Flex 3 Beta 1

Adobe® Flex™ Builder™ 3

Using Flex Builder

Adobe®

Contents

Introduction

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, and the Adobe® Flex™ Charting 2 components.

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

This topic contains the following sections:

What you can do with Flex Builder

Using Flex Builder, you can develop Flex 2 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 56.
- Create ActionScript projects that use the Flash API. For more information, see "About ActionScript projects" on page 74.
- Write and edit your application source code using editors that provide features such as code hinting, streamlined code navigation, and automatic syntax error checking. For more information, see Chapter 9, "Code Editing in Flex Builder," on page 165.

- Use the MXML editor in Design mode to simplify using view states and transitions, to design using absolute layout options, to drag components on to the design canvas and then reposition and resize them as needed, and so on. For more information, see Chapter 5, "Building a Flex User Interface," on page 105.
- 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 Chapter 8, "Creating Custom MXML Components," on page 157.
- 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 60 and "Managing project resources" on page 65.
- Build your applications using predefined builders or create custom builders using Apache Ant. For more information, see Chapter 10, "Building Projects," on page 189.
- 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 207 and "Managing launch configurations" on page 209.
- Test and debug your applications using the integrated debugging tools in Flex Builder. For more information, see Chapter 11, "Running and Debugging Applications," on page 205.
- Publish your application source code so it can be viewed by users and other developers. For more information, see "Publishing application source code" on page 201.
- 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 78.
- Customize the IDE. For example, you can arrange the interface to include your favorite tools in the specific layout. For more information, see Chapter 4, "Navigating and Customizing the Flex Builder Workbench," on page 85.

Flex Builder versions

Flex Builder is available in two versions: a standard version and a version that includes the Flex charting components.

Flex Builder The standard version of Flex Builder provides a full-featured IDE that allows you to create Flex and ActionScript applications using the Flex framework and Flash API. Flex Builder includes MXML, ActionScript, and CSS editors, as well as debugging tools.

For an introduction to Flex Builder, see Chapter 2, "Flex Builder Workbench Basics," on page 25.

Flex Builder **with Charting** This version of Flex Builder includes a rich library of interactive charts and graphs that enables rich data dashboards and interactive data analysis.

For more information about the Flex product line, see "About the Flex product line" in *Getting Started with Flex*. For an introduction to the charting components available in this version, see Chapter 5, "Using Flex Charting Components" in *Getting Started with Flex*.

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.

Standalone 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 standalone 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 standalone 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 standalone 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.

System requirements

Before you install Flex Builder, make sure your computer meets the system requirements. For more information, see the Adobe website at www.adobe.com/go/sysreqs.

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 licences. 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.

To 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 9) 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.

Typographical conventions

The following typographical conventions are used in this book:

Italic font indicates a value that should be replaced in an example, such as a folder name in a folder path.

Bold font indicates a verbatim entry.

`Code font` indicates code snippets or MXML or ActionScript elements in the text.

`Code font italic` indicates a value that should be replaced in a code snippet.

`Code bold font` indicates a verbatim entry in a code snippet.

PART 1

Getting Started with Flex Builder

1

This part contains a guide to Flex and Flex Builder documentation and an introduction to the basics of the Flex Builder workbench.

The following chapters are included:

CHAPTER 1

Learning Flex Builder

1

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.

This topic contains the following sections:

How to get started 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.

Flex Builder 1.5 users

- Start by reading "How Flex Works", which gives you a quick overview of essential Flex concepts.
- Read *Getting Started with Flex*. This guide introduces you to Flex 2 and tell you what's new and different in this version.
- To set up your first project (a new concept for Flex 1.5 users) and use the new IDE, see Chapter 8, "Create Your First Application" and Chapter 16, "Use the Code Editor" in *Getting Started with Flex*.
- 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 Chapter 2, "Flex Builder Workbench Basics," on page 25 and especially "Flex Builder basics" on page 25.
- After you're familiar with the Flex Builder workbench, see *Migrating Applications to Flex 2*, which describes how to migrate Flex 1.5 applications to Flex 2.

Web application designers

- Start by reading "How Flex Works", which gives you a quick overview of essential Flex concepts.
- Read *Getting Started with Flex*. This guide introduces you to Flex 2. Most importantly, it describes the basic elements of a Flex application and explains how you define them in MXML and use ActionScript 3.0 to add programmatic interactivity. It also introduces the standard Flex components, with which you build rich Internet applications, and Flex 2 features, such as *view states* and *transitions,* that are essential to creating rich user interface experiences.
- 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 "The Flex Development perspective in Design mode" on page 34 and Chapter 5, "Building a Flex User Interface," on page 105.
- For lessons that teach the basics of designing Flex applications, see Chapter 10, "Create a Constraint-based Layout" and Chapter 14, "Use View States and Transitions" in *Getting Started with Flex*.

Experienced web application developers

- Start by reading "How Flex Works", which gives you a quick overview of essential Flex concepts.
- Read *Getting Started with Flex* so you can compare what you already know about building applications in HTML, JSP, ASP, PHP, and so on, to how Flex applications are built using the Flex framework, MXML, and ActionScript 3.0.
- For all the details of the Flex framework, including code samples, see *Flex Developer's Guide*.
- For in-depth information about the steps involved in building and deploying a Flex application, see *Building and Deploying Flex 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 "What you can do with Flex Builder" on page 7, "Understanding Flex Builder projects" on page 50, "Understanding code editing in Flex Builder" on page 166, "Understanding how projects are built" on page 190, and "Understanding running and debugging applications in Flex Builder" on page 205.
- For all the API details, see *Adobe Flex Language Reference*. If you use Flex Builder to write code, this guide is conveniently available by pressing F1.

Flash and ActionScript developers

- For an overview of the changes that have been made to ActionScript 3.0, see "ActionScript 2.0 to 3.0" in *Migrating Applications to Flex 2* (available online at: www.adobe.com/go/flex_documentation).
- For all the API details, see *Adobe Flex Language Reference*. If you use Flex Builder to write code, this guide is conveniently available by pressing F1.
- 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 74.
- To understand how to design and lay out applications in Flex Builder, see Chapter 5, "Building a Flex User Interface," on page 105 and the suggested reading in "Web application designers" on page 16.

Enterprise application developers

- For an overview of the enterprise features of the Flex framework, see Chapter 3, "Building a Flex Application" in *Getting Started with Flex* and the suggested reading in the section "Experienced web application developers" above.

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 Chapter 2, "Flex Builder Workbench Basics," on page 25.

Getting the most from the Flex and Flex Builder documentation

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 documents 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.

This section covers the following topics:

- "Accessing the Flex Builder documentation" on page 17
- "Accessing additional online Flex Builder resources" on page 19

Accessing the Flex Builder documentation

The following table summarizes the documentation included in the Flex Builder help system.

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

Title	Description/Audience	Where to Find It
Getting Started with Flex	Basic introduction with tutorials to Flex concepts and the development tool. Intended for beginning users, as well as intermediate and advanced users who want an overview of the Flex product family.	• View in Flex Builder: Select Help › Help Contents, and then select Adobe Flex Help › Getting Started with Flex 2. • View online: www.adobe.com/go/flex2_livedocs • Get the PDF: www.adobe.com/go/flex_documentation
Using Flex Builder	Comprehensive information about all Flex Builder features. Intended for all Flex Builder users.	• View in Flex Builder: Select Help › Help Contents, and then select Adobe Flex Help › Using Flex Builder. • View online: www.adobe.com/go/flex2_livedocs • Get the PDF: www.adobe.com/go/flex_documentation
Flex Developer's Guide	Describes how to develop rich Internet applications with Flex. Intended for application developers who are learning Flex or want to extend their Flex programming knowledge.	• View in Flex Builder: Select Help › Help Contents, and then select Adobe Flex Help › Flex 2 Developer's Guide. • View online: www.adobe.com/go/flex2_livedocs • Get the PDF: www.adobe.com/go/flex_documentation
Building and Deploying Flex Applications	Describes how to build and deploy Flex applications, including how to design, configure, build, deploy, and secure a Flex application.	• View in Flex Builder: Select Help › Help Contents, and then select Adobe Flex Help › Building and Deploying Flex 2 Applications. • View online: www.adobe.com/go/flex2_livedocs • Get the PDF: www.adobe.com/go/flex_documentation
Creating and Extending Flex Components	Describes how to create custom Flex components in MXML and ActionScript.	• View in Flex Builder: Select Help › Help Contents, and then select Flex Help › Creating and Extending Flex 2 Components. • View online: www.adobe.com/go/flex2_livedocs Get the PDF: www.adobe.com/go/flex_documentation

Title	Description/Audience	Where to Find It
Migrating Applications to Flex 2	Describes how to migrate Flex 1.x applications to Flex 2.	• View online: www.adobe.com/go/flex2_livedocs Get the PDF: www.adobe.com/go/flex_documentation
Programming ActionScript 3.0	Describes how to use ActionScript 3.0. For API information, see the *Adobe Flex Language Reference*.	• View in Flex Builder: Select Help › Help Contents, and then select Flex Help › Programming ActionScript 3.0 • View online: www.adobe.com/go/flex2_livedocs Get the PDF: www.adobe.com/go/flex_documentation
Adobe Flex Language Reference	Description of the ActionScript and MXML APIs for Flex.	• View in Flex Builder: Select Help › Help Contents, and then select Flex Help › Adobe Flex 2 Language Reference • View online: www.adobe.com/go/flex2_livedocs

Accessing additional online Flex Builder resources

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
Flex Documentation Resource Center	Product manuals in PDF format, errata, tutorials, 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

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. To see a list of documents available in Help, see "Accessing the Flex Builder documentation" on page 17.

This section covers the following topics:

Opening Help

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

To open Flex Builder Help:

- Select Help > Help Contents.

  All the Flex and Flex Builder documentation is available in the help system. For a complete list of available documentation, see "Getting the most from the Flex and Flex Builder documentation" on page 17.

Using dynamic help

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

To enable Dynamic Help:

- Select Help > Dynamic Help.

Using context-sensitive help

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

To access context-sensitive help for workbench elements:

1. Select an editor, view, dialog box, or other user interface element in the workbench.
2. Press F1.

For quick access to the *Adobe Flex Language Reference* while writing code, see "Getting help while writing code" on page 173.

Searching Help

You can do a full text search of Flex and Flex Builder Help.

To search in-product help:

1. In Flex Builder Help (Help > Help Contents), click the Search tab.
2. Type a word or phrase in the text box, and click Go.
3. Double-click a topic in the list of results to display it.

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

Selecting 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.

To select a search scope:

1. Click the Search Scope link in the Help viewer (Help > Help Contents).

   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.

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

Using Help bookmarks

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

To 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.

To view your Help bookmarks:

- In Flex Builder Help, select the Bookmarks tab.

  All the bookmarks you set are listed. Double-clicking a bookmark displays 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.

To display Help in an external browser:

1. In Flex Builder, select Window > Preferences > Help.
2. Select the Use External Browser option.
3. (Optionally) 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.

Setting Help preferences

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

To set Help preferences:

1. Select Window > Preferences > Help.
2. Set the following options as needed:

   Use External Browser lets you display Help in the web browser of your choice. Select the Use External Browser option and then select the Web Browser link to select your web browser. (See "Changing the default web browser" on page 211.)

   Open Window Context Help determines how context-sensitive Help links are displayed when you press F1 in the workbench. 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 choose to instead display the context-sensitive Help links in an *infopop* (similar to a tooltip).

   Open Dialog Box Help lets you show context-sensitive Help links in an infopop when pressing F1 in a dialog box.

   Help View Document Open Mode determines where Help content appears. When you select Help Contents from the Help menu, Help appears in the Help viewer; when you press F1, Help appears in the Dynamic Help view. This is the default behavior when the Open in Place option is selected. To display the Help content as a document in the editor area, select Open in Editor Area.
3. Click OK.

NOTE

The preferences on the Help Server page (Window › Preferences › Help › Help Server) are not supported by Flex Builder.

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 important getting started 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.

Printing the Flex and Flex Builder documentation

You can print single help topics from the Flex Builder Help viewer or access the documentation PDF files on the Adobe website (see "Accessing the Flex Builder documentation" on page 17).

To print documentation in the Flex Builder Help viewer:

- With a Help topic displayed in the Help viewer, select Print Page in the Help viewer toolbar.

Discussing the Flex and Flex Builder documentation with LiveDocs

The Flex and Flex Builder documentation is available online in LiveDocs format. The LiveDocs version of the Flex and Flex Builder Help looks similar to the in-product help, but it lets you comment on the contents of individual help pages. You can add useful information on a specific Flex or Flex Builder topic based on your own experience, or view suggestions from other Flex designers and developers.

For Flex LiveDocs, see www.adobe.com/go/flex2_livedocs.

CHAPTER 2

Flex Builder Workbench Basics

2

This topic provides an overview of the Adobe Flex Builder workbench. Flex Builder is built on Eclipse, an open-source, integrated development environment (IDE); you use it to develop Flex 3 and ActionScript 3.0 applications using powerful coding, visual layout and design, build, and debugging tools.

This topic contains the following sections:

Flex Builder basics

The Flex Builder workbench is a full-featured development environment that is tailored to assisting you in developing Flex, Apollo, 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 3 and ActionScript 3.0 applications in this full-featured IDE. Much of the basic functionality of the IDE comes from Eclipse. For example, managing, searching, and navigating resources are core features of the IDE. The Flex Builder plug-ins add the features and functionality needed to create Flex 3 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're 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 46), 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. It's essentially a special work environment that helps you accomplish a specific type of task. For example, Flex Builder contains two perspectives. The Development perspective is used for developing applications and the Debugging perspective is used when debugging your applications.

If you're using the Flex Builder plug-in configuration (see "Flex Builder configurations" on page 9), 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 30.

Editor An editor allows you to edit files of various types. Which editors you have available to you depends on the number and type of Eclipse plug-ins you have installed. Flex Builder contains editors for writing MXML, ActionScript 3.0, and CSS code. For more information about Flex Builder code editing, see Chapter 9, "Code Editing in Flex Builder," on page 165.

Views A view typically supports an editor. For example, when editing MXML, the Components and Flex Properties views are also displayed in the 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 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 and in 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 60.

Resource The term *resource* is used generically to refer to the files and folders within the projects in a workspace. For more information, see "Managing project resources" on page 65.

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 3, ActionScript 3.0, and Library projects. For more information, see Chapter 3, "Working with Projects," on page 49.

Launch configurationA *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, if needed, modify these settings. For more information, see "Running your applications" on page 207.

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 editors

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 Chapter 9, "Code Editing in Flex Builder," on page 165.

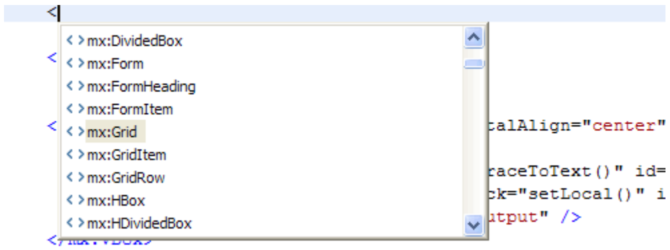
ActionScript editor You use the ActionScript editor to edit ActionScript class and interface files. Although you can embed ActionScript functions into an MXML file by using the `<mx:Script>` tag, it's a common practice to define classes in separate ActionScript files and then import the classes into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

CSS editor You use the CSS editor to edit cascading style sheets. You can then apply styles to the visual elements of your applications. For more information, see Chapter 20, "Using Styles and Themes" in *Flex Developer's 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.



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 about using Content Assist, see "About Flex Builder content assistance" on page 169.

Code navigation

Code navigation simplifies the burden of working with code, especially in large projects with many resources. Code navigation includes the ability to a 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, and the Outline view, which hierarchically presents, and allows you to navigate to, all of the user interface and code elements in a file. For more information, see "Navigating and organizing code" on page 173.

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 code" on page 179.

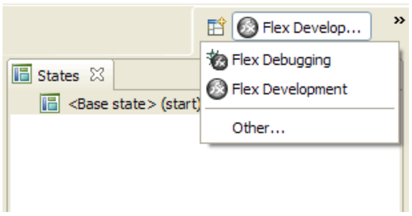
Find references and code refactoring

To better understand the scope of code changes, enable new features, or globally modify code, you can use the Flex Builder search feature to find all references to unique identifiers in a given file, project, or workspace (for example, type, variable, function, accessors, metadata). Then you can use the refactor feature to rename identifiers in your code while updating all references as appropriate. Flex Builder's refactor feature also enables you to move types (interfaces, classes) and update names and references accordingly. For more information, see "Finding references and refactoring code" on page 181.

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: one for development and design and the other for debugging.

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 > Open Perspective from the main menu. Or, you can use the *perspective bar*, which is located in the main workbench tool bar.



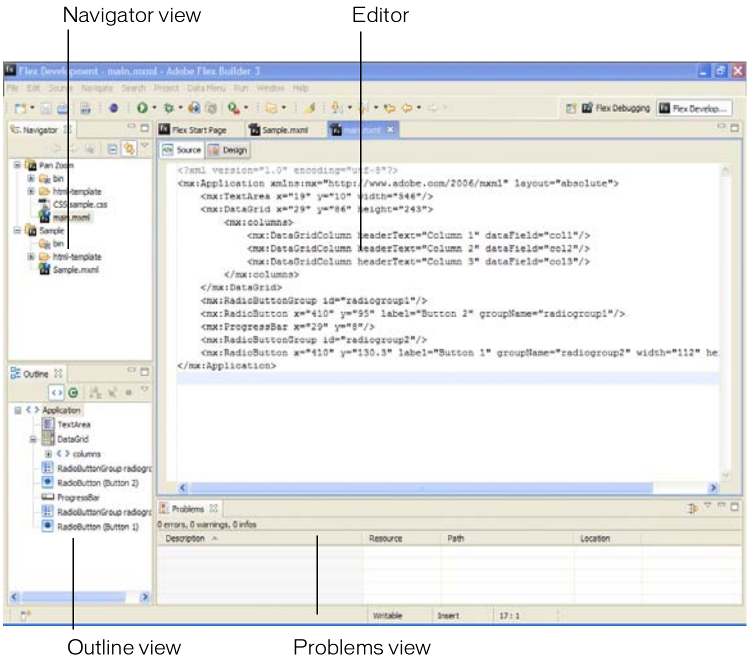
If you're using the plug-in configuration of Flex Builder and have other Eclipse plug-ins installed, you might have many additional perspectives. While predefined perspectives are delivered with each Eclipse plug-in, you can customize them to your liking or create your own. Customizing or creating a 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 85.

For more information about the Flex Builder perspectives, see the following topics:

- "The Flex Development perspective" on page 31
- "The Flex Development perspective in Design mode" on page 34
- "The Flex Debugging perspective" on page 37
- "Other useful workbench views" on page 40

The Flex Development perspective

The development perspective includes the editors and views you need to create Flex 3 and ActionScript 3.0 applications. When you create a project, Flex Builder switches into the development perspective so you can begin developing your application.



```
<?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>
```

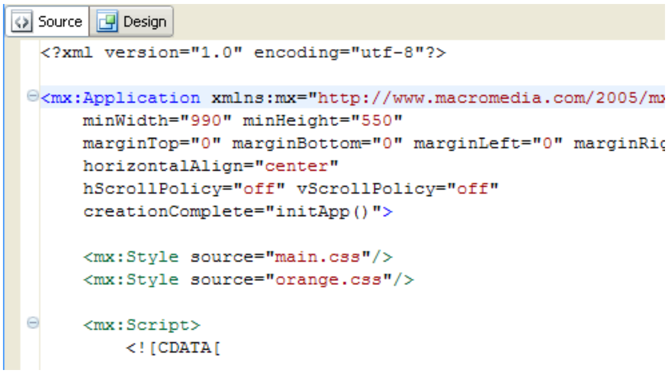
The focal point of the perspective (and the workbench generally) is the editor area. The editor area contains all of the currently open documents in a multitab interface. The supporting views are placed around the editor area. Perspectives predefine the layout of all the elements within it, but you may rearrange them to your liking. For more information, see Chapter 4, "Navigating and Customizing the Flex Builder Workbench," on page 85.

In Source (code editing) mode, the development perspective contains the following elements.

The 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 with which you're working.

The MXML editor operates 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 and CSS editors are single-purpose editors that are used to create ActionScript and CSS files.

