the same name as the complex type or with the same name as the enclosing element.

Managing generated web service code

The Manage Web Services feature lets you add, update, or delete generated web service proxy code.

Generate web service code

1 Select Data > Manage Web Services.

2 Select a project from the tree of projects.

3 Click Add.

4 Click Next.

5 Enter a new WSDL URI or select an existing one from the pop-up menu. Optionally, if you have a LiveCycle Data Services ES installation, you can use a LiveCycle Data Services ES destination.

6 Complete the Configure Code Generation text boxes and click Finish.

Update generated web service code

1 Select Data > Manage Web Services.

2 Within a project, select the WSDL URI for which you want to update generated code.

3 Click Update.

4 Click Refresh to find out if the WSDL document has changed since you imported it.

5 If the WSDL document has changed, select the operations to update (if different from the default selections).

6 Click Finish to regenerate code based on the current version of the WSDL document.

Delete generated web service code

1 Select Data > Manage Web Services.

2 Within a project, select the WSDL URI for the generated code to delete.

3 Click Delete and then click Yes in the Confirm Imported WSDL Deletion dialog box.

Creating an application that uses the generated proxy code

You can use MXML tags or ActionScript to call a web service. Equivalent MXML-centric and ActionScript-centric applications use an address book web service that exposes the following operations:

- AddEntry lets you store someone's address.
- FindEntry lets you retrieve the address when you provide the person's name.

The AddEntry operation takes an Entry object as input and returns nothing. The FindEntry operation takes a string as input and returns an Entry object.

These sample applications demonstrate the following concepts for working with the code that Flex Builder generated from the WSDL document:

- You can call a web service operation directly, as the following example shows:

  `myWebService.callOperation(parameters)`

- The parameters passed to the web service operations are strongly typed and reflect the types that are described in the WSDL document.
- The generated code includes event listeners that are specific to each operation; these listeners return strongly typed results.

The MXML and ActionScript address book applications both use the following generated ActionScript classes:

Generated file	Description
BaseAddressBookService.as	A base implementation of the web service. This class contains the internal code that maps the operations from the WSDL file to Flex WSDLOperation instances and sets the corresponding parameters.
IAddressBookService.as	The service interface that defines all of the methods that users can access.
AddressBookService.as	The concrete web service implementation.
BaseAddressBookService-Schema.as	The XSD schema of the web service as an ActionScript custom type.
FindEntryResultEvent.as	A strongly typed event type object for the FindEntry operation. For each web service operation, Flex Builder generates a strongly typed event type class. You can use these classes to benefit from strongly typed results.
AddEntry_request.as	A request wrapper class for the AddEntry operation with the parameters as members. The request object is intended for use with the MXML tag syntax.
FindEntry_request.as	A request wrapper class for the FindEntry operation with the parameters as members. The request object is intended for use with the MXML tag syntax.
Entry.as	For each complex type defined in the WSDL file, Flex Builder generates a class with the same name as the complex type or with the same name as the enclosing element.

Calling a service with MXML

For an MXML-centric application, you create an instance of the web service in an MXML tag, as the following code snippet shows. Note that the application uses a custom namespace, ws; this namespace is declared as `xmlns:ws="com.adobe.*"` in the `<mx:Application>` tag.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:ws="com.adobe.*">
...
    <!-- This is how we create an instance of the AddressBookService generated class.-->
    <ws:AddressBookService id="myWebService">
        <!-- We need to specify the request variables that are passed to the corresponding
operations. -->
        <ws:addEntry_request_var>
```

```
            <!-- The addEntry operation has one input parameter. This parameter's type is
Entry. -->
            <ws:AddEntry_request>
                <ws:param0>
                    <ws:Entry>
                        <!-- We bind each of the members of an Entry object to the
corresponding text input field.-->
                        <ws:name>{tiNameInput.text}</ws:name>
                        <ws:city>{tiCityInput.text}</ws:city>
                        <ws:street>{tiStreetInput.text}</ws:street>
                        <ws:state>{tiStateInput.text}</ws:state>
                        <ws:postalCode>{tiPostalCodeInput.text}</ws:postalCode>
                    </ws:Entry>
                </ws:param0>
            </ws:AddEntry_request>
        </ws:addEntry_request_var>

        <!-- The findEntry operation has one input parameter of the String type. -->
        <ws:findEntry_request_var>
            <!-- We bind the corresponding text input field to the operation's input
parameter.
              Since the operation's input parameter has a simple type, you can use the
following syntax to do the data binding.
              Alternatively you could do:
              <ws:FindEntry_request>
                <ws:param0>{tiSearch.text}</ws:param0>
              </ws:FindEntry_request>
              -->
            <ws:FindEntry_request param0="{tiSearch.text}">
            </ws:FindEntry_request>
        </ws:findEntry_request_var>
    </ws:AddressBookService>
...
```

On the click event for the Add entry button, call the `addEntry_send()` method to pass the request to the service, as the following code snippet shows:

```
...
<mx:Button label="Add entry" labelPlacement="top" click="myWebService.addEntry_send();
clearInputFields()"/>
...
```

On the click event for the Find entry button, call the `findEntry_send()` method to pass the request to the service, as the following code snippet shows:

```
...
<mx:Button label="Add entry" labelPlacement="top" click="myWebService.addEntry_send();
clearInputFields()"/>
...
```

In a Form control, bind the results of the findEntry operation to Form fields:

```
...
                <mx:HBox id="hbResults" x="10" y="70" width="100%" verticalAlign="middle"
borderStyle="solid" visible="true">
                    <mx:Label text="Results:"/>
                    <!-- In the form that we use to display the data that is returned by the
web service, we bind each
                        member of the last result returned for the findEntry operation to
its corresponding text control. -->
                    <mx:Form width="293" height="181" borderStyle="solid">
                        <mx:FormItem label="Name">
```