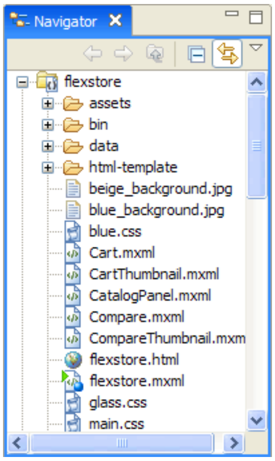


```
<?xml version="1.0" encoding="utf-8"?>

<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 Chapter 9, “Code Editing in Flex Builder,” on page 165. For more information about the development perspective in Design mode, see “The Flex Development perspective in Design mode” on page 34.

The Navigator view

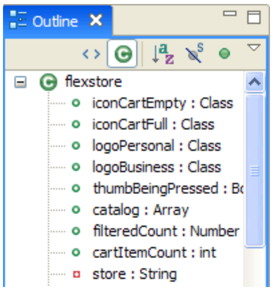
The Navigator view contains all of the projects and resources in the workspace and is therefore an essential element of the Flex Builder workbench and is always displayed in the development and debugging perspectives.



For more information about the Navigator view and working with projects, see Chapter 3, “Working with Projects,” on page 49.

The Outline view in Source mode

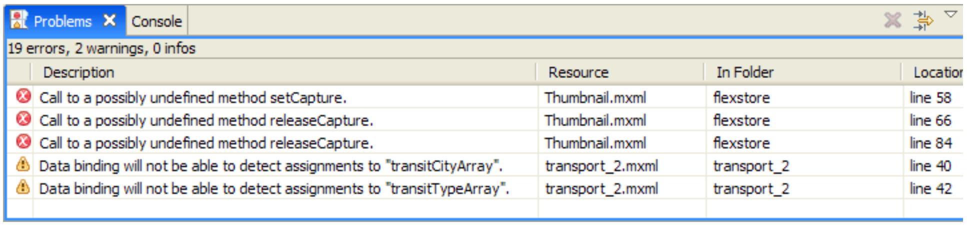
In Source mode, the Outline view presents a hierarchical view of the code structure of the selected MXML or ActionScript document so that you can inspect and navigate the sections or lines of code in the document. The Outline view also displays syntax error alerts that the compiler generates. This view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



For more information about using the Outline view in Source mode, see “Using the Outline view to navigate and inspect code” on page 174.

The 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	Locatior
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 Chapter 11, “Running and Debugging Applications,” on page 205.

NOTE

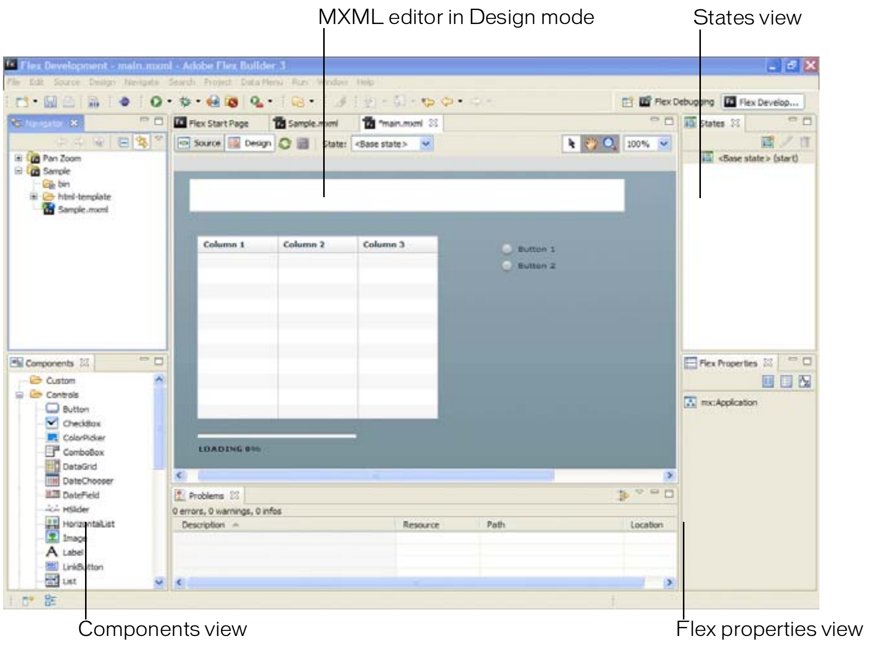
You can also optionally add the Tasks and Bookmarks views. These views provide additional shortcuts for managing and navigating your code. For more information about these views, see “About markers” on page 182. For an introduction to the optional views that are available in Flex Builder, see “Other useful workbench views” on page 40.

The Flex Development perspective in Design mode

You visually lay out and design your Flex applications in the MXML editor in 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 in Design mode, the Outline view displays the MXML structure of your Flex applications.

For more information about designing Flex applications in Flex Builder, see Chapter 5, "Building a Flex User Interface," on page 105.

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 74 and Chapter 11, "Running and Debugging Applications," on page 205.



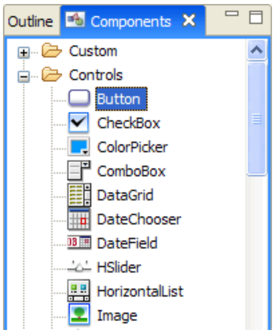
In Design mode, the development perspective contains the following views.

The MXML Editor in Design mode

In Design mode, you interact with your Flex applications visually; dragging and dropping components on to the design surface, 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 pan/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 Chapter 5, "Building a Flex User Interface," on page 105.

The Components view

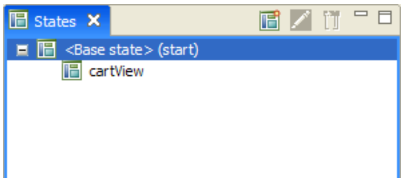
The Components view contains all of the standard Flex components and you select and add them to the design surface. As you create your own custom components, they are also displayed in the Components view.



For more information, see "Adding components" on page 108.

The States view

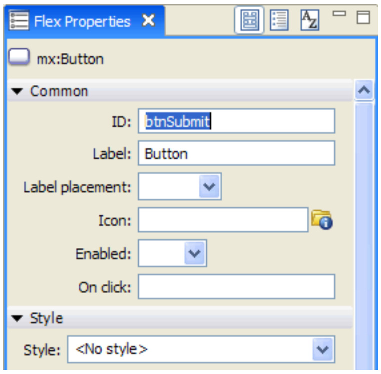
Flex allows you to create applications that change their appearance based on events that are triggered directly by the user or events that are generated programmatically. These user interface transformations are referred to as *view states*. You create and manage view states in the States view.



For more information about view states, see Chapter 6, "Adding View States and Transitions," on page 143.

The Flex Properties view

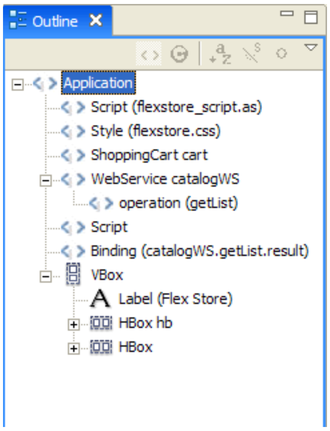
When a Flex component is selected, its properties are displayed in the Flex Properties view. You can set and edit properties as appropriate. You can view a component's properties graphically (as shown in the following example), and as a categorized or alphabetical list.



For more information, see "Setting component properties" on page 120.

The Outline view in Design mode

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.

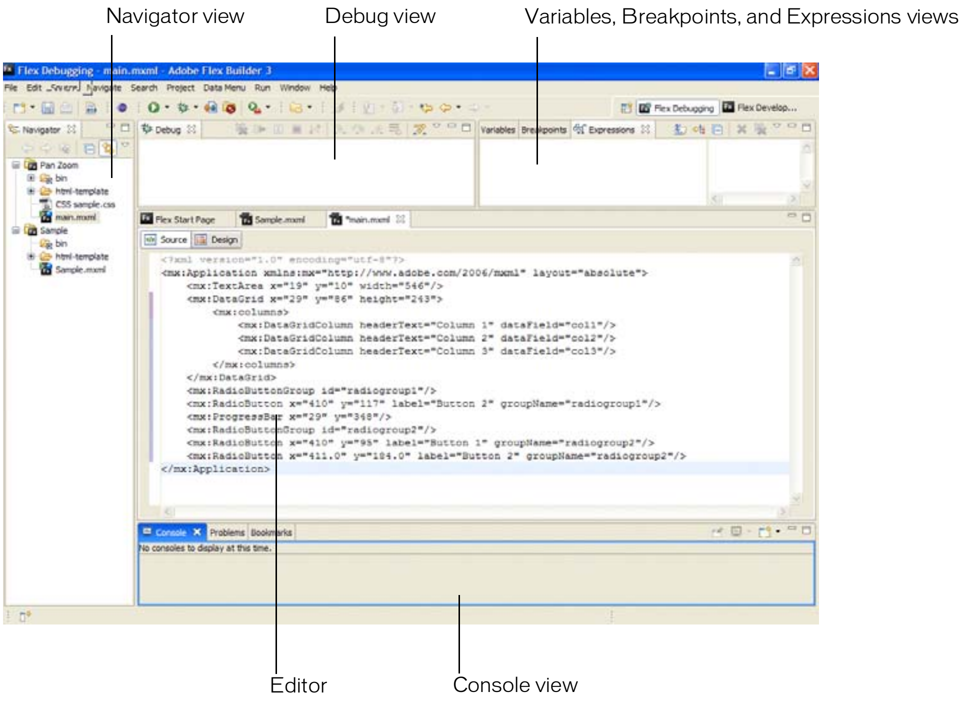


For more information about working with the Outline view in Design mode, see "Inspecting the structure of your MXML" on page 126.

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>
```

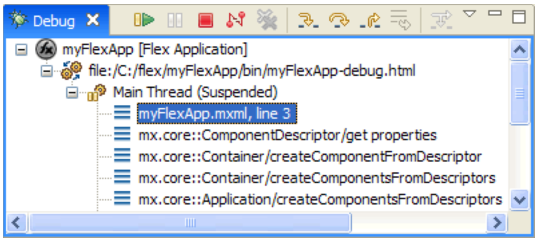
```
No consoles to display at this time.
```

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 the following views.

The 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
```

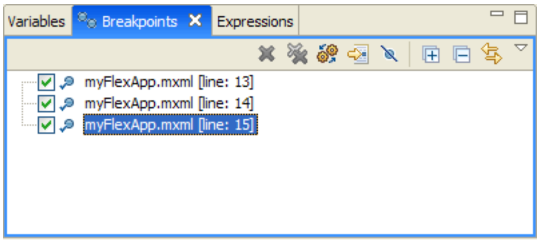
For more information about working with the debug view, see "Managing the debugging session in the Debug view" on page 217.

Flex applications are single-threaded (not multi-threaded 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.

The 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.

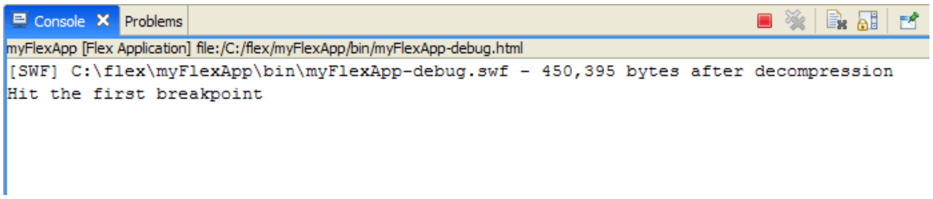


```
myFlexApp.mxml [line: 13]
myFlexApp.mxml [line: 14]
myFlexApp.mxml [line: 15]
```

For more information, see "Managing breakpoints in the Breakpoints view" on page 214.

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 serves the same purpose as the *output panel* in Flex Builder 1.5.

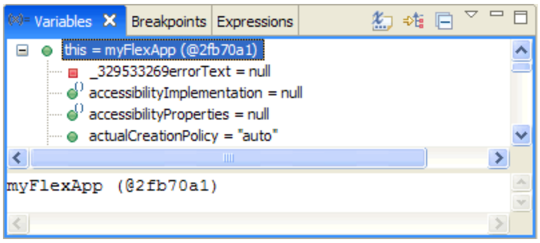


```
myFlexApp [Flex Application] file:/C:/flex/myFlexApp/bin/myFlexApp-debug.html
[SWF] C:\flex\myFlexApp\bin\myFlexApp-debug.swf - 450,395 bytes after decompression
Hit the first breakpoint
```

For more information, see "Using the Console view" on page 218.

The Variables view

The Variables view displays information about the variables in the currently-selected stack frame and serves the same purpose as the *locals* window does in other debuggers (including Flex Builder 1.5). You can select variables to watch (in the Expressions view) and also change variable values during the debugging session and see the changes in the currently running SWF file, to experiment with fixes for the problem you're trying to resolve.

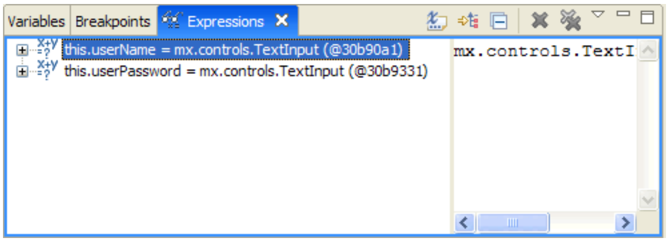


```
this = myFlexApp (@2fb70a1)
  _329533269errorText = null
  accessibilityImplementation = null
  accessibilityProperties = null
  actualCreationPolicy = "auto"
myFlexApp (@2fb70a1)
```

For more information, see "Managing variables in the Variables view" on page 218.

The Expressions view

The Expressions view is used to monitor a set of critical variables and serves the same purpose as the *watch* window does in other debuggers (in Flex Builder 1.5 this was referred to as the *watch list*). You can choose the variables you consider critical in the Variables view and add them to and monitor (watch) them in the Expressions view.



```
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 219.

For more information about debugging Flex and ActionScript applications, see Chapter 11, "Running and Debugging Applications," on page 205.

Other useful workbench views

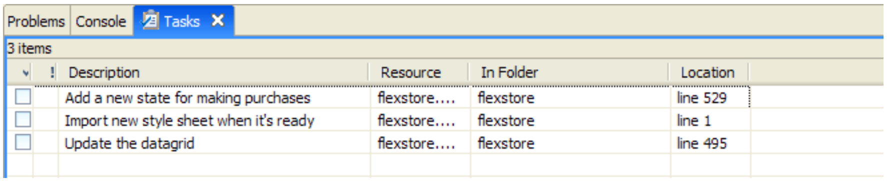
In addition to the editors and views associated with Flex Builder's default development and debugging perspectives, the workbench contains other views that help you streamline the application development process.

You can access views that are not already displayed with a perspective and add them to the workbench by selecting Window > Show Views > Other. 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 89.

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

The Tasks view

The Tasks view is used to manage task markers. The workbench automatically generates task markers, or you create them manually to mark lines of code or resources that require more work before they are complete. Selecting a task locates and displays it in the workbench.

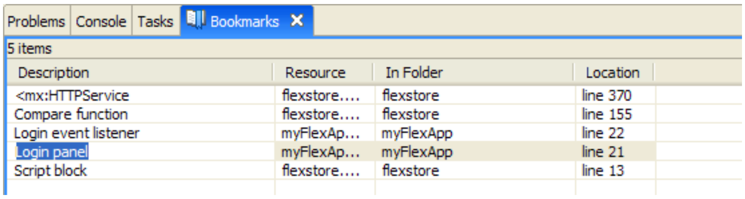


	!	Description	Resource	In Folder	Location
☐		Add a new state for making purchases	flexstore....	flexstore	line 529
☐		Import new style sheet when it's ready	flexstore....	flexstore	line 1
☐		Update the datagrid	flexstore....	flexstore	line 495

For more information about the Tasks view, see "About markers" on page 182.

The 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.

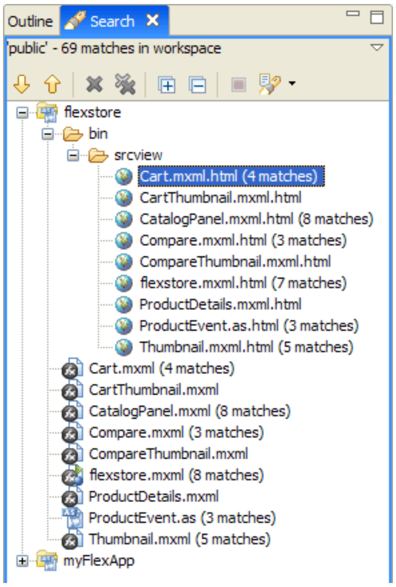


Description	Resource	In Folder	Location
<mx:HTTPService	flexstore....	flexstore	line 370
Compare function	flexstore....	flexstore	line 155
Login event listener	myFlexAp...	myFlexApp	line 22
Login panel	myFlexAp...	myFlexApp	line 21
Script block	flexstore....	flexstore	line 13

For more information about the Bookmarks view, see "About markers" on page 182.

The 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, to filter the list of search results, and so on.


Outline
Search
'public' - 69 matches in workspace
flexstore
bin
srcview
Cart.mxml.html (4 matches)
CartThumbnail.mxml.html
CatalogPanel.mxml.html (8 matches)
Compare.mxml.html (3 matches)
CompareThumbnail.mxml.html
flexstore.mxml.html (7 matches)
ProductDetails.mxml.html
ProductEvent.as.html (3 matches)
Thumbnail.mxml.html (5 matches)
Cart.mxml (4 matches)
CartThumbnail.mxml
CatalogPanel.mxml (8 matches)
Compare.mxml (3 matches)
CompareThumbnail.mxml
flexstore.mxml (8 matches)
ProductDetails.mxml
ProductEvent.as (3 matches)
Thumbnail.mxml (5 matches)
myFlexApp

For more information about the Search view, see "Searching in the workbench" on page 99.

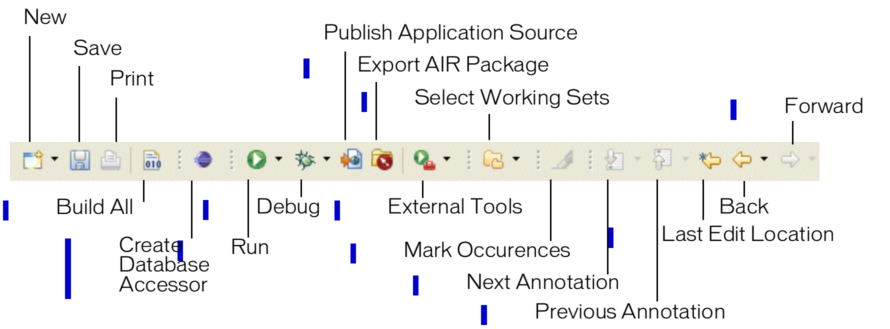
Workbench menus, toolbars, and shortcuts

All of the workbench commands are contained in the menu system, in right-click context menus, from toolbars, and through keyboard shortcuts. For more information, see the following topics:

- "The workbench toolbar" on page 43
- "The MXML editor toolbar" on page 44
- "Using keyboard shortcuts" on page 46

The workbench toolbar

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



The following options appear in the workbench toolbar:

New displays a pop-up menu that displays all the types of projects and documents you can create. For more information about the types of projects and resources you can create, see "About project resource types" on page 54.

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

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

Build All

Create Database Accessor

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 207.

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 213.

Publish Application Source allows you to choose some or all of your application source files to be published with the application SWF file, so that users may view the source code. For more information, see "Publishing application source code" on page 201.

Export AIR Package

External Tools is used to select a custom launch configuration.

Select Working Sets

Mark Occurences

Next Annotation

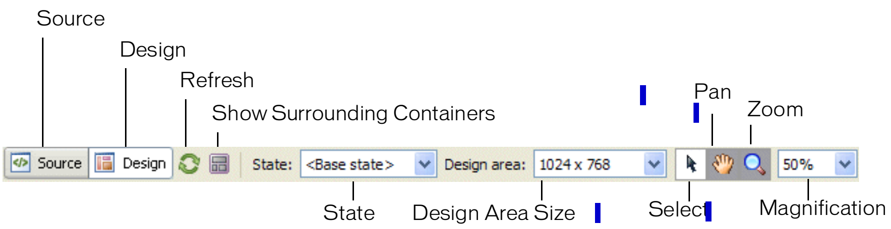
Previous Annotation

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 commands for controlling the editor.



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 Chapter 22, “Creating Skins” in *Flex Developer’s Guide*.

Show Surrounding Containers expands the Design mode view so that you can see and select individual components. For more information, see “Showing surrounding containers” on page 125.

State displays all the defined views states. Selecting view states updates the display of the visual design. For more information, see Chapter 6, “Adding View States and Transitions,” on page 143.

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 112.

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

Pan Mode allows you to grab and scroll around in design view 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 button to return the design view to 100%.

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

Using keyboard shortcuts

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

To 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 96.

Extending the Flex Builder workbench

Flex Builder is a collection of Eclipse plug-ins that provide the tools you need to create Flex 3 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

2

This part describes in detail how to create and manage projects and use the many features of the Flex Builder workbench.

The following chapters are included:

CHAPTER 3

Working with Projects

3

This content has not been updated for Flex Builder 3 Beta 1.

Adobe 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 Navigator view, which lets you add, edit, and delete resources. You can also close projects within a workspace, import resources, link to external resources, and so on.

In addition to Flex projects, Flex Builder also provides a more basic project type called an *ActionScript project*. Using an ActionScript project, you can code and debug ActionScript 3.0 applications that directly access the Flash Player APIs and are compiled into SWF files. ActionScript projects do not use Flex framework or the MXML language. No Flex server is associated with ActionScript projects.

You can also use Flex Builder to create library projects that generate shared component library (SWC) files, which contain components and other resources that you can share between applications or distribute to other developers.

This topic contains the following sections:

Understanding Flex Builder projects

Before you create projects in Flex Builder, it's helpful to first review the key concepts involved in creating and managing application projects in Flex Builder. (If you prefer to get started immediately and create a project, see "About Flex Builder project types" on page 52 and "Creating Flex projects" on page 56.)

Flex 3 and ActionScript 3.0 applications

Using Flex Builder you can create Flex 3 and ActionScript 3.0 applications. Flex applications can be compiled into stand-alone SWF files that do not require the Flex server. When you build Flex applications that require the Flex server, you can use either Flex Builder or the server to compile the application SWF files. For more information, see "About Flex Builder project types" on page 52 and "About ActionScript projects" on page 74.

Flex libraries

You can also use Flex Builder to 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 more information, see "About library projects" on page 78.

Applications contained in projects

To begin building a Flex or ActionScript 3.0 application in Flex Builder, you must first create a project. When you create a Flex project, a main application file is created for you. You then add, as needed, 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 3.0 project, a main ActionScript file is created and you can then build an application by using ActionScript and the Flash Player API. For more information, see "Creating Flex projects" on page 56 and "Managing projects" on page 60.

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. You can use only one workspace per instance of Flex Builder; however, you can select a different workspace each time you start 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 Navigator view, and you can manage them as you need to—adding resources, organizing your projects into folders, building projects in the workspace, and so on. For more information, see "Managing projects" on page 60 and "Managing project resources" on page 65.

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 70.

More than one application in a project

Flex Builder lets you define more than one project file 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 root folder of your project. For more information, see "Managing project application files" on page 64.

Automatic project builds

When your projects are built, debug and release SWF files are generated in the output folder. 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 189.

Custom Ant scripts

Apache Ant is a Java-based build tool that you can use to create custom scripts for building your Flex applications in Flex Builder. For more information, see "Customizing builds with Apache Ant" on page 201.

Learn more about Flex Builder projects

If you haven't already done so, review the getting started lesson in Chapter 8, "Create Your First Application" in *Getting Started with Flex.* For more information, see the following sections in this topic:

- "About Flex Builder project types" on page 52
- "Projects in the Navigator view" on page 53
- "About project resource types" on page 54
- "Creating Flex projects" on page 56
- "Managing projects" on page 60

About Flex Builder project types

You use Flex Builder to create projects of the following types in the following configurations.

Flex projects

Flex 2 project configuration options are based on how your Flex application accesses data and if you have Flex Data Services installed. Here are the options:

- **Basic Flex project**

  The basic configuration allows you to develop Flex applications using the Flex framework and to access data using XML and Web services. You compile your projects in Flex Builder and then deploy them to a web server.
- **ColdFusion Flash Remoting Service project**

  This project configuration allows you to access data using the ColdFusion Flash Remoting Service.
- **Data Services project**

  This project configuration allows you to take advantage of the LiveCycle Data Services ES features. You must have LiveCycle Data Services ES installed. You compile your projects by using Flex Builder, or you can use LiveCycle Data Services ES to compile the application source files on the server.

ActionScript 3.0 projects

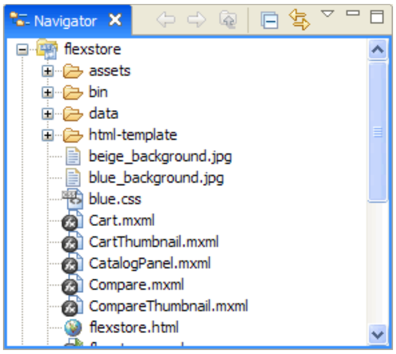
Based on the Flash API, not the Flex framework, ActionScript projects are intended to let ActionScript developers use Flex Builder to code, build, and debug ActionScript-only applications. Because these projects do not use MXML to define a user interface, you cannot view the application layout and design in Design mode. You work exclusively in the source editor, using 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 74.

Library projects

Library projects are used to package and distribute components and other resources. They generate SWC files that you can add to other projects or distribute to other developers. For more information, see "About library projects" on page 78.

Projects in the Navigator view

All of the projects in a workspace are displayed in the Navigator view, as the following example shows. Using this view, you manage your projects by adding resources (folders and files), deleting resources, importing and linking to external resources, moving resources to other projects in the workspace, and so on.



Flex Builder provides wizards to help you create projects. The New Flex Project wizard automatically generates project configuration files, the output (`bin`) folder where your compiled application resides, and the main application file. (In the previous example, the main application file is myflexproject.mxml.) The New ActionScript Project wizard generates a main ActionScript application file.

From the Navigator view, you can open the project resources for editing. For example, double-clicking the myflexproject.mxml file opens it in the Flex Builder MXML editor. In the MXML editor, you can edit MXML and also ActionScript in `<mx:Script>` blocks and CSS in `<mx:Style>` blocks, or you can switch into 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 Chapter 9, "Code Editing in Flex Builder," on page 165 and Chapter 5, "Building a Flex User Interface," on page 105.

You then add additional projects, files, and folders and organize and manage them as needed (see "Managing project resources" on page 65).

You can also modify the Navigator view's display. 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 85.

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

For more information about working with projects in the Navigator view, see "Managing projects" on page 60 and "Managing project resources" on page 65.

About project resource types

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

Resource type	Description
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 (see "Managing project application files" on page 64).
MXML component	A standard Flex component file with the `<mx:Canvas>` tag as the root MXML element. See "Creating MXML components visually" on page 158.
ActionScript file	A text file template for creating ActionScript functions.

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 76.
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 77.
CSS file	A text file template for creating a Cascading Style Sheets file.
Folder	A standard file system folder for organizing the contents of your projects. See "Creating folders and files in a project" on page 66.
File	An unformatted text file. See "Creating folders and files in a project" on page 66.
Other	Select File > New > Other to add any other file types that are registered in Flex Builder. 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 93. You can always add unregistered file types to your projects by importing them (see "Importing resources into a project" on page 67).

For more information about adding resources to your projects, see "Creating folders and files in a project" on page 66.

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 you want to create, the project's name, location, and other options.

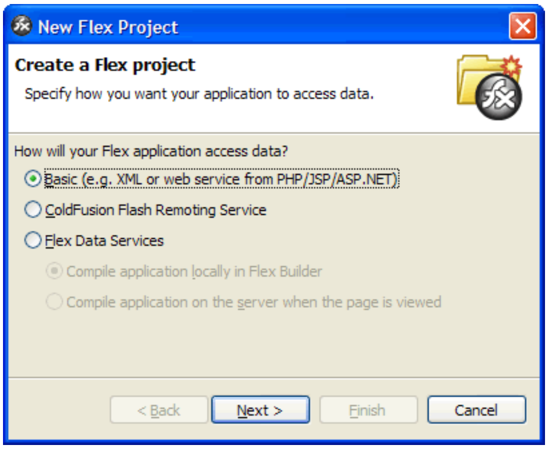
This section describes how to create Flex projects and select a project configuration that suits your data access requirements. For information about creating an ActionScript project, see "Creating ActionScript projects" on page 74. For information about creating library projects, see "About library projects" on page 78.

Selecting the Flex project configuration

As described in "About Flex Builder project types" on page 52, there are three configuration options for Flex projects that are based on how your application will access data. The first step in creating a Flex project is to determine which configuration to choose.

To select a Flex project configuration:

1. Select File > New > Flex Project. The New Flex Project wizard appears:



New Flex Project

Create a Flex project

Specify how you want your application to access data.

How will your Flex application access data?

[✓ Basic (e.g. XML or web service from PHP/JSP/ASP.NET)]
ColdFusion Flash Remoting Service
Flex Data Services
[✓ Compile application locally in Flex Builder]
Compile application on the server when the page is viewed

2. Select the option that represents how your application accesses data. In most cases, you select the first option, which gives you basic data access options such as XML and Web services.

   If you have ColdFusion MX 7 Updater 2 (7.0.2) installed, you can create a Flex project that uses the ColdFusion Flash Remoting Service to access data.

   If you have Flex Data Services installed, you can create an FDS project. You must also select how the application is compiled (locally in Flex Builder or remotely on the server).

3. Click Next to continue creating your Flex project.

If you're creating a basic Flex project, see "Creating a basic Flex project" on page 57; if you're creating a project that uses the ColdFusion Flash Remoting Service, see "Creating a Flex ColdFusion Flash Remoting service project" on page 57; if you're creating a project that uses FDS, see "Creating a Flex Data Services project" on page 58.

Creating a basic Flex project

The basic configuration allows you to develop Flex applications using the Flex framework and gives you basic data access options such as XML and Web services. You compile your projects in Flex Builder, and then deploy them to a web server.

To create a basic Flex project:

1. After you selected the project configuration type, click Next, and set the following project properties:

   Project Name is the name of your project.

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

2. Click Next to set optional project properties (see "Setting optional project properties" on page 60) or click Finish to create the project.

For more information about working with projects, see "Managing projects" on page 60.

Creating a Flex ColdFusion Flash Remoting service project

To access data by using the ColdFusion Flash Remoting service, you must have ColdFusion MX 7 Updater 2 (7.0.2), and (optionally) the ColdFusion Extensions for Flex Builder. For more information, see the ColdFusion product page.

To create a Flex project that uses the ColdFusion Flash Remoting Service:

1. After selecting the ColdFusion Flash Remoting data access option from the first screen of the New Flex Project wizard, you are prompted to specify the following:

   Use Local ColdFusion Server specifies that the local installation of ColdFusion MX 7.0.2 is used by the project.

Root Folder specifies the ColdFusion root folder (for example: C:\CFusionMX7\wwwroot).

Root URL specifies the root URL (for example: http://localhost:8500/).

2. Click Next to continue.
3. You are prompted to specify the following:

   Project Name is the name of your project.

   Project Location is the location of your project. The default location is the workspace, which is, by default, My Documents\Flex Builder 3*project_name* (for example, C:\Documents and Settings*Flex Developer*\My Documents\Flex Builder 3*myFlexApp*). You can choose a different project location by deselecting the Use Default Location option.
4. Click Next to set optional project properties (see "Setting optional project properties" on page 60) or click Finish to create the project.

Creating a Flex Data Services project

To use Flex Data Services in your Flex projects, you must have LiveCycle Data Services ES installed.

You have two options for creating a LiveCycle Data Services ES project. The first option compiles the application locally, and then saves the files (including the SWF file and HTML wrapper) on the server. The second option compiles the application source file directly on the server.

To create a Flex Data Services project:

1. After selecting the Flex Data Services Data Access option from the first screen of the New Flex Project wizard, you are prompted to specify the following:

   Use Default Location for Local Flex Data Services server specifies that the root folder and root URL are set to the default settings of your local Flex development server. This option is selected by default.

   Root Folder is the root folder of the Flex server (web application) from which to serve 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 root URL that you specify. If you are using a remote server, you can specify the location—for example, *myServer**MyApplications*\jrun4\servers\default\flex.

Root URL is a valid root URL of the Flex server (web application) from which to serve 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/.

2. After you finish entering the server properties and click Next, you are prompted to set the following project properties:

   Project Name is the name of your project.

   Project Location is the location of your project. The default location is C:\fds2\jrun4\servers\default\flex. You can choose a different project location.

   If you selected the server compile option, you will also see the following:

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

3. If you selected the option to compile the project in Flex Builder, click Next to set optional project properties (see "Setting optional project properties" on page 60), or click Finish to create the project. If you selected the option to compile the project on the server, click Finish to create the project.

When you create a LiveCycle Data Services ES project in Flex Builder, Flex Builder either creates a directory with same name as your project or uses an existing directory that has that name. That directory is a subdirectory of the root folder that you specified for the project.

For applications that compile locally, Flex Builder saves MXML files in the local Flex Builder project directory, and saves SWF files and HTML wrapper files on the server. Creating a project that compiles locally is a good idea if you plan to deploy your finished application as a SWF file with an HTML wrapper file rather than as an MXML file, because Flex Builder generates the SWF and HTML wrapper for you and saves them to the appropriate location.

For applications that compile on the server, MXML files are saved on the server. No HTML wrapper file is saved when the application is compiled.

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 should map the root of a LiveCycle Data Services ES web application. Before compiling your Flex application in Flex Builder, you must start the corresponding LiveCycle Data Services ES web application.

Setting optional project properties

You can set optional project properties when you are creating a Flex project or after the project has been created by selecting the project in the Navigator view, and then selecting Project > Properties.

To set optional project properties:

1. Set the following advanced project properties:

   Source path is used 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. 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. For more information, see "Modifying a project's build path" on page 195.

   Library path is used to 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. For more information, see "Using SWC files in your projects" on page 82.

   Main source folder is 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 is the name of main application MXML file. By default, Flex Builder uses the project name as the main application filename. You can change this name.

   Output folder is the location of the compiled application files. By default, this is the bin folder but you can change this if you like.

   Output folder URL is used to specify the server location of the compiled application files. This is optional.

Managing projects

To manage your projects and resources, you use the Navigator view to add and import resources into projects, move and delete resources, and so on.

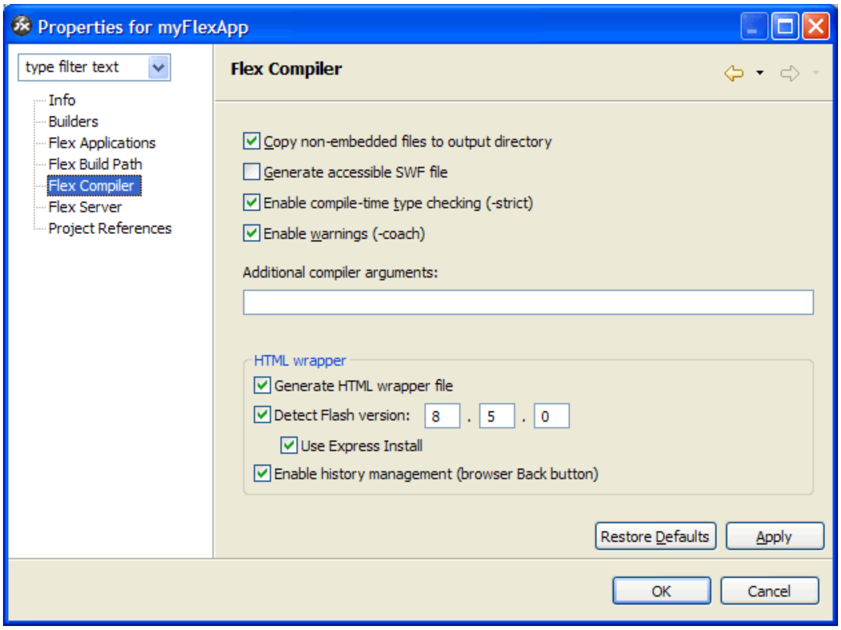
The following topics explain how to manage projects in Flex Builder:

- "Setting Flex project properties" on page 61
- "Importing projects" on page 62

- "Moving a project from one workspace to another" on page 63
- "Deleting projects" on page 63
- "Closing and opening projects" on page 63
- "Switching the main application file" on page 64
- "Managing project application files" on page 64

Setting Flex project properties

Each Flex project has its own set of properties, as the following example shows. You can set these properties by first selecting the project in the Navigator view and then Project > Properties from the main menu, or right-click (Control-click on Macintosh) to display the context menu and select Properties.



[✓] Copy non-embedded files to output directory
[ ] Generate accessible SWF file
[✓] Enable compile-time type checking (-strict)
[✓] Enable warnings (-coach)

Additional compiler arguments:

HTML wrapper

[✓] Generate HTML wrapper file
[✓] Detect Flash version: 8 . 5 . 0
[✓] Use Express Install
[✓] Enable history management (browser Back button)

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

Info displays general information about the project, settings for text encoding, and the operating system line delimiter. (See "Setting project text encoding properties" in Help.)

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 new build scripts or import existing Ant build scripts. (See "Customizing builds with Apache Ant" on page 201.)

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 64.)

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 a project's output folder" on page 195 and "Building projects manually" on page 197.)

Flex Compiler specifies optional compiler preferences such as generating an accessible SWF file, enabling compiler warnings and type checking, specifying additional compiler arguments, and so on. (See "Advanced build options" on page 197.)

Flex Server determines the location of the Flex root folder and the Flex server URL (for Flex Data Services projects only). (See "Creating a Flex Data Services project" on page 58.)

Project References lists the projects that the current project references.

Importing projects

You can work with many projects simultaneously. All projects in the current workspace are displayed in the Navigator view. You can create projects or import existing projects into the workspace. Existing projects must be valid Flex Builder projects and must reside either in another workspace or, if removed from a workspace, in the file system.

To import existing projects into the workspace:

1. Select File > Import.
2. In the Import wizard, select Existing Projects Into Workspace and click Next.
3. In the Import Projects dialog box, select either the root directory or archive file option; then enter or browse for 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.
4. 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.

Moving a project from one workspace to another

Although you can't directly move a project from one workspace to another, you can use a combination of deleting and importing operations to accomplish the same thing. 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 63). After you've removed 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 choose to remove the project from the file system at the same time.

If you don't want to delete the project from the workspace, you can close the project instead. Closing the project lets you keep a reference to it in your workspace while also freeing some system resources. For more information, see "Closing and opening projects" on page 63.

To delete a project:

1. In the Navigator view, select the project you want to delete.
2. Select Edit > Delete or press the Delete key.
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

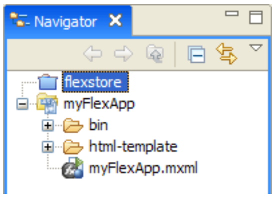
To save memory and improve build time without deleting a project, you can close it. When you close a project, you collapse the project and its resources while keeping a reference to it in the Navigator view. A closed project requires less memory than an open project, and is excluded from builds. (To delete the project from the workspace or the file system, see "Deleting projects" on page 63.)

You can easily reopen the closed project from the Navigator view.

To close a project:

1. In the Navigator view, select the project you want to close.
2. Right-click (Control-click on Macintosh) to display the context menu and select Close Project.

When a project is closed, its name remains visible in the Navigator view. You can select the project and either reopen it or delete it. The following example shows a closed project in the Navigator view.



To reopen a project:

1. In the Navigator view, select the closed project to open.
2. Right-click (Control-click on Macintosh) to display the context menu and select 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 64.

To switch the main application file:

1. In the 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 ActionScript 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, it can be run and 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.

To manage application files:

1. In the Navigator view, select a project.
2. Select Project > Properties.
3. In the Project Properties dialog box, select Flex Applications (or ActionScript Applications if you're working with an ActionScript project).
4. Add and remove application files as needed (see "Setting project application file properties" in Help).
5. Click OK.

Managing project resources

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

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

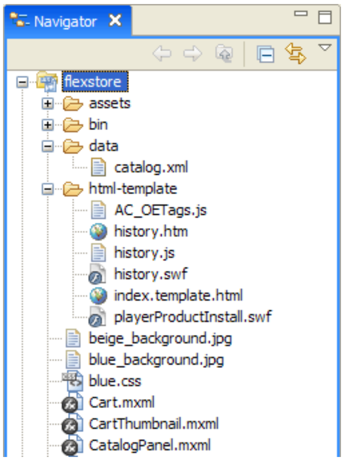
The following topics explain how to manage project resources in Flex Builder:

- "Creating folders and files in a project" on page 66
- "Deleting folders and files" on page 67
- "Importing resources into a project" on page 67
- "Exporting resources" on page 69
- "Moving resources between projects in a workspace" on page 69
- "Refreshing resources in the workspace" on page 70
- "Linking to resources outside the project workspace" on page 70
- "Adding resource folders to the project source path" on page 72
- "Adding project references" on page 73

- "Viewing resource properties" on page 73

Creating folders and files in a project

Projects contain folders and files, and you can add both 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:



To import existing folders and files, see "Importing resources into a project" on page 67.

To create a folder:

1. Select the Navigator view.
2. Select File > New > Folder.

   The New Folder dialog box appears.
3. If you have multiple projects in your workspace, select the project to which you want to add the folder.
4. Enter the folder name and click Finish.

To create a file:

1. Select the Navigator view.
2. Select File > New > File.

   The New File dialog box appears.

3. If you have multiple projects in your workspace, select the project to which you want to add the file.
4. 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 70.

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 (see "Linking to resources outside the project workspace" on page 70), you delete only the link from your project, not the resource itself. However, if you've linked to a folder and you delete any of the files in it, they are removed from the file system.

To delete a resource:

1. In the Navigator view, select the resource to delete.
2. Select Edit > Delete or press the Delete key.
3. To confirm that you want to delete the resource from the file system, click OK.

Importing resources into a project

You can import existing resources into a project by using the File > Import command. You can import resources from the file system or an archive file.

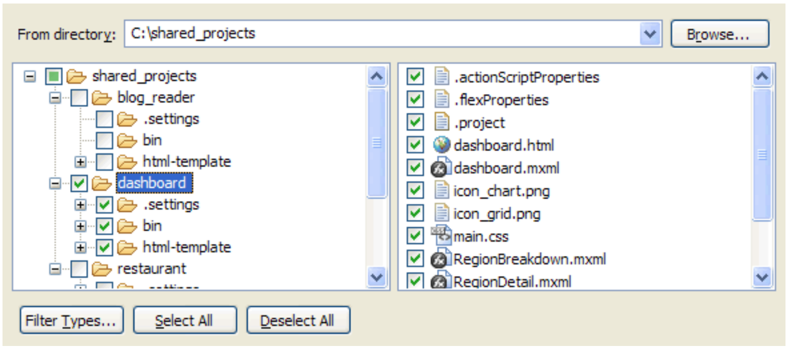
If you'd rather share a resource between projects in the workspace, you can link to the resource instead of importing it. For more information, see "Linking to resources outside the project workspace" on page 70.

To import resources into a project:

1. In the Navigator view, select the project to import resources into.
2. Select File > Import.

   The Import dialog box appears.
3. From the list of import sources, select File System and click Next.

4. Enter or browse to the resources' location, and select the folders and files to import.



5. (Optional) To filter the list of files, click the Filter Types button; in the Select Types dialog box, select the file extension to display in the Import dialog box and click OK.
6. In the Into Folder text box, specify the folder to add the resources to.
7. Set the following import options as needed:

   Overwrite Existing Resources Without Warning specifies that you want to overwrite any duplicate resources that are already in your project.

   Create Complete Folder Structure lets you duplicate the folder structure of the folders you selected to import.