```
                            <mx:Text id="txNameOutput"
text="{myWebService.findEntry_lastResult.name}"/>
...
                    </mx:Form>
                </mx:HBox>
...
```

The following example shows the complete MXML-centric address book application:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:ws="com.adobe.*">
    <mx:Script>
        <![CDATA[
            /**
                @private

                This method clears the input fields from the Add Entry view.
            */
            private function clearInputFields():void {
                tiNameInput.text = "";
                tiCityInput.text = "";
                tiStreetInput.text = "";
                tiStateInput.text = "";
                tiPostalCodeInput.text = "";
            }

        ]]>
    </mx:Script>

    <!-- This is how we create an instance of the AddressBookService generated class.-->
    <ws:AddressBookService id="myWebService">
        <!-- We need to specify the request variables that are passed to the corresponding
operations. -->
        <ws:addEntry_request_var>
            <!-- The addEntry operation has one input parameter. This parameter's type is
Entry. -->
            <ws:AddEntry_request>
                <ws:param0>
                    <ws:Entry>
                        <!-- We bind each of the members of an Entry object to the
corresponding text input field. -->
                        <ws:name>{tiNameInput.text}</ws:name>
                        <ws:city>{tiCityInput.text}</ws:city>
                        <ws:street>{tiStreetInput.text}</ws:street>
                        <ws:state>{tiStateInput.text}</ws:state>
                        <ws:postalCode>{tiPostalCodeInput.text}</ws:postalCode>
                    </ws:Entry>
                </ws:param0>
            </ws:AddEntry_request>
        </ws:addEntry_request_var>

        <!-- The findEntry operation has one input parameter of the String type. -->
        <ws:findEntry_request_var>
            <!-- We bind the corresponding text input field to the operation's input
parameter.
              Since the operation's input parameter has a simple type, you can use the
following syntax to do the data binding.
              Alternatively you could do:
              <ws:FindEntry_request>
                <ws:param0>{tiSearch.text}</ws:param0>
              </ws:FindEntry_request>
              -->
            <ws:FindEntry_request param0="{tiSearch.text}">
            </ws:FindEntry_request>

        </ws:findEntry_request_var>
    </ws:AddressBookService>

    <mx:Panel width="640" height="480" layout="absolute" verticalCenter="0"
horizontalCenter="0" title="AddressBook Client App">
        <mx:TabNavigator x="10" y="10" width="100%" height="100%">
            <mx:Canvas label="Add" width="100%" height="100%" id="tabAdd">
                <mx:HBox x="10" y="10" width="100%" verticalAlign="middle"
borderStyle="solid">
                    <mx:Label text="New entry details:"/>
                    <mx:Form width="293" height="181" borderStyle="solid">
                        <mx:FormItem label="Name">
                            <mx:TextInput id="tiNameInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="City">
                            <mx:TextInput id="tiCityInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Street">
                            <mx:TextInput id="tiStreetInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="State">
                            <mx:TextInput id="tiStateInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Postal Code">
                            <mx:TextInput id="tiPostalCodeInput"/>
                        </mx:FormItem>
                    </mx:Form>
                    <!-- On the click event for the Add entry button, we call the
addEntry_send() method of the web service that we defined earlier. -->
                    <mx:Button label="Add entry" labelPlacement="top"
click="myWebService.addEntry_send(); clearInputFields()"/>
                </mx:HBox>
            </mx:Canvas>
            <mx:Canvas label="Search" width="100%" height="100%" id="tabSearch">
                <mx:HBox x="10" y="10" width="100%">
                    <mx:Label text="Name"/>
                    <mx:TextInput id="tiSearch"/>
                    <!-- On the click event for the 'Find entry' button we call the
'findEntry_send()' method of the web service we defined earlier. -->
                    <mx:Button label="Find entry" click="myWebService.findEntry_send()"/>
                </mx:HBox>
                <mx:HBox id="hbNoResults" x="10" y="40" height="22" visible="false">
                    <mx:Text text="'No entry was found!"/>
                </mx:HBox>
                <mx:HBox id="hbResults" x="10" y="70" width="100%" verticalAlign="middle"
borderStyle="solid" visible="true">
                    <mx:Label text="Results:"/>
                    <!-- In the form that we use to display the data that is returned by the
web service, we bind each
                        member of the last result returned for the findEntry operation to
its corresponding text control. -->
                    <mx:Form width="293" height="181" borderStyle="solid">
                        <mx:FormItem label="Name">
                            <mx:Text id="txNameOutput"
text="{myWebService.findEntry_lastResult.name}"/>
                        </mx:FormItem>
                        <mx:FormItem label="City">
```

```
                            <mx:Text id="txCityOutput"
text="{myWebService.findEntry_lastResult.city}"/>
                        </mx:FormItem>
                        <mx:FormItem label="Street">
                            <mx:Text id="txStreetOutput"
text="{myWebService.findEntry_lastResult.street}"/>
                        </mx:FormItem>
                        <mx:FormItem label="State">
                            <mx:Text id="txStateOutput"
text="{myWebService.findEntry_lastResult.state}"/>
                        </mx:FormItem>
                        <mx:FormItem label="Postal Code">
                            <mx:Text id="txPostalCodeOutput"
text="{myWebService.findEntry_lastResult.postalCode}"/>
                        </mx:FormItem>
                    </mx:Form>
                </mx:HBox>
            </mx:Canvas>
        </mx:TabNavigator>
    </mx:Panel>
</mx:Application>
```

Calling a service with ActionScript

For an ActionScript-centric application, you create an instance of the web service in ActionScript, as the following code snippet shows:

```
...
<mx:Script>
    <![CDATA[
        import com.adobe.*;
        import mx.rpc.events.FaultEvent;
        import mx.controls.Alert;

        // Declare an instance of the generated web service class.
        public var agenda:AddressBookService;
        public function initApp():void
        {
            // Instantiate the new object.
            agenda = new AddressBookService();

...
        }
...
```

If you are using a LiveCycle Data Services ES destination, you pass the destination name to the service as a parameter of the constructor.

Add a result event listener (a function that you have previously defined) for the operation to call, as the following code snippet shows:

```
agenda.addEntryEventListener(myResultHandlingFunction);
```

In the `addEntry()` method, call the addEntry operation of the service and pass the correct values as arguments, as the following example shows:

```
agenda.addEntry(myparam0);
```

In the `searchEntry()` method, call the findEntry operation of the service and bind the results of the findEntry operation to Form fields:

```
...
        /**
        * The handleSearchResult() method is the result event handler for the
        * findEntry operation.
        * It displays the address of a given person by populating the corresponding
        * Form fields with the members of the strongly typed Entry result that
        * you get when the call to the operation succeeds. Or, if we have no record of the
person's address, a message is displayed that
        * no entry was found.
        */
        public function handleSearchResult(event:FindEntryResultEvent):void
        {
            if(event.result != null) {
                // Instantiate a new Entry object with the result that we got from the call.
                var res:Entry = event.result;

                // Populate the result Form fields with the corresponding data.
                txNameOutput.text = res.name;
                txCityOutput.text = res.city;
                txStreetOutput.text = res.street;
                txStateOutput.text = res.state;
                txPostalCodeOutput.text = res.postalCode;

                // Make sure that the no entry found message is not displayed.
                hbNoResults.visible = false;

                // Show the results.
                hbResults.visible = true;

                // Clear the search name field
                tiSearch.text = "";
            }
            else {
                // Display the no entry found message.
                hbNoResults.visible = true;

                // Hide any previous results.
                hbResults.visible = false;
            }
        }
...
```

The following example shows the complete ActionScript-centric address book application:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
creationComplete="initApp()">
<mx:Script>
    <![CDATA[
        import com.adobe.*;
        import mx.rpc.events.FaultEvent;
        import mx.controls.Alert;

        // Declare an instance of the generated web service class.
        public var agenda:AddressBookService;

        /**
        * In the initApp() method we will create a new web service object and
        * add the fault event listener for that web service.
        * Unlike the event listeners, we get only one fault event listener per
        * web service class, as opposed to the event listeners that we get for each
        * operation that returns something.
        */
        public function initApp():void
        {
            // Instantiate the new object.
            agenda = new AddressBookService();

            // Register the fault event listener method.
            agenda.addAddressBookServiceFaultEventListener(handleFaults);
        }

        /**
        * The addEntry() method is used to call the addEntry web service operation.
        * The method gets called when the user clicks the Add Entry button.
        * It builds an Entry object by using the data that the user has provided, calls the
        * web service operation and passes along the strong typed parameter, and then
        * clears the user input fields to provide the visual hint that an action has been
taken.
        */
        public function addEntry():void
        {
            // Instantiate a new Entry object.
            var newEntry:Entry = new Entry();

            // Use the data that the user has provided to populate the Entry object.
            newEntry.name = tiNameInput.text;
            newEntry.city = tiCityInput.text;
            newEntry.street = tiStreetInput.text;
            newEntry.state = tiStateInput.text;
            newEntry.postalCode = tiPostalCodeInput.text;

            // Call the addEntry operation directly from the web service class instance and
            // pass along the required parameter.
            agenda.addEntry(newEntry);

            // Clear the user input fields.
            clearInputFields();
        }

        private function clearInputFields():void {
            tiNameInput.text = "";
            tiCityInput.text = "";
            tiStreetInput.text = "";
            tiStateInput.text = "";
            tiPostalCodeInput.text = "";
        }

        /**
        * The handleFaults() method is a very basic fault event handler method that
        * displays an Alert with the error message.
        */
        public function handleFaults(event:FaultEvent):void
        {
            Alert.show("A fault occured contacting the server. Fault message is: " +
event.fault.faultString);
        }