   Create Selected Folders Only imports and creates folders for the selected folders only.
8. Click Finish.

TIP

As a shortcut for this import procedure, you can quickly add existing resources to a project or workspace by dragging them from the file system into the Navigator view, or by copying and pasting.

To import resources from an archive file:

1. In the Navigator view, select the project to import resources into.
2. Select File > Import.

   The Import dialog box appears.
3. From the list of import sources, select Archive File and click Next.
4. Enter or browse to the archive file's location.

   Select the folders and files to import.
5. (Optional) To filter the list of files, click the Filter Types button; in the Select Types dialog box, select the file extension to display in the Import dialog box and click OK.

6. In the Into Folder text box, specify the folder to add the resources to.
7. Select Overwrite Existing Resources Without Warning to overwrite any duplicate resources that are already in your project.
8. Click Finish to import the selected resources from the archive file.

Exporting resources

You can export project resources to another location in the file system by using the File > Export command.

To export project resources:

1. Select File > Export.

   The Export dialog box appears.
2. Select the export destination.

   You can export project resources to the file system or to an archive file.
3. Select the projects and resources to export.
4. Enter or browse to the export destination (the file system or an archive file).
5. Set the following import options as needed:

   Overwrite Existing Files Without Warning specifies that you want to overwrite any duplicate files that are already in your project.

   Create Directory Structure for Files lets you duplicate the directory structure of the folders you selected to import.

   Create Only Selected Directories exports and creates directories for the selected directories only.
6. Click Finish.

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.

To move resources between projects:

1. In the 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 70.

Refreshing resources in the workspace

When you add resources to a project that is contained in a workspace, you are organizing those resources in a location on a file system. 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 Navigator view.

You can also edit resources outside Flex Builder, directly in the file system.

By default, in the standalone 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.

To manually refresh the workspace:

- In the Navigator view, right-click (Control-click on Macintosh) and select Refresh from the context menu.

  All project resources in the workspace are refreshed.

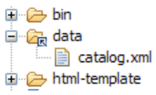
To turn off the automatic refresh preference:

1. Select Window > Preferences > 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 Navigator view (as the following example shows), so that you can distinguish between normal and linked resources.



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 67.

To link to resources outside the project workspace:

1. In the Navigator view, select the project to add linked resources to.
2. Select File > New > Folder (or File if you want to add files).

   The New Folder or New File dialog box appears.
3. Select the project or project folder where you want to add the linked resources.
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. 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 196.

To link to resources using a path variable:

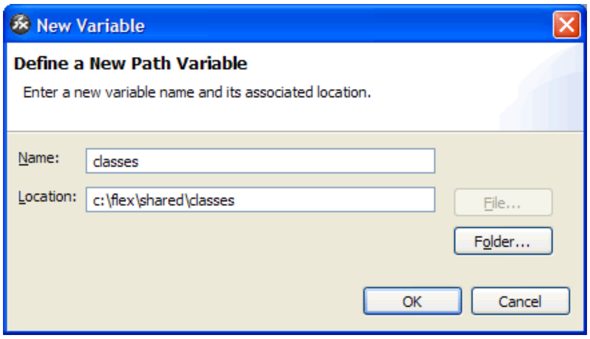
1. In the Navigator view, select the project to add linked resources to.
2. Select File > New > Folder (or File if you want to add files).

   The New Folder (or New File) dialog box appears.
3. Select the project or project folder to add the linked resources to.
4. Click the Advanced button to display the advanced options.
5. 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: classes

Location: c:\flex\shared\classes

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.

TIP You can also define and manage path variables by using the Flex Builder workbench preferences (Window › Preferences › General › Workspace › Linked Resources).

Adding resource folders to the project source path

To share resources between projects, you can place all shared resources into folders that can then be linked to each project using the project's source path. This is the best method for using shared resources such as classes, MXML components, images, and so on. 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.

To add an external resource folder to the source path:

1. Select the project in the Navigator view.
2. Select Project > Properties > Flex Build Path (or ActionScript Applications if you're working with an ActionScript project).
3. On the build path properties page, select the Source Path tab.
4. Click the Add Folder button. The Add Folder dialog box appears.
5. Enter and browse to the folder's path.

6. 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 have been added to the source path are marked in the Navigator view.

Adding project references

Projects within a workspace can refer to other projects. You might do this if, for example, you separate business logic or interfaces into ActionScript projects and you refer to these projects from a Flex project that contains your application's user interface. When you refer to other projects, dependencies are created that affect how your application is compiled. The Flex Builder compiler manages these dependencies for you and compiles each project in the proper order so that your application is compiled successfully.

To refer to other projects:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) to display the context menu and select Properties > Project References.
3. All of the projects in the workspace are listed. Select the projects to which you want to create a reference.
4. Click OK.

Viewing resource properties

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

To view resource properties:

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

About ActionScript projects

Flex Builder allows you to 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 now 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 also use all the debugging tools.

When you create an ActionScript project or a stand-alone ActionScript file to contain functions or 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

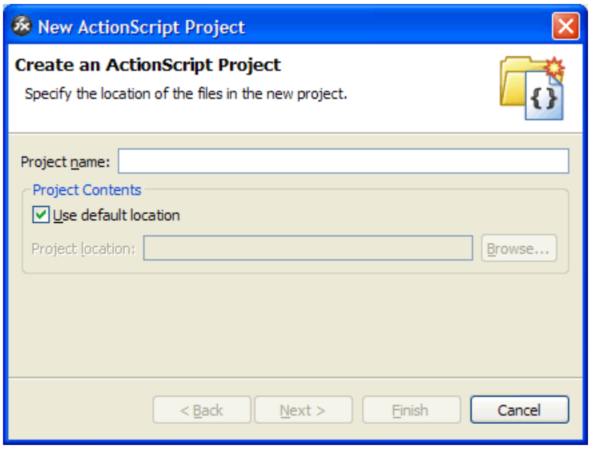
When you create an ActionScript project, the New ActionScript Project wizard guides you through the steps, prompting you for the type of project you want to create, the project's name, location, and other advanced options.

To create an ActionScript project:

1. Select File > New > ActionScript Project.

   You can also create a project by right-clicking (Control-clicking on Macintosh) in the Navigator view and selecting New > ActionScript Project from the context menu. Or, use the keyboard shortcut Alt+Shift+N (Option+Shift+N on Macintosh) to display the New menu and then select ActionScript Project.

The New ActionScript Project wizard appears:



2. Specify the following:

   Project Name is the name of your project.

   Project Location is the location of your project. The default location is the workspace, which is by default: 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 Macintosh, the default workspace location is /Users/*Flex Developer*/Flex Builder 3/ *project_name*.

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

   Source path is used 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 is used to link to 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 is 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 is 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 if you like.

Output folder is the location of the compiled application files. By default, this is the 'bin' folder but you can change this if you like.

Output folder URL is used to specify the server location of the compiled application files. This is optional.

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

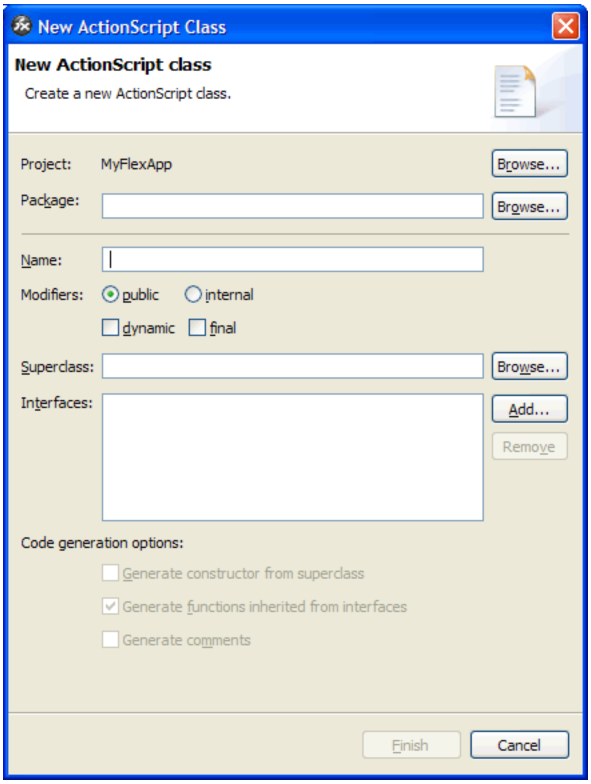
Creating an ActionScript class

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.

To create an ActionScript class:

1. Select File > New > ActionScript Class.

   The New ActionScript Class dialog box appears:



New ActionScript Class

New ActionScript class

Create a new ActionScript class.

Project: MyFlexApp

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.

   For instructions, see "Setting the New ActionScript Class dialog box options" in Help.

   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 "Adding components visually" on page 108.

3. Write the definition of your ActionScript class.

   For more information, see Chapter 9, "Creating Simple Visual Components in ActionScript" in *Creating and Extending Flex Components*.

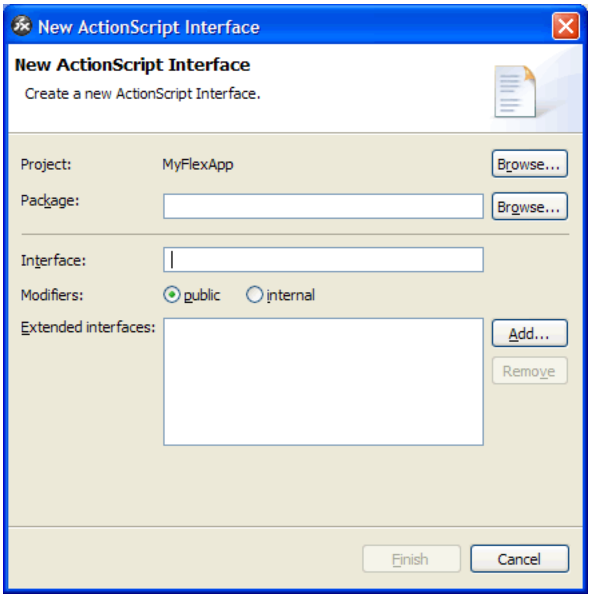
Creating an ActionScript interface

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.

To create an ActionScript interface:

1. Select File > New > ActionScript Interface.

   The New ActionScript Interface dialog box appears:



New ActionScript Interface

New ActionScript Interface

Create a new ActionScript Interface.

Project: MyFlexApp

Package:

Interface:

Modifiers: [✓ public] internal

Extended interfaces:

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

   For instructions, see "Setting the New ActionScript Interface dialog box options" in Help.

3. Add any constants or methods to your ActionScript interface that you want different classes to share.

Related topics

- "Creating an ActionScript class" on page 76

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: Project > Properties > Flex Build Path).

Archived into a SWC file is a SWF file containing 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 allows you to 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" in *Flex Developer's Guide*.

Libraries are useful if you create components entirely in ActionScript and want to use them in Design mode in Flex Builder. ActionScript components are not visually rendered in Design mode until they have been 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 will visually render in Design mode when added to the application.

Configuring libraries for use in your applications

You can 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 with the library are available to use in your application. When you build the application, only the library components you've actually used are compiled into the application SWF file. In other words, all 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 choose to 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's used by the application, but does not merge it into application SWF file. The advantage to 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.

Run-time shared library In Flex projects only, SWC files can also be used as a run-time shared library (RSL), which is similar to a dynamically-linked library on other platforms. You 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 using an RSL. First, the library is loaded into memory once, is cached, and is 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 Chapter 11, "Using Runtime Shared Libraries" in *Building and Deploying Flex Applications*.

The following topics explain how to create and use SWC libraries in Flex Builder:

- "Creating Flex component library files" on page 80
- "Using SWC files in your projects" on page 82

Creating Flex component library files

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.

Create the Flex library project

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

To create a Flex library project:

1. Select File > New > Flex Library Project.

   The New Flex Library Project wizard appears.

2. You are prompted to specify the following:

   Project Name is the name of your library project.

   Project Location is the location of your library project. The default location is the workspace, which is by default: 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.*

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 that you want to include in the SWC file. You can also add other projects, folder, or library SWC files that you want to include in your library project. See "Using SWC files in your projects" on page 82.
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 70.)
- Add a linked folder containing components to the library project's source path. (See "Deleting folders and files" on page 67.)

NOTE: All the components you want to 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) that you want to include in the SWC when it is built by the compiler.

To select library project elements to include in the SWC file:

1. Select Project > Properties > Flex Library Build Path.
2. The components that you've added to the project (either directly or by linking to them) appear in the Classes tab. Select the component classes that you want to include in the SWC file.
3. (Optional) Select the Resources tab and then select resources that you want included in the SWC file.
4. After you've made your selections, click OK.

Building library projects

After you select elements to include in the SWC file, and if you have the Build Automatically option selected, 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.

Building your library project generates a SWC file, which you can share with other applications or users.

For more information, see the next section.

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 within the project, within 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.

To add a SWC file to the library path:

1. With a project selected in the Navigator view, select Project > Properties > Flex Build Path.
2. Select the Library tab.

   The core library (SWC) files, such as frameworks.swc and playerglobal.swc, appear.
3. Select any of the three options you have for adding SWC files:

   Add SWC adds a compiled SWC file.

   Add SWC Project adds a SWC library project.

   Add SWC Folder allows you to add a folder containing SWC files.
4. Enter or browse to and select the location of the SWC file, project, or folder.
5. Click OK. The SWC file, library project, or folder is added to the library path.

To merge the SWC file into the application SWF file when compiled:

1. With a project selected in the Navigator view, select Project > Properties > Flex Build Path.
2. Select the Library 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.

To set the SWC file as an external library file:

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

2. Select the Library 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.

To use the SWC file as an RSL:

1. With a project selected in the Navigator view, select Project > Properties > Flex Build Path.
2. Select the Library 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) If you want to extract the SWF file in the SWC file when it is placed in the deploy location, select the Auto Extract SWF to RSL URL option.
7. Click OK.

This procedure simplifies the process for creating RSLs manually. To do this, you extract the SWF file from the SWC file and set the values of the `runtime-shared-library` and `external-library-path` compiler options.

For more information about using SWC files as an RSL, see Chapter 11, “Using Runtime Shared Libraries” in *Building and Deploying Flex Applications.*

Related topics

- About library projects
- Creating Flex component library files

CHAPTER 4

Navigating and Customizing the Flex Builder Workbench

4

This topic shows you how to navigate and customize the Adobe Flex Builder 2 workbench. It also shows you how to perform searches in the workbench, and how to work in Source and Design modes. For an overview of the workbench, including an overview of perspectives, editors, and views, see Chapter 2, “Flex Builder Workbench Basics,” on page 25.

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

You may find more in-depth information about some of the Eclipse workbench features by referring to the Eclipse Workbench User’s Guide, at http://help.eclipse.org/help31/index.jsp.

This topic contains the following sections:

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 Debug perspective to debug your Flex application.

For an overview of perspectives, see “About Flex Builder perspectives” on page 30.

This section contains the following topics:

Opening 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.

To open a perspective manually:

1. Select Window > Perspective and choose either the Flex Debugging or Development 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 Open Perspective in the upper-right corner of the workbench window.

2. Select the perspective to open from the pop-up menu that appears.

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

When the perspective opens, the perspective title changes to display the name of the perspective you selected. Additionally, 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. For more information, see "Opening perspectives in a new window" on page 87.

Switching between perspectives

When a perspective is open in Flex Builder, an icon for that perspective appears in the upper-right corner of the main toolbar. The standalone configuration of Flex Builder contains two perspectives: the Flex Development perspective and the Flex Debugging perspective. If you are using the plug-in configuration of Flex Builder, you may have several other perspectives available.

To switch between perspectives:

- Click the perspective icon in the toolbar.

Setting the default perspective

You can see what your default perspective is by selecting Window > Perspective > Other (Window > Open Perspective > Other in the plug-in configuration of Flex Builder). The default perspective is indicated by the word *default* in parentheses following the perspective name.

To change the default perspective:

1. Select Window > Preferences.
2. Click the plus button to expand the General category and select Perspectives.
3. Under Available Perspectives, select the perspective to define as the default, and click Make Default.
4. 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.

To open perspectives in a new window:

1. Select Window > Preferences.
2. Click the plus button to expand the General category and select Perspectives.
3. Under Open a New Perspective, select In a New Window.
4. To switch back to the default, select In the Same Window.
5. Click OK.

Creating a customized perspective

You can change the editors and views that are visible in a given perspective to modify a perspective's layout. For example, you might want 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 > Open Perspective submenu, the Window > Show View submenu, and action sets (buttons and menu options) that appear in the toolbar and in the main menu items.

For more information about customizing the workbench in general, see "Customizing the workbench" on page 95.

To modify a perspective's layout:

1. Open the perspective to modify.
2. Show views and editors as desired.

   For more information, see "Opening views" on page 90, and "Opening files for editing" on page 93.
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 and click OK.

To 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 you've previously defined yourself. You can't delete a perspective you didn't create.

To delete a custom perspective:

1. Select Window > Preferences.
2. Click the plus button to expand the General category and select Perspectives.

3. Under Available Perspectives, select the perspective to delete.
4. Click Delete, and then click OK to close the Preferences dialog box.

Resetting perspectives

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

To reset a perspective:

1. Select Window > Preferences.
2. Click the plus button to expand the General category and select Perspectives.
3. Under Available perspectives, select the perspective to reset.
4. Click Reset, and then click OK to close the Preferences dialog box.

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 "Flex Builder basics" on page 25.

This section contains the following topics:

Opening views

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

To open a view:

- 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 "Creating a customized perspective" on page 87.

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 91.

Moving and docking views

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

To change the location of a view:

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

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

   TIP 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's tab, select Move > View, move the view to the desired location, and click the mouse button again.

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

To rearrange tabbed 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.

To switch between views:

- 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's toolbar), the view becomes hidden again.

NOTE: If you convert the Navigator view to a fast view, and then open a file from the Navigator fast view, the fast view automatically is hidden to allow you to work with that file.

To 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.

To 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.

   TIP: You can also follow these steps to create fast views.

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.

For more information see "The Tasks view" on page 41 and "The Problems view" on page 33.

To filter the Tasks or Problems views:

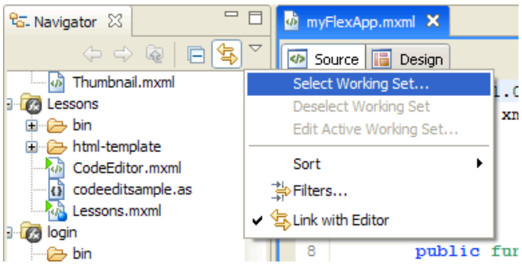
1. In the toolbar of the Tasks or Problems view, click Filter.
2. Complete the Filters dialog box and click OK.

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

To create a working set:

1. In the Navigator view, open the toolbar menu and select Select Working Set. The Select Working Set dialog box appears:



2. Select New and the New Working Set dialog box appears. 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. When you're done selecting projects, click Finish.

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

To display all projects in the workspace:

- In the 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.

To open a file for editing:

- 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.

To associate editors with file types:

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.

TIP

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.

To 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.

TIP

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.

To tile editors:

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 mouse 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 or 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.

To maximize a view or editor:

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

To 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.

TIP 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.

This section contains the following topics:

- "Rearranging the main toolbar" on page 95
- "Changing keyboard shortcuts" on page 96
- "Changing fonts and colors" on page 96
- "Changing the placement of tabs" on page 98
- "Controlling single- and double-click behavior" on page 99

Rearranging the main toolbar

Flex Builder lets you rearrange sections of the main toolbar. Sections of the main toolbar are divided by a space.

To rearrange sections of the main toolbar:

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.

TIP 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

To customize keyboard shortcuts:

1. Select Window > Preferences.
2. Click the plus sign to expand the General category, and select Keys.
3. In the View screen of the Keys dialog box, select the command for which to customize a keyboard shortcut, and then click Edit.
4. On the Modify tab, make your changes and click 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.

This section contains the following topics:

- "About changing fonts" on page 96
- "Changing fonts" on page 97
- "Changing colors" on page 98

About changing fonts

The workbench lets you configure the following fonts:

Banner font is the font that appears in the title area of many wizards. For example, the New Project wizard uses the Banner font for the top title.

Dialog font is the font that appears in widgets and dialog boxes.

Header font is the font that appears as a section heading.

Text font is the font that appears in text editors.

CVS Console font is the font that appears in the CVS console.

Ignored Resource font is the font that displays resources that CVS ignores.

Outgoing Change font is the font that displays outgoing changes in CVS.

Console font (defaults to text font) is the font that appears in the debug console.

Detail Pane Text font (defaults to text font) is the font that appears in the detail panes of debug views.

Memory View Table font (defaults to text font) is the font that appears in the table of the memory view.

Java Editor Text font (defaults to text font) is the font that appears in Java editors.

Properties File Editor Text font (defaults to text font) is the font that appears in Properties File editors.

Compare Text font (defaults to text font) is the font that appears in textual compare or merge tools.

Java Compare Text font (defaults to text font) is the font that appears in Java compare or merge tools.

Java Properties File Compare Text font (defaults to properties file editor text font) is the font that appears in Java properties file compare or merge tools.

Part Title font (defaults to properties file editor text font) is the font that appears in view and editor titles.

NOTE Don't use bold or italic for the Part Title font, because the workbench uses bold and italic versions of this font to display progress.

View Message font (defaults to properties file editor text font) is the font that 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 (Window> Preferences > 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 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

To change fonts:

1. Select Window > Preferences.
2. Click the plus button to expand the General category, click the plus button to expand the Appearance category, and select Colors and Fonts.

3. Expand the Basic, CVS, Debug, Text Compare, or View and Editor Folders categories to locate and select the font to change.
4. Click 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.

5. Set font preferences as desired.

Changing colors

The workbench uses colors to distinguish different elements, such as error text and hyperlink text.

To change colors:

1. Select Window > Preferences.
2. Click the plus button to expand the General category, click the plus button to expand the Appearance category, and select Colors and Fonts.
3. Expand the Basic, CVS, Debug, Text Compare, or View and Editor Folders categories to locate and select the color to change.
4. Click the color bar to the right to open the color picker.
5. Select a new color.

In general, 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).

Changing the placement of tabs

Flex Builder lets you change the placement of view and editor tabs. Tabs for open views and editors can appear at the upper or lower edge of the corresponding view or editor.

To change the placement of tabs:

1. Select Window > Preferences.
2. Click the plus button to expand the General category and select Appearance.
3. Complete the dialog box and click OK.

Controlling single- and double-click behavior

You can control how the workbench responds to single and double clicks.

To control click behavior:

1. Select Window > Preferences.
2. Select the General category.
3. Complete the dialog box and click OK.

Searching in the workbench

Flex Builder provides a search tool that lets you quickly locate resources. This section provides an overview of how to use search in the workbench. For more information about searching for text in a particular file, see "Finding and replacing text in the editor" on page 181.

This section contains the following topics:

- "Searching for files" on page 99
- "Using the Search view" on page 100

Searching for files

Flex Builder lets you conduct complex searches for files. For example, you can search for all files that contain a particular set of characters, or files that end with .xml.

To search for files:

1. Do one of the following

- Click the Search button on the main toolbar.
- Select Search > File.

2. In the Containing Text text box, type the text to search for.

   For example, to search for a file called revenue2006.html, you might type **revenue**.

3. Select Case Sensitive to perform a case sensitive search.
4. Select Regular Expression to enable searches that include regular expressions as part of the search parameter.

   You can use the following wildcards:

   - * matches any set of characters, including the empty string
   - ? matches for any character
   - \ is the escape for a literal

- If you want to search for an asterisk, question mark, or backslash character, type a backslash before the character to indicate that you are not using the character as a wildcard (for example, \?).

5. In the File Name Patterns text box, enter the file type to locate (for example *.txt).

   You can click Choose for quick access to a list of registered extensions.
6. Under Scope, select a scope for your search.

   You can elect to search the entire workspace, previously selected resources, projects enclosing the selected resources, or predefined working sets.
7. Click Search.

   The Search view appears and lists the results of your search. You can cancel the search while it's in progress by clicking Cancel.

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

Using the Search view

The Search view displays the results of your search.

To open a file from the list:

- Double-click the file.

To remove a file from the list:

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

To remove all files from the list:

- Click Remove All Matches.

To navigate between matched files:

- Click Show Next Match or Show Previous Match.

To view previous searches:

- Click the down arrow next to Show Previous Searches and select a search from the pull-down list.

To return to the Search view after closing it:

1. Select Window > Other Views > Basic. (Window > Show View > Other in the plug-in configuration of Flex Builder.)
2. Expand the Basic 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.

To view your file in Design mode:

- Click Design at the top of the editor area.

To view your file in Source mode:

- Click Source at the top of the editor area.

To work in both Source and Design modes simultaneously:

1. Right-click (Control-click on Macintosh) 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.

To switch between the Source and Design modes:

- Press Control+’ (Control+’ on Macintosh).

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.

To obtain a list of available keyboard shortcuts:

- 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. For more information, see "Setting project properties" in Help.

To 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 to save your changes.

For more information about setting Flex Builder preferences, including detailed information about the various Preferences dialog boxes, see "Setting running and debugging preferences" in Help.

PART 3

Developing a Flex Application User Interface

3

This part describes how to build a Flex user interface in Flex Builder and includes the use of view states, transitions, behaviors, and custom components.

The following chapters are included:

CHAPTER 5

Building a Flex User Interface

5

You can use Adobe Flex Builder to build the user interface of a Flex application rapidly. The building blocks of user interfaces are MXML containers and controls. A control is a user-interface component such as a Button, TextArea, and ComboBox control. 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.

For more information about components, see Chapter 7, "Using Flex Visual Components" in *Flex Developer's Guide*.

This topic contains the following sections:

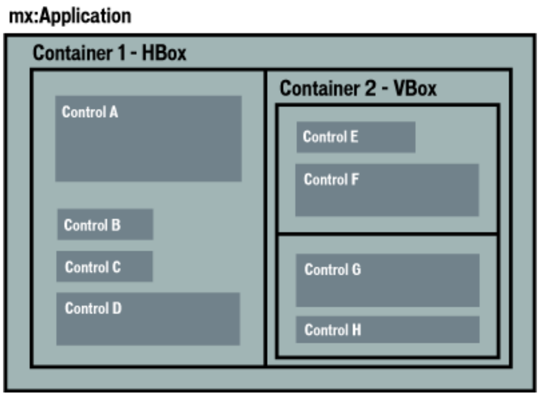
About the structure of Flex user interfaces

The Adobe Flex framework is a component-based system for building rich Internet applications. A Flex application typically consists 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).

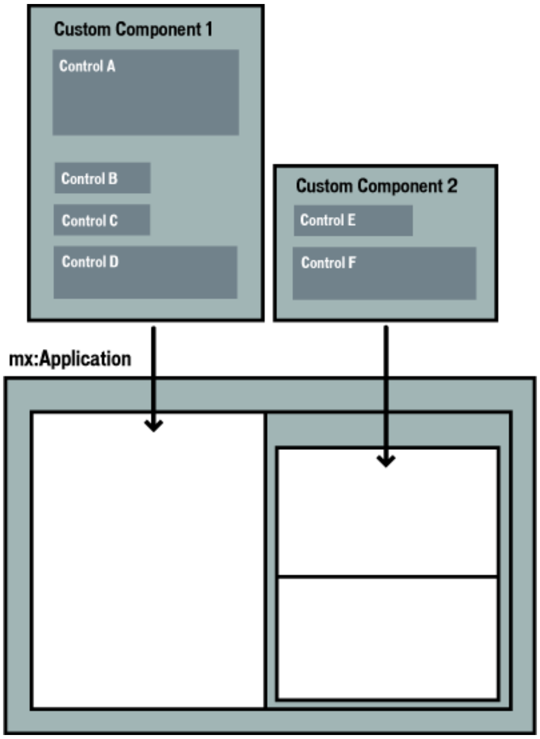
Flex gives you several options for structuring your user interface. You can insert containers and controls directly in the MXML application file, or you can insert them in separate MXML files to create custom components, and then insert the custom components in the application file.

TIP: Generally, it's 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.

Related topics

- Chapter 8, "Creating Custom MXML Components," on page 157
- Chapter 7, "Creating Simple MXML Components" in *Creating and Extending Flex Components*

Adding components

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

This section contains the following topics:

- "Adding components visually" on page 108
- "Adding components by writing code" on page 110
- "Adding Flash components (SWC files)" on page 111

Related topics

- "Adding navigator containers" on page 134
- "Adding data provider controls" on page 138
- "Working with components visually" on page 112

Adding components visually

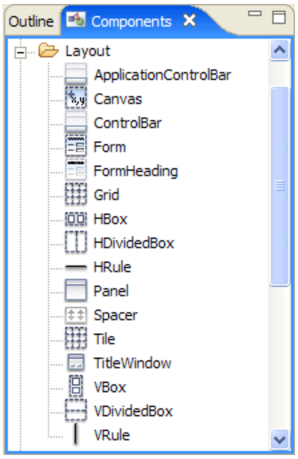
You can use Flex Builder to add standard Flex containers and controls to your user interface visually. You can also add custom components that you define in separate MXML and ActionScript files and save in the current project or in the source path of the current project.

To add components visually:

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 in the view 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 Chapter 8, "Creating Custom MXML Components," on page 157.

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.

To add components in complex layouts:

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

   Flex Builder highlights the current target container.
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.

Related topics

- "Adding Flash components (SWC files)" on page 111
- "Working with components visually" on page 112

Adding 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.

To add components using Content Assist:

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

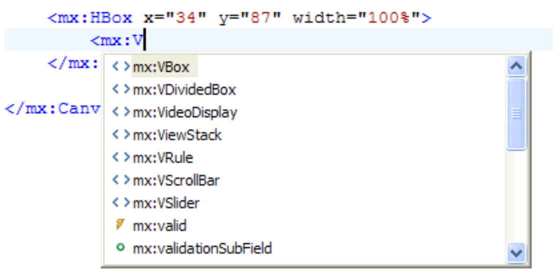
   The MXML file can be the main application file (a file with an Application container) or a custom MXML component file.
2. Place the insertion point in the parent container tag.

   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. Start typing the component tag.

   As you type, a pop-up menu appears suggesting possible entries.
4. If necessary, use the arrow keys to select your tag from the menu, and 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 all the custom components that 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 Chapter 8, "Creating Custom MXML Components," on page 157.

Related topics

- "Using Content Assist" on page 172
- "Adding components visually" on page 108

Adding Flash components (SWC files)

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

NOTE You can use SWC files created in Flash 8 only for skinning, not as components. Until the next version of Flash is released, you can only use SWC files created in Flex 2 as components.

To add a Flash component:

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. To set or learn the library path, select the project in the 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 197.

   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, begin typing the component tag and then use Content Assist to quickly complete the tag.

Related topics

- "About SWC files" in *Building and Deploying Flex Applications*

Working with components visually

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

This section contains the following topics:

Related topics

Using the MXML editor in Design mode

The MXML editor has two modes: Source mode for writing code, and Design mode for developing applications visually.

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

To view an MXML file in Design mode:

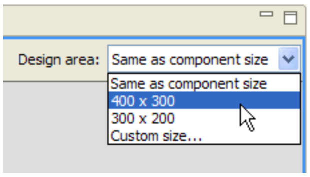
1. If the MXML file is not already open in the MXML editor, double-click the file in the Navigator view to open it.
2. If the MXML editor displays source code, click Design at the top of the editor area.

   You can quickly switch between modes by pressing Control+' (Command+' on Macintosh).

   Switching between Source and Design modes automatically shows or hides design-related views like the Components, Properties, and States views. You can turn this behavior on and off by selecting Window > Preferences, navigating the Flex > Editors > MXML Editor branch in the Preferences tree control, and then toggling the Automatically Show Design-related Views option.

To set the size of the design area:

- 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.

To select, move, and resize items in the design area:

- Click on the Select Mode button on the editor toolbar or go to Design > Select Mode from the main menu. Select Mode is activated by default when you open a document. Press V on the keyboard to enter Select Mode.

To grab and scroll around the design area:

- Click on the Pan Mode button on the editor toolbar or select Design > Pan Mode from the main menu. You cannot select or move items in Pan Mode. Press H to enter Pan Mode from the keyboard. To temporarily enter Pan Mode press the spacebar on the keyboard.

To magnify items in the design area:

There are several ways to use the zoom tool. You can select percentages from the main and pulldown 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 Mode or Zoom In (Ctrl+=)or Zoom Out (Ctrl+- or Alt-click). You can also select the Magnification submenu and choose a specific percentage. The design area changes to your selection.
- Click on 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 pulldown menu next to the Select, Pan, and Zoom Mode buttons on the editor toolbar. The design area changes to the selected percentage or fit to 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 Zoom In (Ctrl+-) or Zoom Out (Ctrl+= or Alt-click) keyboard shortcuts from the design area.

For more keyboard shortcuts see 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.

To select multiple components in the MXML editor's Design mode:

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

To deselect multiple components:

- Do one of the following:
  - Click the background container.
  - Click an unselected component.
  - Click in the gray margin around the root component.

Related topics

- “Showing surrounding containers” on page 125

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 132.

To change the position of a component, do one of the following:

- 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’s a Canvas container or if it’s 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 Chapter 17, “Using Layout Containers” in *Flex Developer’s Guide*.
- 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.

Related topics

- “Using snapping to position components” on page 117
- “Aligning components” on page 118
- “Nudging components” on page 119

Sizing components

You can change the size of a component in an MXML file visually with your mouse, by selecting menu options, or by editing its properties in the Flex Properties view.

You can also dynamically size components by using a constraint-based layout, where you anchor one or more sides of a component to the edges of the component's container or the container's constraint regions. For more information, see “About constraint-based layouts” on page 128.

To size a component visually:

- In the MXML editor's Design mode, click one of the resize handles on the component and drag the handle to resize the component.

  To constrain the proportions of the component, hold down the Shift key while dragging.

  As you resize, snap lines appear to line up the edges of the component with nearby components. To turn off this behavior, select Design > Enable Snapping from the menu.

To make two or more components the same width or height:

1. In Design mode, select two or more components.

   For more information, see “Selecting multiple components in an MXML file” on page 114.
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.

To size a component by editing its properties:

1. Select the component in the Design mode.

   You can Control-click 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.

Related topics

- "Positioning components" on page 115
- "Setting component properties" on page 120

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

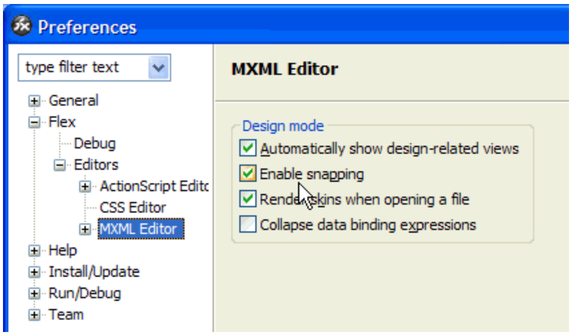
You can disable snapping for one component or for all components.

To disable snapping:

- With the MXML file open in the MXML editor's Design mode, select Design > Enable Snapping.

To enable or disable snapping as a preference:

1. Open the Preferences dialog box by selecting Window > Preferences.
2. Open the MXML editor preferences by selecting Flex > Editors > MXML Editor in the sidebar of the dialog box.



MXML Editor

Design mode

[✓] Automatically show design-related views
[✓] Enable snapping
[✓] Render skins when opening a file
[ ] Collapse data binding expressions

3. Select or deselect the Enable Snapping option in the preferences.

Related topics

- "Positioning components" on page 115
- "Nudging components" on page 119

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 allows you to 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 132.

To 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 114.

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.

Related topics

- "Positioning components" on page 115

Nudging components

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

This feature is not available on Macintosh.

To nudge components by one pixel:

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

To nudge components by ten pixels:

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

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

Setting component properties

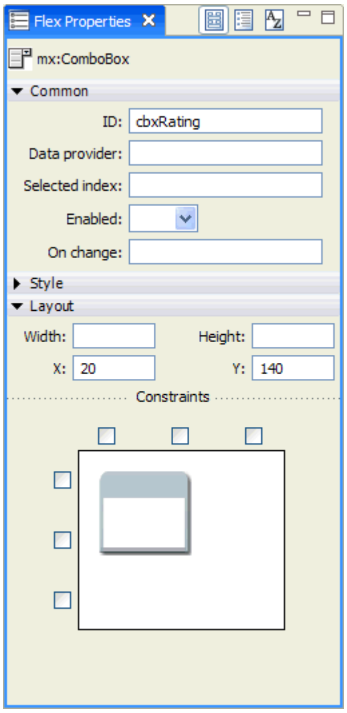
You can visually set the properties of components, including custom MXML components.

To edit the text displayed by a component:

- To edit text displayed by a component like a Label or TextInput control, double-click the component and make your edits.

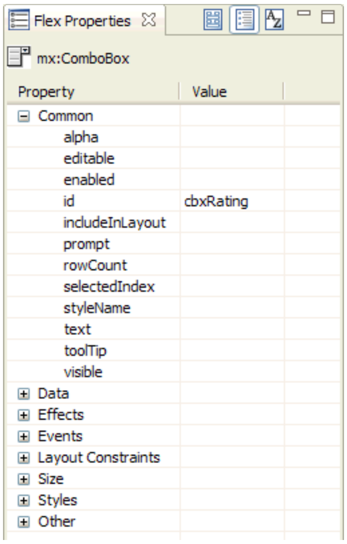
To set other properties of a component:

- In Design mode, select the component and set its properties in the Flex Properties view (Window > Flex Properties).



ID: cbxRating
Data provider:
Selected index:
Enabled:
On change:
Width:
Height:
X: 20
Y: 140

You can also set the properties in a categorized or alphabetical table view by clicking the view buttons in the toolbar:



Property	Value
⊟ Common	
alpha	
editable	
enabled	
id	cbxRating
includeInLayout	
prompt	
rowCount	
selectedIndex	
styleName	
text	
toolTip	
visible	
⊞ Data	
⊞ Effects	
⊞ Events	
⊞ Layout Constraints	
⊞ Size	
⊞ Styles	
⊞ Other	

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

Applying CSS styles to components

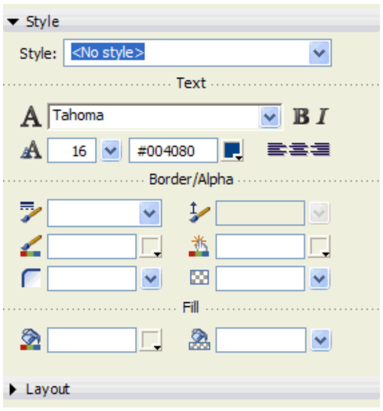
You can use CSS styles to define, maintain, and easily modify the general appearance of components. CSS styles in Flex can be defined at the site level with external styles, at the document level with embedded styles, or at the component level with inline styles. For more information, see Chapter 20, "Using Styles and Themes" in *Flex Developer's Guide*.

You can use Flex Builder to visually apply inline styles to a component. An example of an inline style is `<mx:Button color="red"...>`. The color property in this Button tag applies only to this specific Button instance.

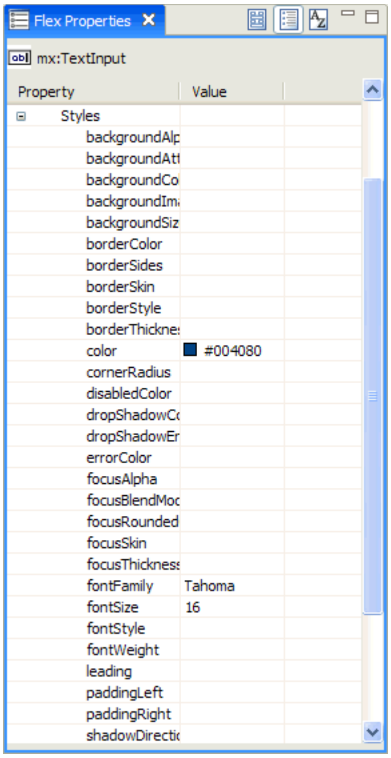
To apply an inline style to a component:

1. Click the component in the MXML editor's Design mode to select it.
2. Enter the style property values in the Flex Properties view.

In the Flex Properties view, you can set the styles in the Standard, Category, or Alphabetical view of properties. The Standard view provides a form to set common styles for the selected component.



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



Flex Properties

mx:TextInput

Property	Value
Styles	
backgroundAlp	
backgroundAtt	
backgroundCo	
backgroundIm	
backgroundSiz	
borderColor	
borderSides	
borderSkin	
borderStyle	
borderThickne	
color	■ #004080
cornerRadius	
disabledColor	
dropShadowCo	
dropShadowEr	
errorColor	
focusAlpha	
focusBlendMoc	
focusRounded	
focusSkin	
focusThickness	
fontFamily	Tahoma
fontSize	16
fontStyle	
fontWeight	
leading	
paddingLeft	
paddingRight	
shadowDirecti	

NOTE Multiword style names in Flex appear slightly different than their HTML counterparts, but they're the same styles. For example, fontFamily in Flex is the same style as font-family in HTML.

To apply an external or embedded style to a component:

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:

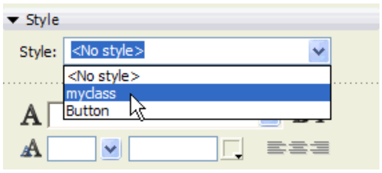
   ```
   <mx:Style source="styles.css"/>
   ```

   The following expression embeds two styles in the MXML file:

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

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.



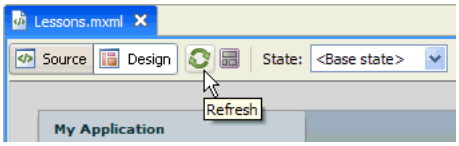
The Style pop-up menu lists the styles defined in the external style sheet or embedded in the current file.

Refreshing Design mode to render properly

If necessary, you can refresh the MXML editor's 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.

To refresh Design mode to render your layout properly:

- Click the Refresh button in the editor toolbar.



Showing surrounding containers

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

To show surrounding containers:

1. Select a container in the layout.

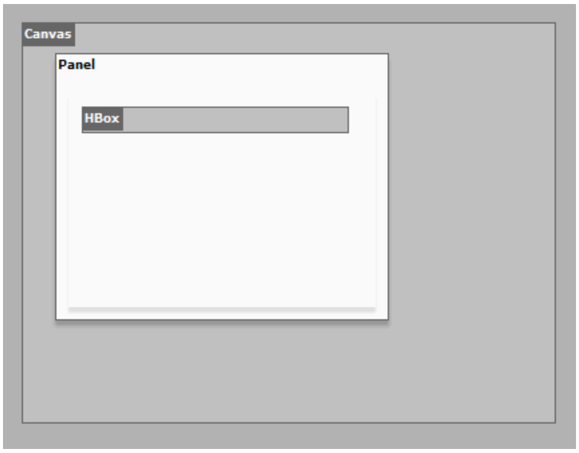
   If it's too difficult to select a container, select a control inside it instead.