        /**
        * The searchEntry() method is used to call the findEntry web service operation.
        * It gets called when the user clicks the Find Entry button.
        * It adds the specific findEntry event listener, and if the user has provided a name
        * it makes a call to the web service operation with the provided name.
        */
        public function searchEntry(name:String):void
        {
            // Register the event listener for the findEntry operation.
            agenda.addfindEntryEventListener(handleSearchResult);

            // Call the operation if we have a valid name.
            if(name!= null && name.length > 0)
                agenda.findEntry(name);
        }

        /**
        * The handleSearchResult() method is the result event handler for the
        * findEntry operation.
        * It displays the address of a given person by populating the corresponding
        * form fields with the members of the strongly typed Entry result that
        * you get when the call to the operation succeeds. Or, if we have no record of a
person's address, a message is displayed that
        * no entry was found.the
        */
        public function handleSearchResult(event:FindEntryResultEvent):void
        {
            if(event.result != null) {
                // Instantiate a new Entry object with result that we got from the call.
                var res:Entry = event.result;

                // Populate the result form fields with the corresponding data.
                txNameOutput.text = res.name;
                txCityOutput.text = res.city;
                txStreetOutput.text = res.street;
                txStateOutput.text = res.state;
                txPostalCodeOutput.text = res.postalCode;

                // Make sure that the no entry found message is not displayed.
                hbNoResults.visible = false;

                // Show the results.
                hbResults.visible = true;

                // Clear the search name field.
                tiSearch.text = "";
            }
            else {
                // Display the no entry found message.
                hbNoResults.visible = true;

                // Hide any previous results.
                hbResults.visible = false;
            }
        }
    ]]>
</mx:Script>
    <mx:Panel width="640" height="480" layout="absolute" verticalCenter="0"
horizontalCenter="0" title="AddressBook Client App">
        <mx:TabNavigator x="10" y="10" width="100%" height="100%">
            <mx:Canvas label="Add" width="100%" height="100%" id="tabAdd">
                <mx:HBox x="10" y="10" width="100%" verticalAlign="middle"
borderStyle="solid">
                    <mx:Label text="New entry details:"/>
                    <mx:Form width="293" height="181" borderStyle="solid">
                        <mx:FormItem label="Name">
                            <mx:TextInput id="tiNameInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="City">
                            <mx:TextInput id="tiCityInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Street">
                            <mx:TextInput id="tiStreetInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="State">
                            <mx:TextInput id="tiStateInput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Postal Code">
                            <mx:TextInput id="tiPostalCodeInput"/>
                        </mx:FormItem>
                    </mx:Form>
                    <mx:Button label="Add entry" labelPlacement="top" click="addEntry()"/>
                </mx:HBox>
            </mx:Canvas>
            <mx:Canvas label="Search" width="100%" height="100%" id="tabSearch">
                <mx:HBox x="10" y="10" width="100%">
                    <mx:Label text="Name"/>
                    <mx:TextInput id="tiSearch"/>
                    <mx:Button label="Find entry" click="searchEntry(tiSearch.text);"/>
                </mx:HBox>
                <mx:HBox id="hbNoResults" x="10" y="40" height="22" visible="false">
                    <mx:Text text="'No entry was found!"/>
                </mx:HBox>
                <mx:HBox id="hbResults" x="10" y="70" width="100%" verticalAlign="middle"
borderStyle="solid" visible="false">
                    <mx:Label text="Results:"/>
                    <mx:Form width="293" height="181" borderStyle="solid">
                        <mx:FormItem label="Name">
                            <mx:Text id="txNameOutput"/>
                        </mx:FormItem>
                        <mx:FormItem label="City">
                            <mx:Text id="txCityOutput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Street">
                            <mx:Text id="txStreetOutput"/>
                        </mx:FormItem>
                        <mx:FormItem label="State">
                            <mx:Text id="txStateOutput"/>
                        </mx:FormItem>
                        <mx:FormItem label="Postal Code">
                            <mx:Text id="txPostalCodeOutput"/>
                        </mx:FormItem>
                    </mx:Form>
                </mx:HBox>
            </mx:Canvas>
        </mx:TabNavigator>
    </mx:Panel>
</mx:Application>
```

Working with special types

In addition to the simple classes that the code generator creates for WSDL-defined complex types and request and result event wrappers, special classes are generated for the following types:

- Simple types that enumerate accepted values
- Array types

Generated code for simple types, enumerations, and restrictions

When the code generator introspects a WSDL document and encounters the definition of a simple type that is a restriction of a certain type, and the WSDL provides a list of accepted values, the generated ActionScript model exposes the acceptable values. To do so, the actual value has `[Inspectable]` metadata attached to it. When using MXML syntax, the values read from the WSDL document are proposed as hints, and any other values are rejected by the compiler.

As an example, consider the definition of a simple type that is a string with possible length unit values. Its definition inside the WSDL document is:

```
<s:simpleType name="Lengths">
    <s:restriction base="s:string">
        <s:enumeration value="Angstroms"/>
        <s:enumeration value="Nanometers"/>
        <s:enumeration value="Microinch"/>
        <s:enumeration value="Microns"/>
        <s:enumeration value="Mils"/>
</s:restriction>
</s:simpleType>
```

The code generator translates this into an ActionScript class called Lengths.as with a single member (_Lengths), of type String. Inside the generated class, it also adds the `[Inspectable]` metadata with the values it read from the WSDL document, as the following example shows:

```
public class Lengths
{
    /**
    * Constructor, initializes the type class.
    */
    public function Lengths() {}
        [Inspectable(category="Generated values",
        enumeration="Angstroms,Nanometers,Microinch,Microns,Mils", type="String")]
        public var _Lengths:String;
        public function toString():String {
        return _Lengths.toString();
    }
)
```

Generated code for array types

When an operation must return an array of a specific type and it is defined as a complex type inside the WSDL document, a class is generated for it. This class extends mx.collections.ArrayCollection and implements all of the utility methods that are defined in the base class. To make it easier to see what the underlying type of the array is, all of the methods are slightly modified from their original. For example, `getItemAt(index:int)` becomes `getBaseTypeAt(index:int)`, `getItemIndex(item:Object)` becomes `getBaseTypeIndex(item:BaseType)`, and so on.

Consider the following structure:

```
<s:complexType name="ArrayOfAlert">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="Alert" type="tns:Alert" />
    </s:sequence>
</s:complexType>
```

This structure results in the following generated class:

```
public class ArrayOfAlert extends ArrayCollection {
/**
    * Constructor - initializes the array collection based on a source array
    */
    public function ArrayOfAlert(source:Array = null) {
        super(source);
    }
    public function addAlertAt(item:Alert,index:int):void {
        addItemAt(item,index);
    }
    public function addAlert(item:Alert):void {
        addItem(item);
    }
    public function getAlertAt(index:int):Alert {
        return getItemAt(index) as Alert;
    }
    public function getAlertIndex(item:Alert):int {
        return getItemIndex(item);
    }
    public function setAlertAt(item:Alert,index:int):void {
        setItemAt(item,index);
    }
    public function asIList():IList {
        return this as IList;
    }
    public function asICollectionView():ICollectionView {
        return this as ICollectionView;
    }
}
```

One exception to this naming rule happens when the base type is another complex type named Item. This causes the generation of methods that override the ones implemented in the base class, but without actually specifying it. In this case, the base type is renamed to MyItem, inside method names. When used as actual argument type, the base type name is left unchanged.

Adding a header to an operation call and getting the header from an operation result

Operations defined in a WSDL document can optionally have headers attached to them, either when calling an operation or when returning the result. The code that Flex Builder generates lets the user handle both the request headers and the result headers.

Request headers

If the WSDL document defines request headers for an operation, the generated code has the following extra methods for each operation:

`setOperationName_header()` Sets the operation's request header. You must pass it the exact header object that the operation expects.

`getOperationName_header()` Gets the object that was previously passed as the operation header.

`addOperationName_header(header_arguments:headerType)` Adds a new header item for the operation. This is the preferred way of adding headers becaue it automatically takes care of encoding the header and you need to provide only a strong typed object, the method argument.

You can add a request header to an operation call, as the following example shows:

```
...
public function doCall():void {
    var myHeader:LicenseInfo = new LicenseInfo()
    //Or use the keywordSearchRequest_header property to set the value.
    myService.addkeywordSearchRequest_header(myHeader);
    myService.addkeywordSearchRequestEventListener(handleResults);
    myService.keywordSearchRequest(inputParams);
}
...
```

Result headers

Some operations also return information in headers along with the actual result. You can retrieve the array of headers by accessing the `headers` property on the typed result event, inside the operation's result event handler, as the following example shows:

```
...
public function doCall():void {
    myService.addkeywordSearchRequestEventListener(handleResults);
    myService.keywordSearchRequest(inputParams);
}
public function handleResult(event:KeywordSearchRequestResultEvent):void {
    trace(event.result.TotalResults); //do something with the result
    trace(event.headers); //do something with the result headers
}
```

Handling the result of an operation call

After calling a web service operation, you must retrieve the result and manipulate it. The result of a web service call can be either a SOAP-encoded response or a SOAP fault that contains an error.

To facilitate access to the response of a web service operation call, the code generator also creates some typed event classes. Because Flex operates asynchronously, you must use events to be notified of the result or fault. You can do this in any of the following ways:

- By using the token returned when the operation call is placed
- By using the utility methods defined on the typed service
- By binding elements via MXML to the service's `lastResult` property

Using a return token

When the call to the operation is made, the user gets an AsyncToken object returned immediately. This can be used to attach event listeners (methods that are executed when a specific event occurs), by using syntax like the following example:

```
...
private function doCall():void {
    var ret:AsyncToken;
    ret = myService.keywordSearchRequest(input);
    ret.addEventListener
        (KeywordSearchRequestResultEvent.KeywordSearchRequest_RESULT,handleResult);
    ret.addEventListener(FaultEvent.FAULT,handleFaults);
}
...
```

In this example, the `handleResult()` and `handleFaults()` methods are user-defined methods. When using this approach, the result handling method has an argument of type ResultEvent, instead of a strongly typed, operation-specific event, as in the following example:

```
public function handleResults(event:ResultEvent):void {
    trace(event.result); //do something with the result
}
```

Using utility methods

Through metadata, the service provides a list of events to which the user can listen. There is a single unified fault event, and one result event for each operation. This is because each operation returns a different type of result, but the fault is the same for all. The service provides the following utility methods for attaching event listeners to an operation call:

- The method to add the fault event listener for the service, as the following example shows:

```
public function addSearchServiceFaultEventListener(listener:Function):void
```

- The method on each operation to add the listener to process the result, as the following example shows:

```
public function addkeywordSearchRequestEventListener(listener:Function):void
```

The types of events that the service exposes are available via code hints and as metadata on the service. Each event metadata is commented and exposes the method that dispatches it, as the following example shows:

```
/**
 * Dispatched when a call to the operation KeywordSearchRequest completes successfully
 * and returns some data.
 * @eventType KeywordSearchRequestResultEvent
 */
[Event(name="KeywordSearchRequest_result", type="KeywordSearchRequestResultEvent")]
```

When using this approach, the result handling method takes the strongly typed, operation-specific event, as the following example shows:

```
public function handleResults(event:KeywordSearchRequestResultEvent):void
{
      trace(event.result);//do something with the result
}
```

Using MXML binding

The generated service class exposes for each operation a specific property, `*operationName*_lastResult`, which is populated each time the result of the operation call is returned. You can bind this property through MXML, and when the operation returns a value, it automatically populates all bound items. The following example shows a binding to `*operationName*_lastResult` in an MXML tag:

```
<mx:Text text={myService.keywordSearchRequest_lastResult} />
```

Automatically generating Flex Ajax Bridge code

You use the Create Ajax Bridge feature to generate JavaScript code and an HTML wrapper file that let you more easily use a Flex application from JavaScript in an HTML page. This feature works in conjunction with the Flex Ajax Bridge JavaScript library, which lets you expose a Flex application to scripting in the web browser. The generated JavaScript code is very 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 "Using the Flex Ajax Bridge" on page 844 in the *Adobe Flex 3 Developer Guide*. The sample Ajax Bridge application referenced in this topic is available in an importable Flex Builder project at http://learn.adobe.com/wiki/display/Flex/Download+Projects.

The Create Ajax Bridge feature generates JavaScript proxy code that is specific to the Flex application APIs that you want to call from JavaScript. You can generate code for any MXML application or ActionScript class in a Flex 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 nonrecursively
- 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.

Generating Ajax Bridge code

1 Right-click a project in the Flex Navigator 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 checkbox 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 may 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 Flex 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.

Managing 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 a Flex application, you have three choices:

- Add a cross-domain policy file to the root of the server that hosts the data service. See "Using cross-domain policy files" on page 208.
- 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. See "Setting up Flex Builder to use a proxy for accessing remote data" on page 208.

Using 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—a situation that you want to avoid.

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 the following tech note: http://www.adobe.com/go/tn_14213.

Setting 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. LiveCycle Data Services ES provides a complete proxy management system for Flex applications. 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 this, you need to run a web server and scripting language on your local development computer.

For more information about creating your own proxy, see the following tech note: http://www.adobe.com/go/tn_16520.

Setting up Flex 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 Flex Builder, you can modify both the project build settings and the launch configuration to manage the use of a proxy.

If you use Flex 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.

Modifying the project build path

1 In the Flex Navigator, 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, you must modify the project's launch configuration.

Modifying the launch configuration

1 With the project's main application file open in Flex Builder, right-click in the editor and select Run As > Open Run Dialog. 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 15: Flex Builder User Interface Reference

In the Adobe® Flex® Builder™ IDE, you typically access Flex Builder project properties, work in various Flex Builder views, and create resources to include in your Flex Builder project.

Topics

Setting project properties

Project text encoding properties

Use the Info properties page to specify Flex, ActionScript, and Flex Library project text encoding properties.

Specify text encoding properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > Resource.

3 Specify the following properties:

Text File Encoding Sets the default text file encoding. You can select the file encoding of the application container (UTF-8) or other text encoding formats such as ASCII and UTF-16.

New Text File Line Delimiter Sets the text file line delimiter, which can be inherited from the application container or by the selected type of operating system.

4 Click OK.

Project compiler properties

Use the Compiler properties page to modify how Flex, ActionScript, and Flex Library projects are built.

Set project compiler properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > (Flex, ActionScript, or Flex Library) Compiler.

3 You can set the following compiler properties:

Copy Non-Embedded Files To Output Folder Copies all project assets (images for example) that are part of but not embedded into the SWF file into the output folder. For example, an image file that is embedded into the SWF file (`<image source="@Embed('filename')"/>`) is not copied to the output folder; it is compiled into the SWF file.

Generate Accessible SWF File Enables accessibility features when compiling the application. For more information about creating accessible applications, see "Creating Accessible Applications" on page 916 in the *Adobe Flex 3 Developer Guide*.

Enable Strict Type Checking (-strict) Specifies that projects are compiled in strict mode. This option enforces typing and reports run-time verifier errors when projects are compiled. All errors appear in the Problems view. For more information about strict typing errors, see "Viewing errors and warnings" on page 131 in *Building and Deploying Adobe Flex 3 Applications*.

Enable Warnings (-warnings) Specifies that ActionScript projects are compiled in Warnings mode, which generates migration warnings. These include ActionScript syntax anomalies, obsolete or removed ActionScript 2.0 APIs, and differences in the behavior of ActionScript 2.0 and 3.0 APIs. Warnings appear in the Problems view. For more information about debugging your applications, see "Running and Debugging Applications" on page 136 in *Using Adobe Flex Builder 3*. For more information about warnings, see "Viewing errors and warnings" on page 131 in *Building and Deploying Adobe Flex 3 Applications*.

Namespace URL and Manifest File (Flex Library projects only) You can create a custom namespace for the components contained within a library project by specifying a namespace URL and a manifest file. For more information, see "Using compc, the component compiler" on page 121 in *Building and Deploying Adobe Flex 3 Applications*.

Additional Compiler Arguments Allows you to add optional compiler arguments. For more information about using compiler arguments, see "Using the Flex Compilers" on page 90 in *Building and Deploying Adobe Flex 3 Applications*.

Run-time web browser properties

For Flex and ActionScript projects, you can also set the following run-time web browser properties:

Generate HTML Wrapper File Automatically generates the HTML wrapper file when the project is compiled and places it in a folder called html-template within your project.

Require Flash Version Checks to see if the user has the specified version of Flash Player installed and if not prompts the user to install it.

Use Express Install Prompts the user to install the correct version of Flash Player using Express Install, which is a streamlined (quicker and easier) version of the Flash Player installer.

Enable Integration With Browser Navigation Enables your application to use the web browser's back button.

Modify the compiler properties as needed, and click OK.

Setting project application file properties

Use the Applications properties page to specify the Flex and ActionScript project files that should be compiled as applications. Flex

Set applications properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > Flex Applications (or ActionScript Applications).

3 You can set the following compiler properties:

Select the Runnable Application Files Selects the list of project files that have been set as runnable files.

Add Allows you to select project files set as runnable application files (compiled as separate SWF files).

Remove Removes the selected file from the list of runnable application files.

Set as Default Sets the selected application files as the default (main) application file in your project.

4 Modify the list of application files as needed, and click OK.

Flex and ActionScript project build path properties

Use the Build Path properties page to specify Flex and ActionScript project build path properties. For information about setting build path properties for Flex Library projects, see "Flex library project build path properties" on page 212.

Specify build path properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Macintosh) to display the context menu and select Properties > (Flex or Action-ScriptActionScript) Build Path.

3 Specify the following properties:

Source path Specifies the link to external resources to your application. For example, if you have a folder of shared ActionScript classes, you can link to that folder by adding it to the source path. You can also edit a folder's path, remove the folder from the source path, and arrange the order of the folders using the Up and Down buttons.