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's 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.

To show a container's parents and immediate children:

- With surrounding containers turned on, click the container.

To 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.

To dismiss the surrounding containers:

- Press the F4 key again.

Related topics

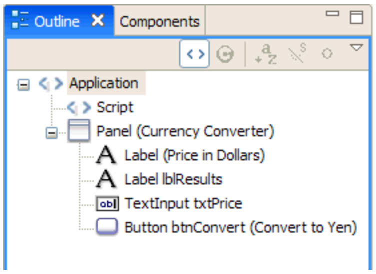
- "Selecting multiple components in an MXML file" on page 114
- "Hiding container borders" on page 127

Inspecting the structure of your MXML

You can 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.

To view the structure of the MXML:

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



To select components in the structure:

- 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.

To hide the borders around containers:

- 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.

To copy a component to another MXML file:

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.

To delete a component in the MXML editor's Design mode:

- 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.

Laying out your user interface

To lay out a Flex user interface, you typically start by inserting and positioning components in a container that has absolute positioning, and then defining layout constraints for the components so that they adjust automatically when a user resizes the application window.

In some situations, using nested layout containers such as the VBox containers is preferable to using absolute positioning. See Chapter 17, "Using Layout Containers" in *Flex Developer's Guide*. These situations include the following:

- 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.

This section contains the following topics:

- "About constraint-based layouts" on page 128
- "Inserting and positioning components in the layout" on page 131
- "Setting layout constraints for components" on page 132

Related topics

- "Adding navigator containers" on page 134
- Chapter 6, "Adding View States and Transitions," on page 143

About constraint-based layouts

Flex supports constraint-based layouts. You can arrange components in your user interface with the freedom and pixel-point accuracy of absolute positioning, while also setting constraints that stretch or move the components when the user resizes the application window.

You could achieve the same goal with nested layout containers (see Chapter 17, "Using Layout Containers" in *Flex Developer's Guide*), but a constraint-based layout provides a simpler means to dynamically control the size and position of components.

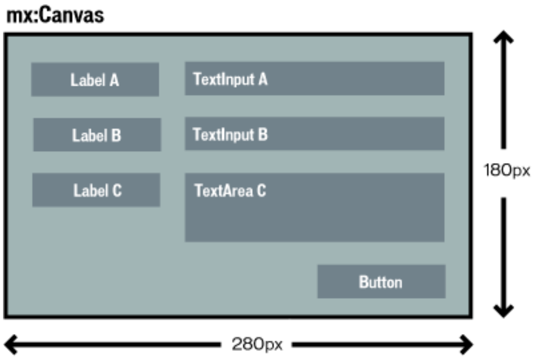
To create a constraint-based layout, you must use a container that has absolute positioning. The `layout="absolute"` overrides the container's layout rule and allows you to drag and position components anywhere in the container. This property can only be used with the `Application`, `Canvas`, and `Panel` 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 is set by the width of the container.

NOTE: In Flex, all constraints are set relative to the edges of the container or a constraint region. They cannot be set relative to other controls.

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:

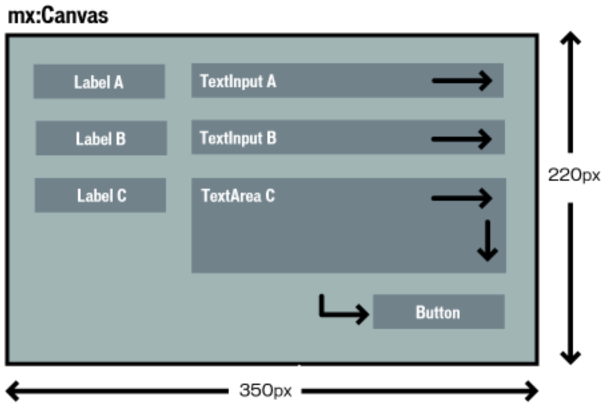


Also, a number of layout constraints are applied to the controls to ensure that the layout adjusts correctly when the user resizes the application:

- 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 illustration shows how the constraints make the controls behave when the user resizes the layout:



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

Row and Column Constraints

You may also define a grid of horizontal and vertical constraint regions, `ConstraintColumn` regions and `ConstraintRow` regions. Components can be constrainted 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 contrainer 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.

The following example illustrates a `Canvas` container with a label, a list control, and two buttons in a `HorizontalBox`. The `Canvas` is partitioned into three columns by dragging `ConstraintColumns` into the `Canvas` container and setting the their dimensions in the Flex Properties view. The controls are positioned and constrained by dragging them into the constraint regions and setting their constraint values in the Flex Properties > Layout > Constraints view.

- The first `ConstraintRow` is fixed at 50 pixels high, the second row is 60% of the vertical space in the parent container (after both fixed height constraints are allocated), and third row is 40 pixels. The second row also has a minimum height constraint so the Listbox will not shrink smaller than
- The `Label` is constrained 10 pixels from the top of the `ConstraintRow` and left edges of the `Canvas`.
- The `Listbox` is constrained vertically within the second ConstraintRow. By anchoring the `Listbox` to the top and bottom of the ConstraintRow, it will stretch as the row is resized. The `Listbox` is anchored between the left edge of the first ConstraintColumn and the right edge of the third ConstraintColumn.
- The HorizontalBox is anchored to the right edge of the Canvas and the bottom edge of the third ConstraintRow.

Related topics

- Chapter 10, "Create a Constraint-based Layout" in *Getting Started with Flex*
- "Setting layout constraints for components" on page 132

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.

To insert and position components in the layout:

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

   For more information, see "About constraint-based layouts" on page 128.
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 dragging it or by setting its `x` and `y` properties in the Flex Properties view (Window > Flex Properties).

   For more information, see the following topics:
   - "Positioning components" on page 115
   - "Using snapping to position components" on page 117
   - "Aligning components" on page 118
   - "Nudging components" on page 119

- "Setting component properties" on page 120

4. Set layout constraints for the component.

   For instructions, see the following section.

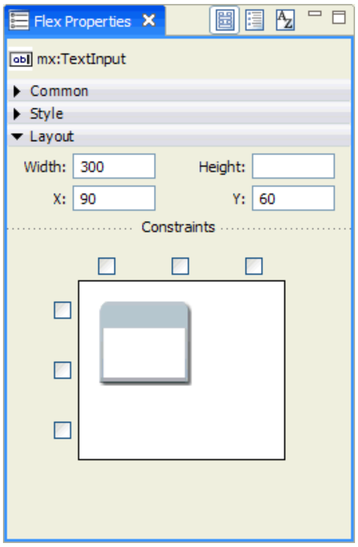
Setting layout constraints for components

You can use Flex Builder to set the layout constraints of containers, controls, and custom components positioned in a container that has absolute positioning.

To set the layout constraints of components:

1. In the MXML editor's Design mode, select the component positioned in the container that has absolute positioning.
2. In the Properties view, expand the Layout category.

   Options for setting constraints appear.



Width: 300
Height:
X: 90
Y: 60

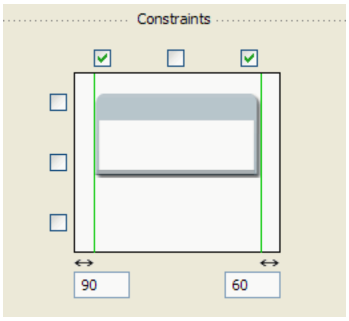
   If you see a list of properties instead of a form, click the Standard View button in the view's toolbar.

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.

To set the distance, enter a number of pixels in the text box that appears near a selected constraint in the Flex Properties view. In the following example, you 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. In the following example, the left and right edges of the component are anchored to the edges of the ConstraintColumn identified by `id="col1"`.

If you're defining a TextInput control, the MXML code for this example is:

```
<mx:TextInput left="col1:90" right="col1:60"/>
```

Related topics

- Chapter 10, "Create a Constraint-based Layout" in *Getting Started with Flex*
- "About constraint-based layouts" on page 128
- "Constraint-based layout" in *Flex Developer's Guide*

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 lets 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 143.

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.

This section contains the following topics:

- "Creating layouts in navigator containers" on page 135
- "Letting users select a view in a ViewStack container" on page 136

Related topics

- "About navigator containers" in *Flex Developer's Guide*

Creating layouts in navigator containers

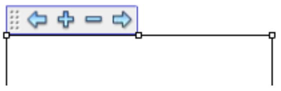
After you insert a navigator container in your application, you can use Flex Builder to create the layout of each of the container's views. The views are child containers of the navigator container.

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.

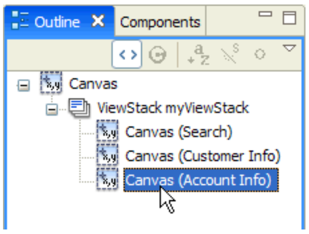
To create layouts in a navigator container:

1. In the MXML editor's Design mode, drag a ViewStack, TabNavigator, or Accordion container from the Components view into the application.
2. Add or remove a child container in the navigator container by clicking the Plus (+) or Minus (-) button at the top of the navigator 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, select a child container by clicking the child's tab.
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.

For more information, see "Laying out your user interface" on page 128.

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

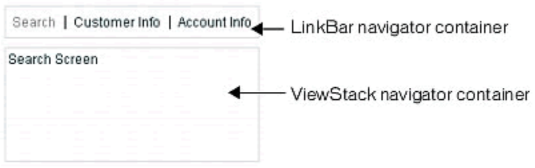
   For instructions, see "Letting users select a view in a ViewStack container" on page 136.

Related topics

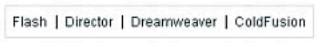
- "ViewStack navigator container" in *Flex Developer's Guide*
- "TabNavigator container" in *Flex Developer's Guide*
- "Accordion navigator container" in *Flex Developer's Guide*
- "Adding navigator containers" on page 134

Letting users select a view in a ViewStack container

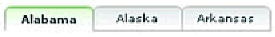
A ViewStack container consists of a collection of child containers stacked on top of each other with only one container visible, or active, at a time. The ViewStack container does not have a built-in mechanism for letting users select a child container. You must add a LinkBar, ButtonBar, or TabBar navigator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



To let users select a child container in a ViewStack container:

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>
```

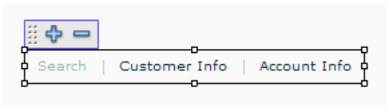
2. Drag a LinkBar, ButtonBar, or TabBar container from the Components view into your layout, above the ViewStack container.

   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.

Related topics

- "ViewStack navigator container" in *Flex Developer's Guide*
- "LinkBar control" in *Flex Developer's Guide*
- "TabBar control" in *Flex Developer's Guide*
- "ButtonBar and ToggleButtonBar controls" in *Flex Developer's Guide*
- "Creating layouts in navigator containers" on page 135

Adding data provider controls

You can 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.

To add a data provider 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 Chapter 13, "Using Data-Driven Controls" in *Flex Developer's 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>
```

Related topics

- Chapter 11, "Use List-based Form Controls" in *Getting Started with Flex*
- "Specifying data providers in MXML applications" in *Flex Developer's Guide*
- Chapter 12, "Working with Data in Flex Builder," on page 221

Adding charting components

You can use Flex Builder to add charting components to display data in your user interface. The charting components in Flex 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 Chapter 47, "Chart Types" in *Flex Developer's Guide*.

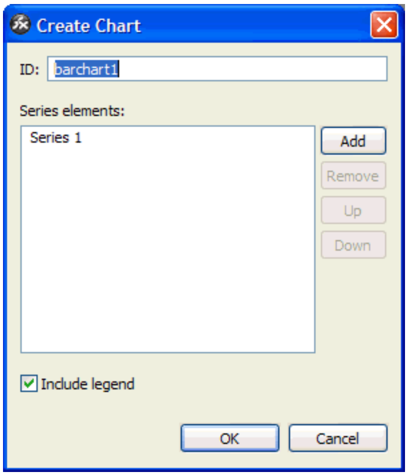
The Flex charts are available in Adobe Flex Builder with Charting. A trial version of the charts is included in the standard version of Flex Builder. You can also obtain the full version of the charts by purchasing Adobe Flex Charting components 2, a separate product.

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 Chapter 46, "Introduction to Charts" in *Flex Developer's Guide*.

To add a charting component:

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

   The Create Chart dialog box appears:

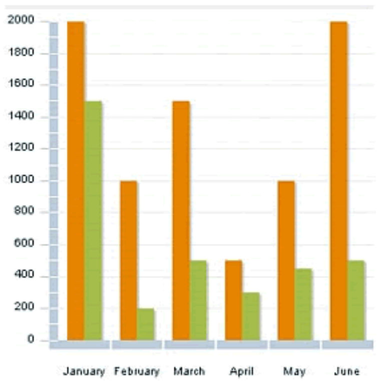
   ID: barchart1

   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 first is the gross profit for six months represented by the orange columns, and the second is the net profit during the same period represented by the green columns.

   Remove a data series by selecting it in the list and clicking the Delete 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 6

Adding View States and Transitions

6

You can use Adobe Flex Builder 2 to create applications that change their appearance depending on the task that the user is performing. For example, the base state of the application could be the home page and include a logo, a sidebar, and some 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.

This topic contains the following sections:

About view states and transitions

A *view state* is one of several layouts that you define for a single MXML application or component. You can 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 Chapter 29, “Using View States” in *Flex Developer’s Guide*.

Generally, you don’t 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 Chapter 18, “Using Navigator Containers” in *Flex Developer’s 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 Chapter 30, “Using Transitions” in *Flex Developer’s Guide*.

Related topics

- Chapter 14, “Use View States and Transitions” in *Getting Started with Flex*
- Chapter 8, “Creating Custom MXML Components,” on page 157

Creating a view state

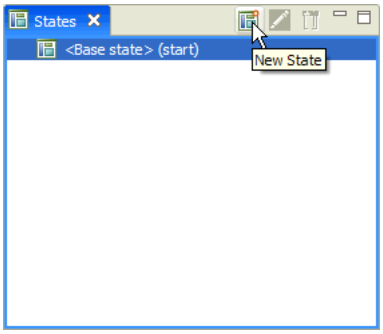
You can use Flex Builder to create a view state for an application or a component defined in an MXML file. First, you create a base state for the application or component, and then you create a second state based on the base state.

To create a state:

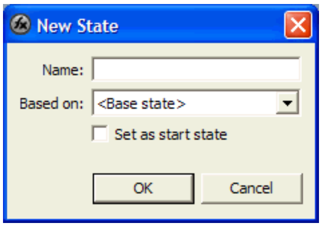
1. Using the layout tools in Flex Builder, design the layout of the base state of your application or component.

   For more information, see "Building a Flex User Interface" on page 105.

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



   The New State dialog box appears. The default option is to base the new state on the initial state.



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 147.

Related topics

- “About view states and transitions” on page 143
- Chapter 14, “Use View States and Transitions” in *Getting Started with Flex*
- Chapter 29, “Using View States” in *Flex Developer’s Guide*

Creating a state based on an existing state

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

To create a state based on another state:

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

   For more information, see “Creating a view state” on page 145.
2. In the States view (Window > States), click the New State button.

   The New State dialog box appears.
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.

   The new state appears in the States view.
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.

   For more information, see “Switching states at run time” on page 147.

Related topics

- “About view states and transitions” on page 143
- Chapter 29, “Using View States” in *Flex Developer’s Guide*.

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.

To set a non-base state as the starting state:

1. With the MXML file opens in the MXML editor's 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.

Related topics

- "Creating a state based on an existing state" on page 146

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.

To set the initial state of a component:

1. With the MXML file open in the MXML editor's 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'"
```

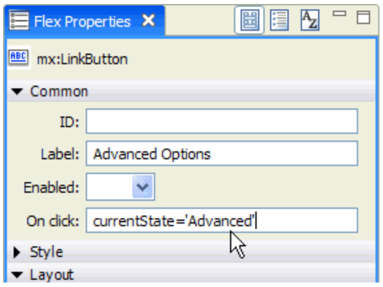
For more information, see "Applying view states" in *Flex Developer's Guide*.

To define an event handler to switch from one state to another at run time:

1. Ensure that the initial state has a control, such as a Button control, that the user can click.

   Select the control in the MXML editor's Design mode, and enter the following value in the On Click text box in the Flex Properties view:

   currentState='*viewstatename*'

   ID:
   Label: Advanced Options
   Enabled:
   On click: currentState='Advanced'

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, run the application by clicking the Run button in the MXML editor's 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 150.

Related topics

- "About view states and transitions" on page 143
- Chapter 14, "Use View States and Transitions" in *Getting Started with Flex*
- Chapter 29, "Using View States" in *Flex Developer's Guide*.

Modifying the appearance of existing states

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

To modify the appearance of an existing state:

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

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

   Flex Builder displays the state's layout in the MXML editor's 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.

Related topics

- "About view states and transitions" on page 143
- Chapter 29, "Using View States" in *Flex Developer's Guide*.

Deleting a view state

You can use Flex Builder to delete a view state.

To delete a state:

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

   Flex Builder displays the state's layout in the MXML editor's Design mode.

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 smoother visually 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.

This section describes the general steps in creating a basic transition.

To create a basic transition:

1. Make sure you create at least one view state in addition to the base state.

   For more information, see "Creating a view state" on page 145.

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 (in bold):

   ```
   <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” in *Flex Developer’s Guide*.

   To set the properties of effects, see “Working with effects” in *Flex Developer’s Guide*.

7. Test the transition by clicking the Run button in the MXML editor’s toolbar, and then switching states after the application starts.

Related topics

- “About view states and transitions” on page 143
- Chapter 14, “Use View States and Transitions” in *Getting Started with Flex*
- Chapter 30, “Using Transitions” in *Flex Developer’s Guide*

CHAPTER 7

Adding Interactivity with Behaviors

7

You can use Adobe Flex Builder 2 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 Chapter 19, "Using Behaviors" in *Flex Developer's Guide*. For a tutorial, see "Use Behaviors" in *Getting Started with Flex*.

This topic contains the following sections:

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 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 Chapter 19, "Using Behaviors" in *Flex Developer's Guide*.

Creating a behavior for a component

You can 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" in *Flex Developer's Guide*.

To create a behavior for an MXML component:

1. Select the component in the MXML editor's Design mode by clicking it in the layout.
2. Define a trigger for the effect by selecting a trigger in the Effects category of the Flex Properties view.

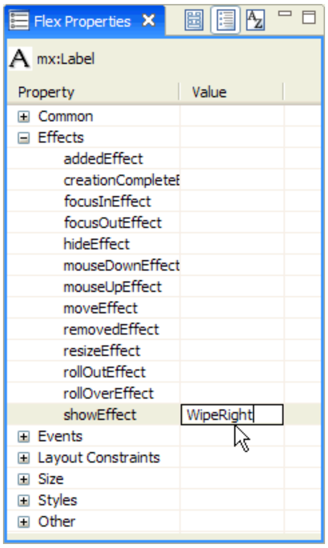
   The names of triggers use the following convention:

   *trigger*Effect

   In this convention, *trigger* is the trigger name. For example, to define a roll-over trigger, select the rollOverEffect trigger.

   For a list of triggers you can use, see "Available triggers" in *Flex Developer's 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.



showEffect WipeRight

For a list of effects you can use, see “Available effects” in *Flex Developer’s 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” in *Flex Developer’s 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” in *Flex Developer’s Guide*.

Related topics

- “Use Behaviors” in *Getting Started with Flex*

CHAPTER 8

Creating Custom MXML Components

8

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 2 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 76.

You can also build an MXML component directly using code. For more information, see Chapter 7, "Creating Simple MXML Components" in *Creating and Extending Flex Components*.

This topic contains the following sections:

About the uses of 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. Finally, you might want to write a completely new kind of component that isn't available in the Flex framework.

If your component is mostly composed of existing components, it's convenient to define it using MXML. However, if it's a completely new kind of component, you should define it as an ActionScript component. For more information, see "Creating an ActionScript class" on page 76.

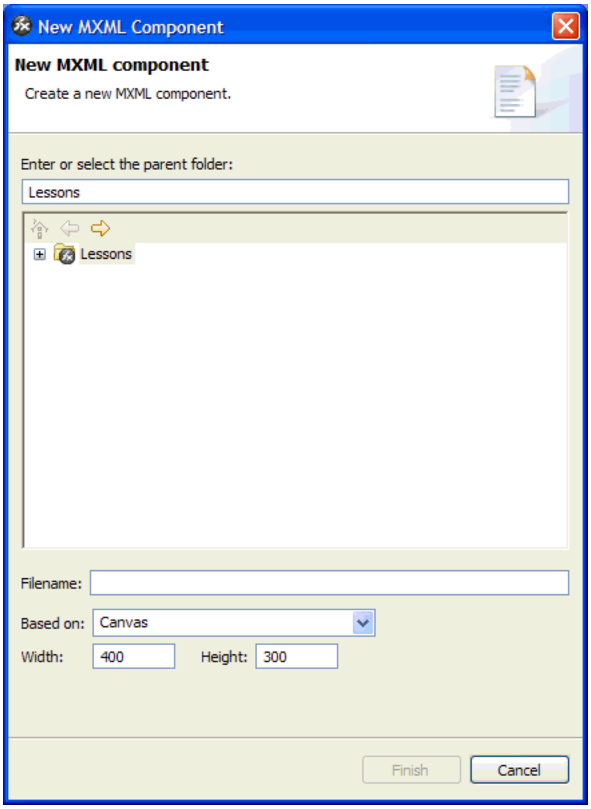
Creating MXML components visually

You can use Flex Builder to create custom MXML components.

To create a custom MXML component:

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, 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 to fixed width and height or percentages, or 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/height or no width/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 160.

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 "Adding components visually" on page 108.

   NOTE The Components view only lists 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 Chapter 7, "Creating Simple MXML Components" in *Creating and Extending Flex Components*.

Related topics

- Chapter 15, “Create a Custom Component” in *Getting Started with Flex*.
- “Creating an ActionScript class” on page 76
- “Creating an ActionScript interface” on page 77

Designing components visually

You can lay out a custom component visually in the MXML editor like 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 112.

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.

Rapidly editing 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.

   TIP 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 78.

PART 4

Programming Flex Applications

4

This part describes how to edit, build, run, and debug applications in Flex Builder.

The following chapters are included:

CHAPTER 9

Code Editing in Flex Builder

9

You can edit MXML, ActionScript, and CSS code in Flex Builder, and a separate editor is provided for each. However, since the Flex Builder workbench is both project- and document-centric, you don't need to concern yourself with selecting editors, because they are associated with resource types and are opened automatically. This topic discusses the shared capabilities of the Flex Builder editors, which includes code hinting, navigation, formatting, refactoring, and other productivity-enhancing features.

This topic does not discuss design and layout of Flex applications using the MXML editor in Design mode. For more information about Design mode, see Chapter 5, "Building a Flex User Interface," on page 105.

This topic contains the following sections:

Understanding code editing in Flex Builder

When designing and developing Flex and ActionScript applications, you work in the Flex Builder development perspective, which contains the Flex Builder editors and all the views that support code editing and design tasks. The configuration of the 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 31.

Here's a brief introduction to the basics of code editing in Flex Builder.

MXML, ActionScript 3.0, and CSS editors

Flex Builder contains three editors: one each for writing MXML, ActionScript, and CSS.

The MXML editor allows you to 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 Chapter 5, "Building a Flex User Interface," on page 105).

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 74.

You use the CSS editor to write Cascading Style Sheets (.css files). For more information, see Chapter 20, "Using Styles and Themes." in *Flex Developer's 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 referred to as Content Assist. Other features that assist you in quickly developing your code include: MXML tag completion, automatic import management, and integration with *Adobe Flex Language Reference*. For more information, see "About Flex Builder content assistance" on page 169.

Custom component and ActionScript class and interface code hints

In addition to the built-in support for the Flex framework, code hints are also provided for all custom components and ActionScript classes and interfaces that are included in your project. For more information, see "About Content Assist" on page 169.

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 173.

Find references and code refactoring

The Flex Builder search feature allows you to find all references to unique identifiers in a given file, project, or workspace (for example, type, variable, function, accessors, metadata). You can then use the refactor feature to rename identifiers in your code while updating all references as appropriate. The Flex Builder refactor feature also enables you to move types (interfaces, classes) and update names and references accordingly. For more information, see "Finding references and refactoring code" on page 181.

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 "About syntax error checking" on page 186.

Editor and debugger integration

The MXML and ActionScript editors work in concert with the 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 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 Chapter 11, "Running and Debugging Applications," on page 205.

Learn more about code editing in Flex Builder

If you haven't already done so, review the getting started lesson. For more information, see Chapter 16, "Use the Code Editor" in *Getting Started with Flex*, and then refer to the following sections in this topic for more information:

- "About Flex Builder content assistance" on page 169
- "Navigating and organizing code" on page 173
- "Formatting code" on page 179

About Flex Builder content assistance

The Flex Builder editors contain many features that simplify and streamline code development. The following content assistance features are available in Flex Builder:

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 169.

Auto import management As you enter code with the help of Content Assist, the fully qualified type names are automatically imported into the document. 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 (see "Organizing import statements" on page 180).

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.

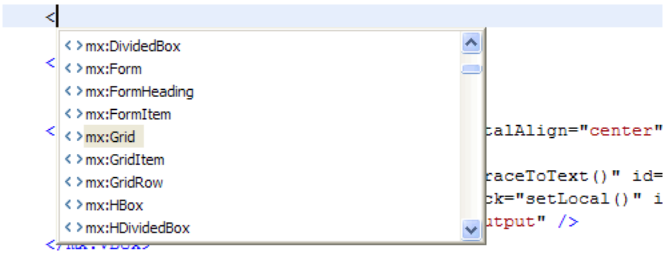
Context-sensitive language reference Help The *Adobe Flex Language Reference* is integrated into the editor and you can easily access it by selecting an ActionScript language element, an MXML component tag or attribute, and pressing F1. For more information, see "Getting help while writing code" on page 173.

Formatting assistance To streamline the work of coding your applications, the editors automatically format and color your lines of code. For more information, see "Formatting code" on page 179.

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, when you begin entering a tag, you are prompted with the list of tags that can be added to the parent tag.

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:



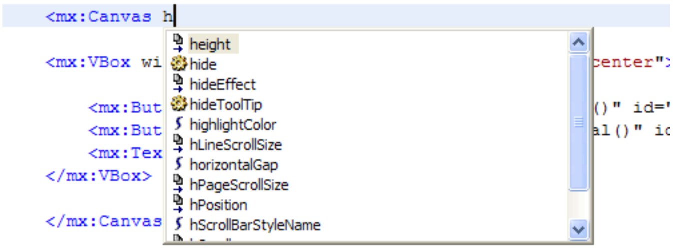
```
<
<                                   alAlign="center"
<                                   aceToText()" id=
                                    k="setLocal()" i
                                    tput" />
```

mx:DividedBox
mx:Form
mx:FormHeading
mx:FormItem
mx:Grid
mx:GridItem
mx:GridRow
mx:HBox
mx:HDividedBox

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.

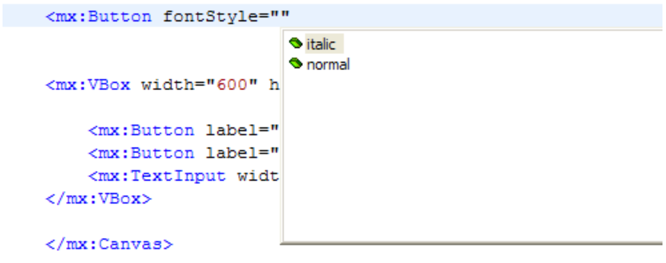
Code hints appear whenever the framework or language (MXML, ActionScript, and CSS) provides options for you to complete the current expression. For example, if you type within an MXML component, you are prompted with a list of all properties of that component. The following example shows code hints for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

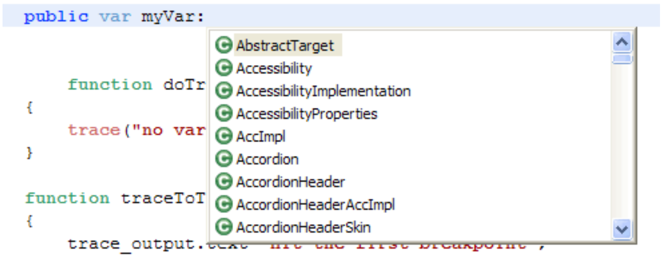
Selecting and entering properties displays possible property values (if predefined values exist). The following example shows code hints for property values:



```
<mx:Button fontStyle=""


<mx:VBox width="600" h

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

</mx:Canvas>
```

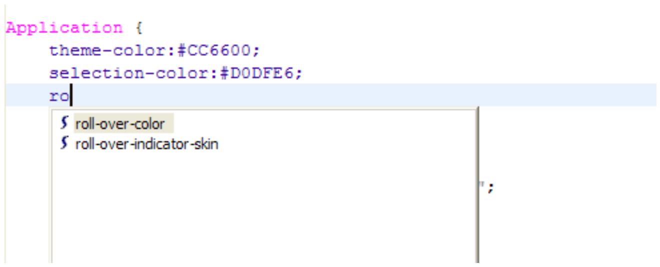
italic
normal

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.
```

Content Assist also provides hints for CSS styles within embedded `<mx:Style>` tags or in stand-alone CCS 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

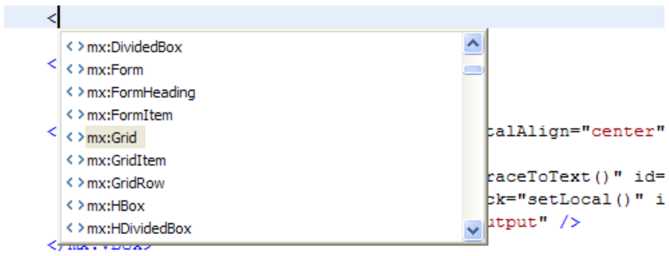
You can use code hints to write MXML, ActionScript, and CSS more rapidly and efficiently. Code hints appear as you enter code into the editor.

To 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.

To 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 context-sensitive help" on page 21.

To enable Dynamic Help:

- Select Help > Dynamic Help.

Navigating and organizing code

The Flex Builder editors provide many shortcuts for navigating your code. These include folding and unfolding code blocks, opening the source of code definitions, browsing and opening types, and so on.

For more information about these shortcuts, see the following topics:

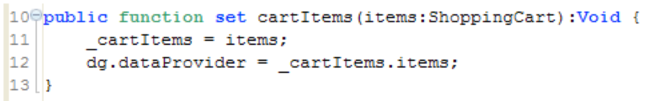
- "Folding and unfolding code blocks" on page 173
- "Using the Outline view to navigate and inspect code" on page 174
- "Using Quick Outline view in the editor" on page 177
- "Opening code definitions" on page 178
- "Browsing and opening types" on page 178
- "Showing line numbers" on page 179

Folding and unfolding code blocks

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*.

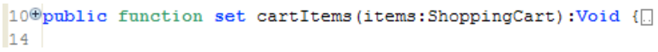
To fold a code block:

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



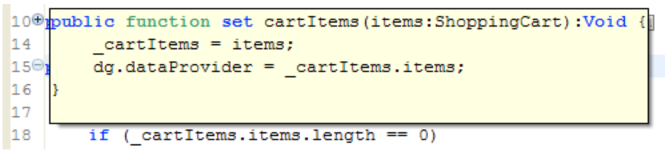
```
public function set cartItems(items:ShoppingCart):Void {
    _cartItems = items;
    dg.dataProvider = _cartItems.items;
}
```

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



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

You can then unfold the code block to make it visible again or 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)
```

To unfold a code block:

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

To set code-folding preferences:

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

Using the Outline view to navigate and inspect code

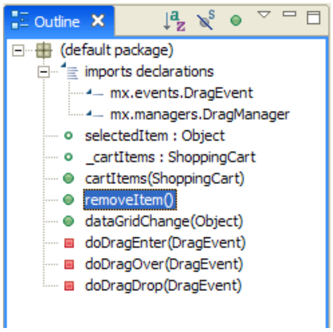
The Outline view is part of the Flex Development perspective (see "The Flex Development perspective" on page 31), and is therefore available when you edit code and design your application. You use the Outline view to more easily inspect and navigate the structure of your MXML and ActionScript documents.

The Outline view contains two modes: Class mode and MXML mode. 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).

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.



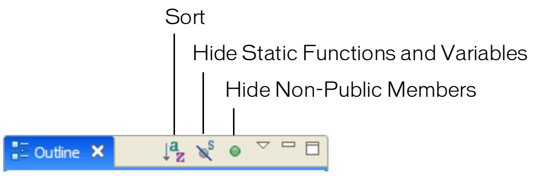
```
(default package)
  imports declarations
    mx.events.DragEvent
    mx.managers.DragManager
  selectedItem : Object
  _cartItems : ShoppingCart
  cartItems(ShoppingCart)
  removeItem()
  dataGridChange(Object)
  doDragEnter(DragEvent)
  doDragOver(DragEvent)
  doDragDrop(DragEvent)
```

Not included in this view are 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:



Sort alphabetically sorts all items in the view rather than by the sequence in which they occur in the document.

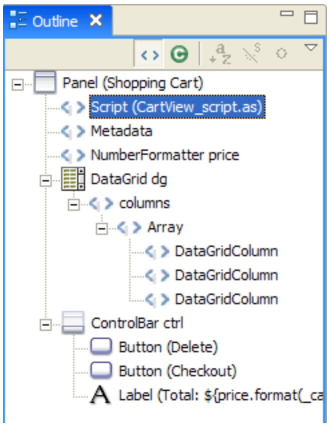
Hide Static Functions and Variables hides all static functions and variables.

Hide Non-Public Members hides non-public members.

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.

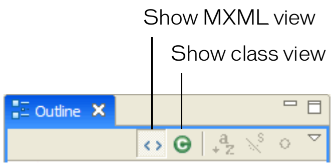
In MXML mode, each item in the Outline view represents an MXML tag and the following types of tags are displayed: components, controls, nonvisual tags such as `WebService` or `State`, component properties that are expressed as child tags (layout constraints, for example), and compiler tags such as `Model`,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



Not included in the Outline view display in MXML mode are 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

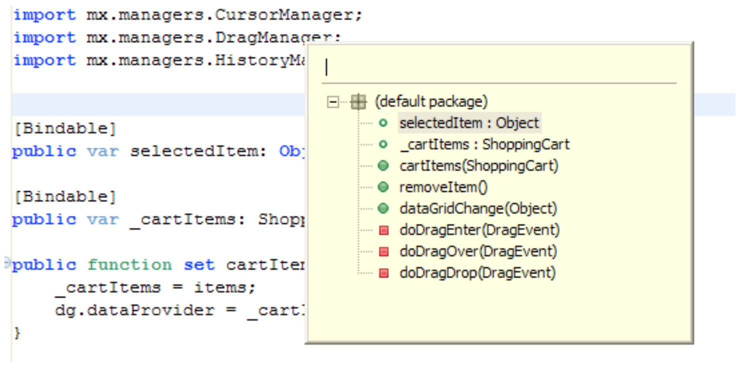
When the Outline view is in MXML mode, the toolbar contains two additional commands that allow you to switch between the MXML and class views.



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

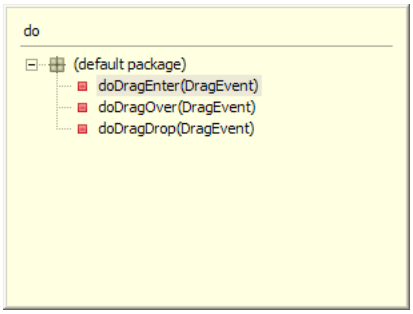
Within the ActionScript and MXML editors, you can access the Quick Outline view, which displays the Outline view in Class mode. The Quick Outline view is displayed in a pop-up window within the editor itself, not as a separate view, and you can use it to quickly navigate and inspect your code.



```
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 cartIter
    _cartItems = items;
    dg.dataProvider = _cart
}
```

```
(default package)
    selectedItem : Object
    _cartItems : ShoppingCart
    cartItems(ShoppingCart)
    removeItem()
    dataGridChange(Object)
    doDragEnter(DragEvent)
    doDragOver(DragEvent)
    doDragDrop(DragEvent)
```

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.



```
do
(default package)
    doDragEnter(DragEvent)
    doDragOver(DragEvent)
    doDragDrop(DragEvent)
```

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

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

To open the Quick Outline view:

- With an ActionScript or MXML document open in the editor, press Control+O.

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.

To 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.

To 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 (Command key on Macintosh) 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 types

To quickly browse all the available types in your project (including the Flex framework), use the Open Type dialog box.

To browse and open class types:

1. From anywhere in the workbench, press Control+Shift+T (Command+Shift+T on Macintosh).

   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, double-click the class type name.

   The source file opens in the editor.

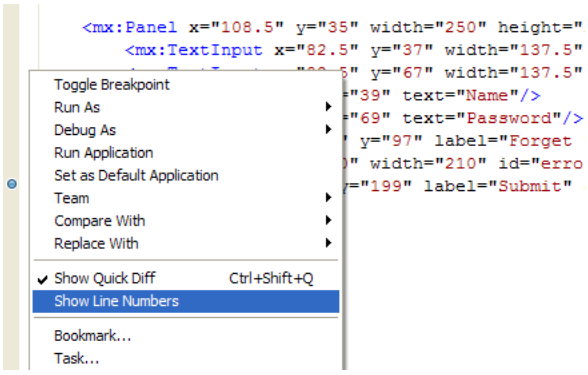
NOTE: You can browse, but not edit, the Flex framework classes.

Showing line numbers

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

To show line numbers in the editor:

1. In the editor, right-click (Control-click on Macintosh) 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 Show Line Numbers.

   Line numbers appear in the editor.

Formatting 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, and so on.

For more information about these shortcuts, see the following topics:

- "Organizing import statements" on page 180
- "Adding comment blocks" on page 180
- "Manually indenting code blocks" on page 180
- "Finding and replacing text in the editor" on page 181
- "About markers" on page 182

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. To help organize the code in your ActionScript documents, you can alphabetically sort import statements.

To sort import statements:

- With a ActionScript document that contains import statements open in the editor, press Control+Shift+O (Command+Shift+O on Macintosh).

Adding comment blocks

You can quickly add a new comment to your code or select existing lines of code and convert them into a comment block.

To add a comment block:

1. In the editor, select the line where you want to add the comment.
2. Press Control+Shift+C (Command+Shift+C on Macintosh).

To convert code into a comment block:

1. In the editor, select the lines of code to convert to a comment.
2. Press Control+Shift+C (Command+Shift+C on Macintosh).

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.

To 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.

To set indent preferences:

1. Select Window > Preferences > Flex > Editors.
2. You can select the indent type (tab or space) and the indent and tab size. (For more information, see "Setting editor preferences" on page 250.)

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 "Searching in the workbench" on page 99.

To find and replace text in the editor:

1. Open the document to search.
2. Do either of the following:
   - Press Control+F (Command+F on Macintosh).
   - 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.

Finding references and refactoring code

The Flex Builder search feature allows you to find and mark references to unique identifiers in ActionScript and MXML files, projects, or workspaces (for example, type, variable, function, accessors, metadata). You can refactor your code to rename identifiers while updating all references, move types (interfaces, classes), and update names and references accordingly.

To find all references:

1. In Source mode, select Search > References from the main menu (Ctrl-Shift-R). Then select File, Project, or Workspace. You can also right click to select References or Declarations from the context menu in the editor window.

NOTE: In the standalone version, these options are located in the Edit menu.

2. Search results appear in the Search window. You can choose from tree view and a list view.

To mark references:

1. Click on an identifier in the editor. All instances will be marked in a yellow box by default. Matching annotations will appear in the right margin.

2. To change the appearance of marked references, select Window > Preferences > General > Editors > Text Editors > Annotations.

To refactor your code:

1. In Source mode, select Source > Refactor from the main menu. You can also right click to select Declarations or References from the context menu in the editor window.

NOTE

In the standalone version these options are located in the Edit menu.

2. Choose Rename or Move.
3. Enter new name or file destination in the Rename Variable or Move dialog boxes.
4. If you select Rename and click Preview, any entries that cannot be confirmed will appear in a different window. Select the changes you wish to accept.

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 within those views. Selecting markers opens the associated document in the editor and, optionally, highlights the specific line of code.

The workbench generates task and problem markers automatically. You can manually add tasks and bookmarks. Each type of marker is summarized as follows.

Tasks Task markers represent a work item. Work items are generated automatically by the workbench and can also be added manually. 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 184.

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 187.

Bookmarks You can manually add bookmarks to either a line of code or to 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 185.

NOTE: The Tasks and Bookmarks views are not displayed by default in the Flex Builder development perspective. For more information about adding these views, see "Opening views" on page 90.

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 back to the location where the marker was set.

To show resource marker locations in the Navigator view:

1. In the Bookmarks, Problems, or Tasks views, select a marker.
2. Right-click (Control-click on Macintosh) to display the context menu, and select Show in Navigator.

   The resource that contains the marker is located and highlighted in the Navigator view.

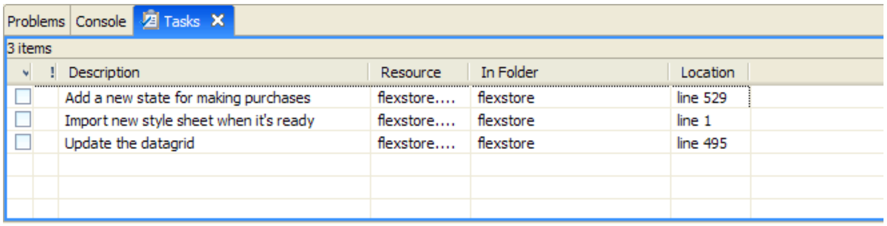
To go to a marker location:

1. Select a marker in the Bookmarks, Problems, or Tasks views.
2. Right-click (Control-click on Macintosh) the marker and select Go To.

   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 works items. All tasks are displayed and managed in the Tasks view (Window > Other Views > Basic > Tasks), as shown in the following example:



	!	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

You can add tasks to a line of code in a file or to a file or folder.

To add a task to a line of code:

1. Open a file in the editor, and then locate and select the line of code where you want to add a task.
2. Select Edit > Add Task.

   The Add Task dialog box appears.
3. Enter the task name and select a priority (High, Normal, Low), and click OK.

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

To add a task to a resource:

1. In the Navigator view, select a resource.
2. Select Edit > Add Task.

   The Add Task dialog box appears.
3. Enter the task name and select a priority (High, Normal, Low), and click OK.