Library path Specifies the link to external resource libraries (SWC files). By default, the library path of new ActionScript projects contains the playerglobal.swc and utilities.swc files. You can link library projects, SWC folders, and compiled SWCs to the library path. You can also edit the path entry, remove the entry from the library path, and arrange the order of the folders using the Up and Down buttons.

Main source folder Specifies by default the root of your project. You can, however, choose a different folder within the project. You can browse the project folder structure and create a folder for the source if needed. (Optional)

Output folder Specifies the location of the compiled application files. By default, this is named bin but you can change this if you like. For more information, see "Setting up a project output folder" on page 124.

Output folder URL Specifies the server location of the compiled application files. (Optional)

4 Click OK.

Flex library project build path properties

Use the Flex Library Build Path properties page to specify Flex Library project build path properties.

Specify build path properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Macintosh) to display the context menu and select Properties > Flex Library Build Path.

3 Specify the following properties:

Classes Tab lists the classes that can be included in the SWC file. The classes that are listed are added directly to the project or linked to it using a folder in the source path. For more information, see "Selecting library project elements to include in the SWC file" on page 49.

Resources Tab lists all the resources in the project. You can select the resources to include in the SWC file.

Source Path Tab lets you add and manage folders in the source path. Components contained in folders on the source path can be selected as component classes to include in the SWC file.

Library Path Tab lets you add other SWC files to the project. You can add other projects to the library path and modify library settings to use your library project as an RSL, and so on. For more information, see "Using SWC files in your projects" on page 50.

You can also set the following properties:

Main Source Folder Specifies the location of the project source files. By default, the project source files are located in the root of the project. You can specify a different location (folder) in the project.

Output Folder Specifies the location of the compiled SWC file. By default, this is named bin but you can change this if you like. For more information, see "Setting up a project output folder" on page 124.

4 Click OK.

Flex server properties

Use the Flex Server properties page to specify server properties for Flex projects that use either the ColdFusion Flash Remoting Service or LiveCycle Data Services ES.

Specify server properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > Flex Server.

3 Specify the following properties:

Root folder Specifies the root folder of the Flex web application (for example, C:\fds2\jrun4\servers\default\flex).

Root URL Specifies the URL of the Flex web application root (for example, http://localhost:8700/flex/).

Context root Specifies the context root of the Flex server. Typically, this is same as the last portion of the server URL (for example, flex).

4 Click OK.

Project builder properties

Use the Builder properties page to configure the builders that will be used to compile your projects. For more information, see "Customizing builds with Apache Ant" on page 131.

Access project builder properties

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > Builders.

Project references

Use the Project References properties page to create references to other projects in the workspace. For more information, see "Alternatives to using project references" on page 44.

1 In the Flex Navigator view, select a project.

2 Right-click (Control-click on Mac OS) to display the context menu and select Properties > Project References.

Using Flex Builder views

This section provides information about the views provided by Flex Builder:

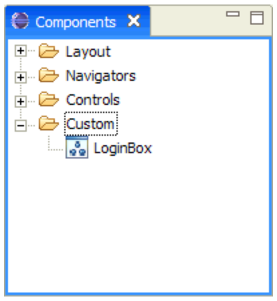
Components view

The Components view lets you rapidly insert Flex components in MXML files in the MXML editor’s Design mode.

Nonvisible components, such as effects, formatters, and validators, are not listed in the Components view. You must insert these components in the code.

Custom category

The Components view has a category of components called Custom:



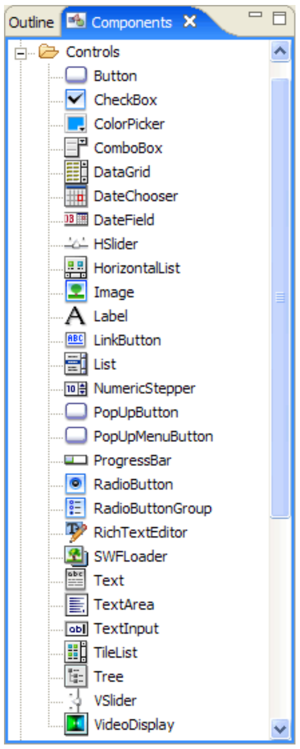
You must open a file in Flex Builder to view the components in this category. The category lists the custom components available to the project that contains the currently active document. If there is no active document, or the active document is not in a project, the Custom category is empty.

The Components view only lists visible custom components (components that inherit from the UIComponent class).

The Custom category lists all the custom components that you saved in the current project or in the source path of the current project. For example, if you create a component file called LoginBox.mxml and save it in your project, the LoginBox component appears in the Custom category of the Components view. Like any component, you can insert these components in your layout by dragging them from the Components view.

Controls category

The Components view has a category of components called Controls:



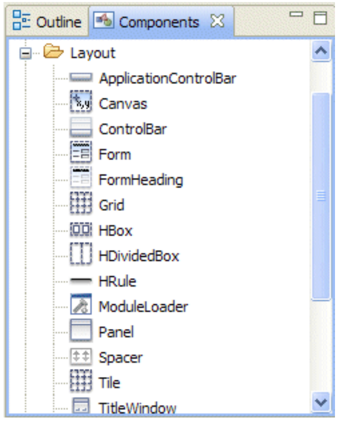
The following table briefly describes the components in this category:

Component	Description
Button	Control that displays a variable-size button that can include a label, an icon image, or both.
Checkbox	Control that shows whether a particular Boolean value is true (checked) or false (unchecked).
ColorPicker	Control that allows the user to select a color from a palette.
ComboBox	Data provider control that displays a drop-down list attached to a text field that contains a set of values.
DataGrid	Data provider control that displays data in a tabular format.
DateChooser	Control that displays a full month of days to let you select a date.
DateField	Control that displays the date with a calendar icon on its right side. When a user clicks anywhere inside the control, a DateChooser control pops up and displays a month of dates.
HorizontalList	Data provider control that displays a horizontal list of items.
HSlider	Control that lets users select a value by moving a slider horizontally.
Image	Control that imports a JPEG, PNG, GIF, or SVG image or SWF file.
Label	Control that displays a noneditable single-line field label.
LinkButton	Control that displays a simple hypertext link.
List	Data provider control that displays a scrollable array of choices.
NumericStepper	Control that displays a dual button control you can use to increase or decrease the value of the underlying variable.

Component	Description
PopUpButton	Control that consists of two horizontal buttons: a main button, and a smaller button called the pop-up button, which only has an icon. The main button is a Button control.
PopUpMenuButton	Control that consists of two horizontal buttons: a main button, and a smaller button that displays a menu when clicked.
ProgressBar	Control that provides visual feedback of how much time remains in the current operation.
RadioButton	Control that can be selected or deselected. In a RadioButtonGroup, only one can be selected at a time.
RadioButton Group	Control that forces the RadioButton controls associated with it to be mutually exclusive.
RichTextEditor	Control that lets the user edit text and apply simple styles such as bold and italic.
SWFLoader	Control that displays the contents of a specified SWF file.
Text	Control that displays a noneditable multiline text field.
TextArea	Control that displays an editable text field for user input that can accept more than a single line of input.
TextInput	Control that displays an editable text field for a single line of user input. Can contain alphanumeric data, but input will be interpreted as a String data type.
TileList	Data provider control that displays a tiled list of items. The items are tiled in vertical columns or horizontal rows.
Tree	Data provider control that lets a user view hierarchical data arranged as an expandable tree.
VideoDisplay	Control that lets you play an FLV file in a Flex application. It supports streaming video from the Flash Media Server, FLV files, and from a Camera object.
VSlider	Control that lets users select a value by moving a slider vertically.

Layout category

The Components view has a category of components called Layout:



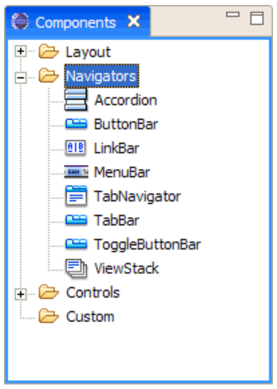
The following table briefly describes the components in this category:

Component	Description
ApplicationControlBar	Control bar that can appear docked across the top of the application, or undocked within an application.
Canvas	Container that lets you explicitly position and size its children. Automatically inserted when creating an MXML Application file.
ControlBar	Container that holds components shared by the other children in a Panel container.
Form	Container that arranges its children in a standard form format. Should only be used in a Panel container.

Component	Description
FormHeading	Control that specifies an optional label for a group of FormItem containers. Only valid in a Form.
Grid	Container that arranges children as rows and columns of cells, similar to an HTML table.
HBox	Container that lays out its children horizontally in a single row.
HDividedBox	Container that lays out its children horizontally like a HBox container, except that it inserts an adjustable divider between each child.
HRule	Control that displays a single horizontal rule. Typically used to create dividing lines within a container.
Panel	Container that displays a title bar, caption, border, and its children. Used as a basic building block in structuring the overall application layout.
Spacer	Invisible control that lets you control the spacing between components in a container.
Tile	Container that arranges its children in multiple rows or columns.
TitleWindow	Container that displays a modal window that contains a title bar, caption, border, close button, and its children. The user can move and resize it.
VBox	Container that lays out its children vertically in a single column.
VDividedBox	Container that lays out its children vertically like a VBox container, except that it inserts an adjustable divider between each child.
VRule	Control that displays a single vertical rule. Typically used to create dividing lines within a container.

Navigators category

The Components view has a category of components called Navigators:



The following table briefly describes the components in this category:

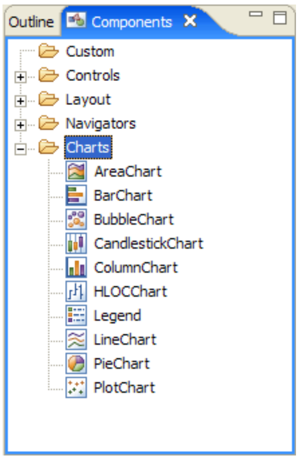
Component	Description
Accordion	Navigator container that organizes information in a series of child panels, where one panel is active at any time.
ButtonBar	Navigator container that defines a row of Button controls designating a series of link destinations. Often used with a ViewStack container.
LinkBar	Navigator container that defines a row of Link controls designating a series of link destinations. Often used with a ViewStack container.
MenuBar	Container that displays a horizontal menu bar that contains one or more submenus of Menu controls.
TabBar	Navigator container that defines a horizontal row of tabs. Often used with a ViewStack container.
TabNavigator	Navigator container that displays a container with tabs to let users switch between different content areas.

Component	Description
ToggleButtonBar	Like ButtonBar, except the button stays pressed.
ViewStack	Navigator container that defines a stack of panels that displays a single panel at a time.

Charts category

The Flex charts are available in Adobe Flex Builder Professional. A trial version of the charts is included in the Flex Builder Standard.

The Components view has a category of components called Charts:



The following table briefly describes the components in this category:

Component	Description
AreaChart	Area charts represent data as an area bounded by a line connecting the data values.
BarChart	Bar charts represent data as a series of horizontal bars, the length of each determined by data values.
BubbleChart	Bubble charts represents data with three values for each data point: one for the *x* axis, one for the *y* axis, and another for the size of the bubble.
CandlestickChart	Candlestick charts represent financial data as a series of candlesticks representing the high, low, opening, and closing values of a data series.
ColumnChart	Column charts represent data as a series of vertical columns, the size of each determined by data values.
HighLowOpenCloseChart	HLOC charts represent financial data as a series of lines representing the high, low, opening, and closing values of a data series.
Legend	Legend controls match the fill patterns on your chart to labels that describe the data series shown with those fills.
LineChart	Line charts represent data as a series of points, in Cartesian coordinates (*x* and *y* axes forming a plane), connected by a continuous line.
PieChart	Pie charts represent data as slices of a pie, the data determining the size of each slice.
PlotChart	Plot charts represent data as Cartesian coordinates, with data points for both the *x* and *y* axes. You provide shape for the data points using a display renderer (the circle renderer, for example).

Flex Properties view

The Flex Properties view lets you rapidly set the properties of visual components in the MXML editor's Design mode. The view is not available in Source mode.

You cannot use the Flex Properties view to set the properties of nonvisible components such as effects, formatters, and validators. You must set the properties of these components in the code.

The toolbar of the Flex Properties contains the following buttons:

Standard view Displays the most commonly used properties in a form layout.

Category view Displays the properties in a categorized list.

Alphabetical view Displays the properties in an alphabetical list.

To use a default property value, or use the value set by a CSS rule, leave the field blank.

Flex Navigator view

The Flex Navigator view allows you to manage the projects and resources contained within the workspace.

Outline view

You use the Outline view to more easily inspect and navigate the structure of your MXML and ActionScript documents.

States view

The States view shows the states defined for the current application in Flex Builder. The list of states appears in a tree structure, showing which states are based on which other states.

Your view contains the following options:

The tree structure Lets you select a state to display and edit in the MXML editor's Design mode. When you select a state in the tree, Flex Builder displays the state's layout in Design mode. You can modify the layout by using the design tools in Flex Builder.

New State Lets you create a state. For more information, see "Setting running and debugging preferences" on page 223.

Edit State Properties Lets you edit some basic properties of the selected state. You can also double-click the state in the tree to edit the properties. For more information, see "Edit State Properties dialog box" on page 219.

Delete State Lets you remove the selected state. If you attempt to delete a state that other states are based on, a dialog box appears to warn you that the other states will be deleted too. You can cancel the operation or confirm that you want to delete the state.

Selecting Edit > Undo restores the state that you deleted.

New State dialog box

The New State dialog box lets you define a new state. It contains the following options:

Name Lets you specify the name of the new state.

Based On Lets you select the state on which to base the new state. The pop-up menu lists the states defined in the current document.

Set As Start State Shows this state when the application starts.

Edit State Properties dialog box

The Edit State Properties dialog box lets you modify the properties of an existing state, such as its name and whether it appears when the application starts. It contains the following options:

Name Lets you specify a new name for the state. You cannot rename the base state.

Set As Start State Shows or hides this state when the application starts.

Creating project resources

This section provides information about dialog boxes used to create project resources.

Setting New ActionScript Class dialog box options

The New ActionScript Class dialog box lets you rapidly create an ActionScript class.

1 (Optional) Specify a package for your class.

   A *package* is a named collection of classes. If you don't specify a package, the class will be declared in the default package (`"package { ... }"`).

   If you specify a package folder that does not exist, the wizard will create it.

2 Name the class file.

   The filename defines the class name. For example, if you name the file LoginBox.as, the class is named LoginBox.

3 Select one of the following modifiers:

   Public Specifies that the class is available to any caller. Classes are internal by default, which means that they are visible only within the current package. To make a class visible to all callers, you must use the `public` attribute.

   Internal Specifies that the class is available to any caller within the same package.

   Dynamic Specifies that the class is a dynamic class that can be altered at run time by adding or changing properties and methods.

   Final Specifies that the class cannot be extended.

4 In the Superclass text box, specify the class from which you want to derive your new class.

   Container classes are commonly used as the superclasses of derived classes used for layout. For example, if you select the HBox class as the superclass, Flex Builder inserts the following code in your class:

```
import mx.containers.HBox;
public class LoginBox extends HBox {

}
```

5 Add any interface that contains constants and methods that you want to use in your new class.

   An *interface* is a collection of constants and methods that different classes can share. For more information on the Flex interfaces, see *Adobe Flex Language Reference* in Help.

6 Select any of the following code generation options:

   Generate Constructor from Superclass Generates a constructor with a `super()` call. If the superclass constructor takes arguments, the arguments are included in the generated constructor and passed up in the `super()` call.

   Generate Functions Inherited from Interfaces Generates function stubs for each interface method. The stubs include a `return` statement that returns null (or 0 or false for primitive types) so that the stubs will compile.

   Generate Comments Inserts a "//TODO: implement function" in the bodies of any generated functions or constructors.

7 Click Finish.

   Flex 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, Flex 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 67.

8 Write the definition of your ActionScript class.

For more information, see "Creating Simple Visual Components in ActionScript" in *Creating and Extending Adobe Flex 3 Components*.

Setting New ActionScript Interface dialog box options

The New ActionScript Interface dialog box lets you rapidly create an ActionScript interface.

1 (Optional) Specify a package for your interface.

A *package* is a named collection of classes and interfaces. If you don't specify a package, the interface will be declared in the default package (`"package { ... }"`).

If you specify a package folder that does not exist, the wizard will create it.

2 Name the interface.

3 Select one of the following modifiers:

Public Specifies that the interface is available to any caller.

Internal Specifies that the interface is available to any caller within the same package.

4 In the Extended Interfaces area, add any interface that contains constants and methods that you want to use in your new ActionScript interface.

Your new interface will be extended with the other interfaces. For more information on the Flex interfaces, see *Adobe Flex Language Reference* in Help.

5 Add any constants or methods to your ActionScript interface that you want different classes to share.

Setting the New MXML Component dialog box options

The New MXML Component dialog box lets you rapidly create an MXML component.

1 Specify the parent folder for your custom component file.

Save the file in the current project or in the source path of the current project if you want the component to appear in the Components view.

2 Specify the filename of the component.

The filename defines the component name. For example, if you name the file LoginForm.mxml, the component is named LoginForm.

3 Select the base component of your custom component.

Custom components derive from existing components. Containers are commonly used as the base components of custom components.

4 Click Finish.

Flex 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, Flex 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 67.

5 Develop your custom component.

For more information, see "Simple MXML Components" on page 45 in *Creating and Extending Adobe Flex 3 Components*.

Create Chart dialog box

The Create Chart dialog box lets you add charting components to your application rapidly. The dialog box has the following options:

ID Specifies the ID for the new chart.

Series Elements Lists the series of data in your chart. For more information, see "About charting" on page 2 in *Adobe Flex 3 Data Visualization Developer Guide*.

Add Lets you add more than one data series to your chart. Enter the new series name in the dialog box that appears after you click the button.

Remove Lets you remove a data series selected in the list.

Up and Down Let you change the order of the data series.

Include Legend Lets you add a Legend control to your chart that displays the label for each data series in the chart and a key showing the chart element for the series.

Setting editor preferences

This dialog box allows you to specify preferences that apply to the MXML, ActionScript, and CSS editors.

1 Open the Preferences dialog and select Flex > Editors:

   Indent Using (Tabs or Spaces) Specifies that closing braces are indented so that your code is properly formatted and easier to read.

   Auto Indent Braces When Typing Specifies that when you select a code expression's opening brace, the closing brace is highlighted in the code editor.

   Enable Code Folding Enables Content Assist to automatically provide code hints while entering code.

2 Click OK.

Setting MXML editor preferences

This dialog box allows you to specify MXML editor preferences.

1 Open the Preferences dialog and select Flex > Editors:

   Automatically Show Design-related Views Specifies that when you switch into Design mode all of the design mode views are displayed.

   Enable Snapping Specifies that the components you place into the editor in Design mode snap to guidelines.

   Render Skins When Opening a File Specifies that when you load a MXML file into the MXML editor the skins will be rendered and displayed.

   Collapse Data Binding Expressions Collapses data binding expressions in the code editor. You can expand these expressions using code folding.

2 Click OK.

Setting ActionScript editor preferences

This dialog box allows you to specify ActionScript editor preferences.

1 Open the Preferences dialog and select Flex > Editors > ActionScript Editor:

   Keep Close Braces Correctly Indented Specifies that closing braces are indented so that your code is properly formatted and easier to read.

Highlight Matching Braces Specifies that when you select a code expression's opening brace, the closing brace is highlighted in the code editor.

Wrap Strings Automatically Wraps lines of code wrap to the size of the editor window.

2 Click OK.

Setting MXML Code Assist preferences

This dialog box allows you to specify MXML formatting and Code Assist preferences.

***Note:** The Code Assist feature is also referred to as Content Assist and these terms are used interchangeably.*

1 From the main menu, select Window > Preferences.

The Preferences dialog box appears.

2 Select the Flex > Editors > MXML Editor > Code Assist preferences page and set the following preferences:

Insert Closing Quote When Completing Attributes Automatically adds a closing quote (") when entering attribute values.

Insert Close Tags When Completing Tags Automatically adds a closing tag (/>) when entering MXML tags.

Insert New Line and Indent When Completing Tags Adds and properly indents a new line of code when a tag is completed.

Insert Default Namespace Attribute for mx:XML, mx:XMLList, and mx:request Automatically adds a namespace attribute to mx:XML, mx:XMLList, and mx:request tags.

Insert CDATA for mx:Script Inserts a CDATA construct when you enter an `<mx:Script>` tag.

Insert CDATA for Child Event Attributes Inserts a CDATA construct for child event attributes.

Insert CDATA for mx:htmlText Inserts a CDATA construct when you enter an `<mx:htmlText>` tag.

Enable Auto-activation Enables Content Assist to automatically provide code hints while entering code.

Set Auto-activation Delay Specifies in one hundreths of a second the delay before code hinting appears.

Set Auto-activation Characters Sets the characters that trigger Content Assist.

3 Click OK.

Setting running and debugging preferences

You can specify that the project is built before it is run as an application and also set other launch-related preferences in Flex Builder.

1 Open the Preferences dialog and select Run/Debug > Launching:

Save Required Dirty Editors Before Launching Specifies that editors containing unsaved data will be saved (the options are Always, Never, or Prompt). The term *dirty editor* refers to editors that contain code that has not been saved.

Wait for Ongoing Build to Complete Before Launching Allows you to set the wait before launching preference to Always, Never, or Prompt.

Launch in Debug Mode When Workspace Contains Breakpoints Select Always, Never, or Prompt.

Continue Launch if Project Contains Errors Select Always or Prompt.

Build (If Required) Before Launching Is selected by default. If needed, builds the project specified in the launch configuration before launching the application.

Remove Terminated Launches When a New Launch is Created Clears all of the terminated launches in the Debug view.

Size of Recently Launched Applications List Specifies the number of recently launched applications that are displayed in the Run As menu. The default is 10.

2 Click OK.

Setting Console view preferences

You can modify preferences that affect how text in the Console view is displayed.

1 Open the Preferences dialog and select Run/Debug > Console:

Fixed Width Console Sets the Console view to a fixed width based on the maximum character width you enter.

Limit Console Output Sets the Console view to a buffer size equal to the maximum buffer size (in characters) that you enter.

Displayed Tab Width Sets the character width of tabs in the Console view.

Show When Program Writes to Standard Out Shows the Console view (if hidden or inactive) when it is invoked by an application that creates output to the view.

Show When Program Writes to Standard Error Shows the Console view (if hidden or inactive) when an error is encountered while an application is run or debugged.

Standard Out Text Color Specifies the color of the standard output text.

Standard Error Text Color Specifies the color of the error output text.

Standard In Text Color Specifies the color of the text that you enter into the Console view.

Standard Background Color Specifies the background color of the output text.

2 Click OK.

Setting Run/Debug preferences

You can specify general settings for running and debugging your applications in Flex Builder.

1 Open the Preferences dialog and select Run/Debug:

Reuse Editor When Displaying Source Code Displays all source files in the same editor, rather than opening a new editor for each source file.

Activate the Workbench When a Breakpoint is Hit Activates the workbench title bar so that it flashes when a breakpoint is hit, alerting you that the debugger is active.

Activate the Debug View When a Breakpoint is Hit Displays the Flex Debugging perspective when a breakpoint is hit.

Skip Breakpoints During a ‘Run to Line’ Operation Is not supported in Flex Builder.

Changed Value Color Specifies the text color of the changed value of a variable.

Changed Value Background Color Specifies the background color for the text of the changed value of a variable.

Memory Unbuffered Color Is not supported in Flex Builder.

Memory Buffered Color Is not supported in Flex Builder.

2 Click OK.

Setting Flex debugging preferences

You can specify debugging settings that are specific to the Flex Builder debugger.

1 Open the Preferences dialog and select Flex > Debug:

Warn When Trying to Launch Multiple Debugging Sessions Is selected by default. Firefox only supports one debugging session at a time; therefore, if you attempt to begin another debugging session, the current session will be terminated. If you deselect this preference, you will never receive a warning and the existing debugging session will be automatically terminated.

Automatically Invoke Getter Functions Is selected by default. When debugging, if a getter function is displayed in the Variables view, the function is invoked. If you turn this preference off, you can manually invoke a getter function in the Variables view by selecting it, right-clicking (Control-clicking on Macintosh) to display the context menu and selecting Invoke Getter.

Changing this preference does not affect the current debugging session, only subsequent debugging sessions.

2 Click OK.

Chapter 16: Creating Custom MXML Components

An application in Adobe Flex typically consists of an MXML application file (a file with an `<mx:Application>` parent tag), one or more standard Flex components, and 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 Flex Builder to build custom MXML and ActionScript components visually and then insert them into your applications. This topic describes how to build custom MXML components visually. For more information on building ActionScript components, see "Creating an ActionScript class" on page 46.

You can also build an MXML component directly using code. For more information, see "Simple MXML Components" on page 63 in *Creating and Extending Adobe Flex 3 Components*.

Topics

About custom components

You might want to create custom components for a number of reasons. For example, you might simply want to add some functionality to an existing component, or you might want to build a reusable component, like a search box or the display of an item in a data grid. You might want to write a completely new kind of component that isn't available in the Flex framework.

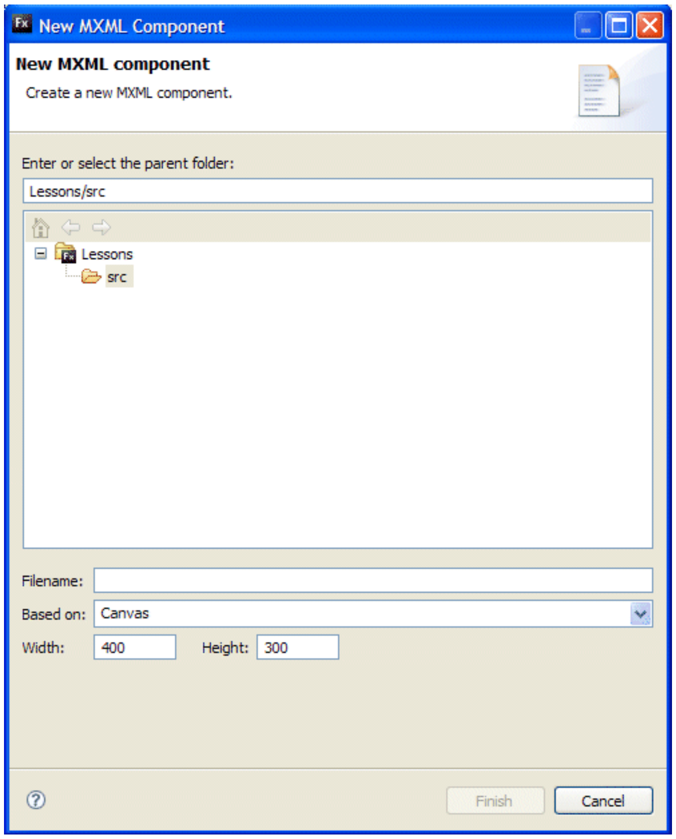
If your component is composed mostly of existing components, it is convenient to define it using MXML. However, if it is a completely new kind of component, you should define it as an ActionScript component. For more information, see "Creating an ActionScript class" on page 46.

Creating MXML components visually

You use Flex Builder to create custom MXML components.

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 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.

5 (Optional) If you base the component on a Panel or TitleWindow component, a Layout pop-up menu appears letting you select how the children of the component will be laid out—absolutely, horizontally, or vertically.

6 (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 will look 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 "Designing components visually" on page 229.

7 Click Finish.

Flex 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, Flex 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 67.

***Note:** The Components view lists only visible custom components (components that inherit from the UIComponent class). For more information, see Adobe Flex Language Reference.*

8 Create your custom component.

For more information, see "Simple MXML Components" on page 63 in *Creating and Extending Adobe Flex 3 Components*.

Designing 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 "Working with components visually" on page 69.

The size of a custom component displayed in Design mode is determined by the following rules:

- If your component has a fixed width and height, Flex 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%, Flex Builder renders it as a percentage of the selected size in the Design Area menu.

Editing and distributing custom MXML components

Flex 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 right-click a custom component to display the context menu and select Open Custom Component.

3 Edit the custom component.

Distributing custom components

Distribute custom components by creating library projects. For more information, see "About library projects" on page 47.