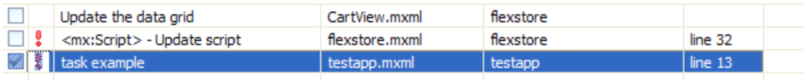
NOTE: The resource, as shown in the Navigator view, does not indicate that it was marked. You can view and manage all task markers in the Task view.

Completing and deleting tasks

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

To mark a task as complete:

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



☐		Update the data grid	CartView.mxml	flexstore	
☐	!	<mx:Script> - Update script	flexstore.mxml	flexstore	line 32
☑		task example	testapp.mxml	testapp	line 13

To delete a task:

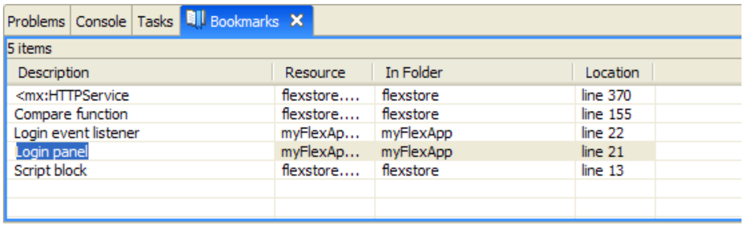
1. In the Tasks view, select a task.
2. Right-click (Control-click on Macintosh) the task and select Delete.

To delete all completed tasks:

- In the Tasks view, right-click (Control-click on Macintosh) 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 > Show Views > Other), as shown in the following example:



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

To add a bookmark to a line of code:

1. Open a file in the editor, and then locate and select the line of code to add a bookmark to.
2. Select Edit > Add Bookmark.

   The Add Bookmark dialog box appears.
3. Enter the bookmark name, and click OK.

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

To add a bookmark to a resource:

1. In the Navigator view, select a resource.
2. Select Edit > Add Bookmark.

   The Add Bookmark dialog box appears.
3. Enter the bookmark name, and click OK.

> **NOTE** The resource, as shown in the Navigator view, does not indicate that it was marked. You can view and manage all bookmarks in the Bookmarks view.

To delete a bookmark:

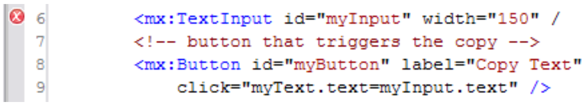
1. In the Bookmarks view, select the bookmark to delete.
2. Right-click (Control-click on Macintosh) to the bookmark and select Delete.

About syntax error checking

The Flex Builder compiler identifies syntax errors and reports them to you so that you can correct them as you're working, before you attempt to run your application.

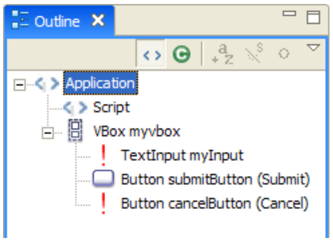
When code syntax errors are encountered, you're notified in the following ways:

- 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:



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

Coding syntax errors are identified when your projects are built. If you don't 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.

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 using 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. You can quickly navigate to the line of code by double-clicking it. The file is opened in the editor and the line of code is highlighted.

To 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 (Control-click on Macintosh) to display the context menu and select Go To.

To show the affected resource in the Navigator view:

- Select the problem and then right-click (Control-click on Macintosh) to display the context menu and select Show in Navigator. The resource is highlighted in the Navigator view.

Problems remain in the Problems view until you correct them or they are otherwise resolved.

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 101. For information about editing existing or creating new keyboard shortcuts, see "Changing keyboard shortcuts" on page 96.

Name	Keyboard shortcut	Description
Switch between Source and Design mode	Control+‘ Command+’ on Macintosh	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 173.
Context-sensitive Help	F1	Displays context-sensitive Help for the currently selected workbench element (editor, view, dialog box, and so on). For more information, see "Using context-sensitive help" on page 21.
Add Block Comment	Control+Shift+C Command+Shift+C on Macintosh	Adds block comment formatting to the currently selected lines of code or adds a comment at the insertion point. For more information, see "Adding comment blocks" on page 180.
Add CDATA	Control+Shift+D Command+Shift+D on Macintosh	Adds a CDATA statement at the insertion point so that you can add ActionScript to an MXML document.
Highlight Matching Bracket	Control+Shift+P Command+Shift+P on Macintosh	Highlights the matching brackets of the selected code statement.
Content Assist	Control+Space Command+Shift+Space on Macintosh	Displays code hinting. For more information, see "Using Content Assist" on page 172.

Name	Keyboard shortcut	Description
Go to Definition	F3	Open the source of an external code definition. For more information, see "Opening code definitions" on page 178.
Last Edit Location	Control+Q	Highlights the last edited line of code.
Organize Imports	Control+Shift+O Command+Shift+O on Macintosh	When editing ActionScript, using this keyboard shortcut alphabetizes any import statements contained in the document. For more information, see "Organizing import statements" on page 180.
Open Type	Control+Shift+T Command+Shift+T on Macintosh	Quickly browse all class types. For more information, see "Browsing and opening types" on page 178.
Open Resource	Control+Shift+R Command+Shift+R on Macintosh	Displays the Open Resource dialog box where you can quickly search for and open a resource in the editor.
Go to Line	Control+L Command+L on Macintosh	Displays the Go to Line dialog box where you can enter a line number and navigate to it in the editor.
Quick Outline	Control+O	Displays the Outline view in Quick mode in the editor. For more information, see "Using Quick Outline view in the editor" on page 177.

CHAPTER 10

Building Projects

10

Adobe Flex Builder 2 automatically builds 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.

If you need to modify the default build settings, there are several options 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.

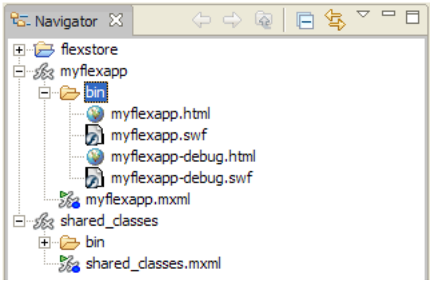
When your applications are ready to be released, you have the option of publishing all or selected parts of the application source code. Users can view your application source code in a web browser, similar to the way they are able to view HTML source code.

This topic contains the following sections:

Understanding how projects are built

When your Flex and ActionScript projects are built, a release and debug version of your application SWF files are placed in the project output folder along with the HTML wrapper files. (Flex Data Services projects may be compiled by Flex Builder or on the server when accessed). When a Flex library project is built, a SWC file is generated.

The debug version of your application contains debugging information and is used when debugging your application. The release 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. In a standard Flex application, a typical output folder resembles the following example.



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 207). For more information about running and debugging your applications, see "Running and Debugging Applications" on page 205.

When you use Flex Data Services, you create Flex applications that leverage the Flex server technologies. When building Flex Data Services applications, you have the option of compiling the output files locally using Flex Builder or on the server when the application is first accessed.

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 via 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).

Flex Data Services projects When using Flex Data Services. 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.

It's important to note that even though you can configure FDS projects to be compiled on the server, Flex Builder compiles these projects as you're developing 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.

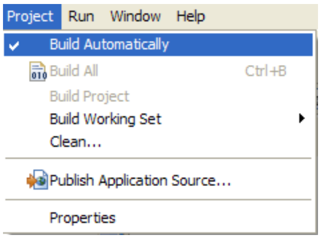
Flex library projects A library project's source files are components and related resources. When library projects are built, a SWC file is generated into the output folder. Archived into the SWC file is a SWF file containing the components and resources and a catalog.xml file that is the manifest of the elements contained within the SWF file.

Automatic builds

In the standalone configuration of Flex Builder, your applications are built automatically. In the plug-in configuration, you must select the Build Automatically option yourself. While you have the option of building your applications manually, as mentioned above, this should not be necessary. It's important to keep in mind that 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 won't get any feedback in the Problems view until the project is compiled; therefore, it's best to set Flex Builder to build automatically. For more information, see "Advanced build options" on page 197.

Advanced project build options

Using 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 197.



The Flex Builder compiler is incremental, building only those resources that have been added or affected by updates and ignoring 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 197.

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 197.

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 197, "Managing project application files" on page 64 and "Advanced build options" on page 197.

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.

For more information about working with the Problems view, see "Using the Problems view" on page 186.

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 MacOS, 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 201.

Command line access to the Flex 2 framework compilers

You have direct command line access to use the Flex 2 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 "Chapter 9, "About the command-line compilers" in *Building and Deploying Flex Applications*.

Learn more about building projects in Flex Builder

If you haven't already done so, review the getting started lesson "Create Your First Application" in the *Getting Started with Flex*. For more information, see the following topics:

- "Customizing project builds" on page 194
- "Advanced build options" on page 197

Customizing project builds

Allowing Flex Builder to automatically build your applications using the default project settings is the recommended approach to building your applications. You can, however, customize project builds to suit your needs. For example, you may want to change the default output folder or modify the compiler options.

For information about these customization options, see the following topics:

- "Enabling and disabling automatic builds" on page 194
- "Setting a project's output folder" on page 195
- "Modifying a project's build path" on page 195

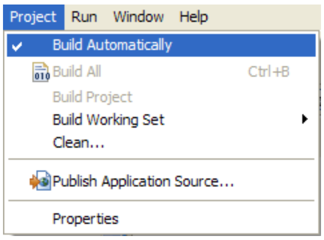
For more information about advanced control over the build process or creating highly customized builds, see "Advanced build options" on page 197.

Enabling and disabling automatic builds

In the standalone 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 197.

To enable or disable automatic builds, do either of the following steps:

- Select Project > Build Automatically.



- Select Windows > Preferences from the main menu, and then select the General > Workspace tab in the Preferences dialog box. Select or deselect the Build Automatically option.

The Build Automatically option affects all projects in the workspace.

Setting a project’s output folder

When you create a project in Flex Builder, by default, the build output is generated into the output folder. This doesn’t apply to Flex Data Services 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 has been created. You can either create a folder or select an existing folder in the workspace.

To change the build output folder:

1. In the 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 Flex Data Services 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.

   WARNING: 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’s 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 78.) 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.

To modify the source path:

1. Select a project in the 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 ActionScript Build Path properties page.)
4. Add a folder to the source path by clicking the New 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 196.
6. Modify the source path as needed, and click OK.

To 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 82).

   You can edit the path to the framework or, if you've 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.

To set or create a path variable:

1. Select a project in the 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 ActionScript Build Path properties page.)
4. You can create a path variable for any item on the path (this includes folders in the source path and SWC folders, projects, and SWC files in the library path). As an example, on the Source Path tab select the Add Folder button. The Add Folder dialog box appears.
5. Enter a path variable using the following format: `${`*`pathvariablename`*`}`.

NOTE: If the variable doesn't already exist, the path entry will fail. The list of existing resource variables is available by selecting Windows > Preferences from the main menu and then selecting General > Workspace > Linked Resources. You can also manage linked resource variables on this properties page.

6. Click OK to add the path variable to the path.

Advanced build options

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.

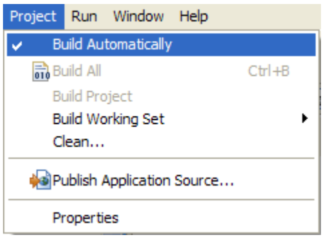
For more information about advanced build customization options, see the following topics:

- "Building projects manually" on page 197
- "Saving project resources automatically" on page 199
- "Performing a clean build" on page 199
- "Changing the project build order" on page 199
- "Customizing builds with Apache Ant" on page 201

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 92.

The build options are available in the Project menu, as shown in the following example.



To build a single project:

1. In the 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's output folder.

   NOTE: If any of your project files need to 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.

To 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's output folder. You are then prompted to save files if you haven't already chosen to save files automatically before a build begins (see previous note).

To build a working set, do either of the following:

- Define a working set by selecting Project > Build Working Set > Select Working Set from the main menu. The Select Working Set dialog box appears. Create a working set by selecting the New button. For more information about creating a working set, see "Creating working sets" on page 92.
- 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's 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.

To save resources automatically:

1. Select Windows > Preferences > General > Workspace.
2. Select the Save Automatically Before Build option.
3. 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.

To perform a clean build:

1. Select Project > Clean from the main menu.

   The Clean dialog box appears.
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

You can 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.

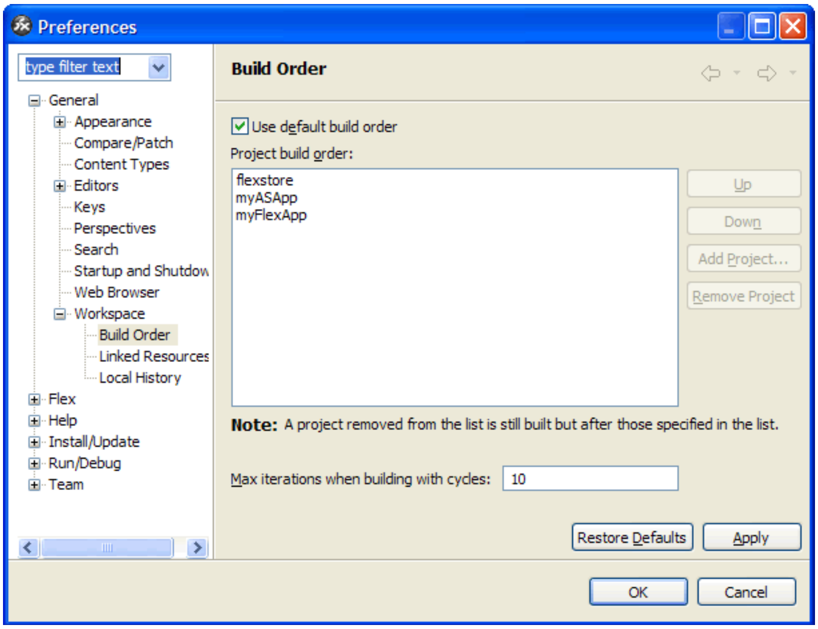
You can, however, change the build order. For example, you might change the build order if you've created a custom Ant builder and associated it with a project in your workspace, and you need to build that project before other projects are built. For more information about creating custom builders, see "Customizing builds with Apache Ant" on page 201.

To change the project build order:

1. Select Window > Preferences from the main menu.

   The Flex Builder Preferences dialog box appears.

2. Select General > Workspace > Build Order.



☑ Use default build order

Project build order:

flexstore
myASApp
myFlexApp

Note: A project removed from the list is still built but after those specified in the list.

Max iterations when building with cycles: 10

   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.

3. 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.

To create a builder:

1. In the Navigator view, select a project and then right-click (Control-click on Macintosh) to display the context menu and select Properties.
2. Select the Builder 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 and the Configuration Type dialog box appears.
4. Select the appropriate configuration type. Flex Builder supports the program type. Select it and click OK to continue. The new builder properties page appears. This is where you define the builder properties and reference the Ant script (an XML file).
5. Once you've defined a new builder, you can apply it to the project by clicking OK.

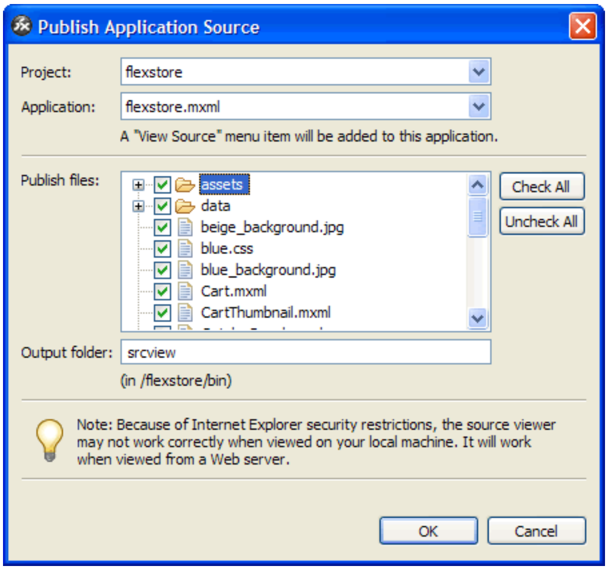
Detailed information about working with Ant build scripts can be found in the Eclipse documentation, which is available at http://help.eclipse.org/help31/index.jsp.

Publishing application source code

When your applications are ready to be released, you have the option of allowing users to view your application's source code and assets. 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. Allowing source viewing is a convenient way to share code with other Flex and ActionScript 3.0 developers.

To publish an application:

1. With the completed application project open in the editor, select Project > Publish Application Source. The Publish Application Source dialog box appears:



Publish Application Source

Project: flexstore

Application: flexstore.mxml

A "View Source" menu item will be added to this application.

Publish files:

- ☑ assets
- ☑ data
- ☑ beige_background.jpg
- ☑ blue.css
- ☑ blue_background.jpg
- ☑ Cart.mxml
- ☑ CartThumbnail.mxml

Output folder: srcview

(in /flexstore/bin)

Note: Because of Internet Explorer security restrictions, the source viewer may not work correctly when viewed on your local machine. It will work when viewed from a Web server.

2. Select the application file that will include the View Source menu. By default, the main application file is selected.
3. Uncheck the source files that you do not want to be published.
4. (Optional) Change the source output folder. By default, a source view folder is added to the project's output folder.
5. 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 which 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 using the Publish Application Source dialog box (see the preceding section). 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've changed the location of the source folder, your code should use the correct location.

CHAPTER 11

Running and Debugging Applications

11

To 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 can 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; evaluate watch expressions while the application is running; and so on.

This topic contains the following sections:

Understanding running and debugging applications in Flex Builder

After your projects are built into applications (see “Building Projects” on page 189), you can run and debug them in Flex Builder.

Use launch configurations to run and debug applications

A launch configuration defines such things as the project name, the main application file, the path to the run and debug versions of the application, and so on. 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 209.

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 207.

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 Flash desktop 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 210.

Run and debug applications in a browser or Flash desktop 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. You can instead run and debug your applications in the Flash desktop player (see “Running the application SWF file in the stand-alone Flash Player” on page 212). 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 212).

Run debugging tools from the code editor

The Flex Builder Debugging perspective provides all of the debugging tools you expect from a robust, full-featured development tool. You can set and manage breakpoints; control application execution by suspending, resuming, and terminating the application; step into and over the code; watch variables; evaluate expressions; and so on. For more information about the Flex Builder debugging tools, see “Debugging your applications” on page 214.

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 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 214.

Learn more about running and debugging applications

If you haven't already done so, review the getting started lesson in Chapter 17, "Debug an Application" in *Getting Started with Flex*. For more information, see the following sections in this topic:

- "Running your applications" on page 207
- "Managing launch configurations" on page 209
- "Debugging your applications" on page 214

Running your applications

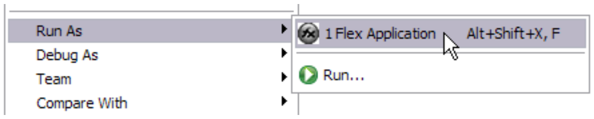
Your applications are run (and debugged) based on a launch configuration. When you create new Flex and ActionScript 3.0 applications, a launch configuration specifies the location of the built applications files, the main application file, and so on. You can modify the launch configuration or create custom launch configurations. For more information, see "Managing launch configurations" on page 209.

Running your projects opens the main application SWF file in your default web browser or directly in the Flash desktop player. For information about changing the default web browser or running and debugging with the Flash desktop player, see "Changing the default web browser" on page 212 and "Running the application SWF file in the stand-alone Flash Player" on page 212.

You can run your projects in a number of ways in Flex Builder. For example, you can use the Run command from the workbench main menu and toolbar, and from the Navigator view and code editor pop-up menus.

To run the currently selected project with the default Flex application launch configuration:

1. In the Navigator view, select the project to run.
2. On the main workbench toolbar, click the Run button ( ). Or, in the code editor, right-click (Control-click on Macintosh) to display the pop-up menu and select Run As > Flex Application.



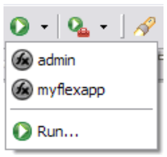
If your project is not built yet, Flex Builder builds and then runs it.

3. 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. (See "Managing project application files" on page 64.)

To create a custom launch configuration:

1. In the Navigator view, select the project you want to run and open the main application file in the editor.
2. On the main workbench toolbar, click the Run drop-down list and select Run.

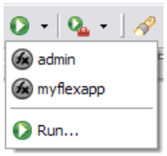


   The Create, Manage, and Run Configurations dialog box appears.

3. In the list of launch configurations, select Flex Application.
4. Click New.
5. Enter the launch configuration name.
6. Optional. Modify the configuration properties as needed.
7. Optional. To run the application immediately, click Run.

To run other application files in your project:

1. In the Navigator view, select the project to run.
2. On the main workbench toolbar, click the Run drop-down list.



NOTE The Run button has two elements: the main action button, and the drop-down list that shows all of 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 run. You can alternatively click the drop-down list and select any of the application files in the project. You can also access the launch configuration dialog box and create or edit a launch configuration by selecting the Run command.

3. Select the application you want to run.

To run a custom launch configuration:

- Select Run > Run from the main menu.

  The launch configuration dialog box appears. You can select an existing custom launch configuration or create a new one. See "Managing launch configurations" on page 209.

  - On the main workbench toolbar, click the Run button's drop down list (see note above) and select Run. The launch configuration dialog box appears. You can select an existing custom launch configuration or create one.

If you're using the plug-in configuration of Flex Builder, the run command works a bit differently. 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.

To run the last launched configuration:

- Click the Run button on the main toolbar (▶) or press Control+F11 (Command+F11 on Macintosh).

Managing launch configurations

Launch configurations are used to both run and debug applications. Flex Builder provides a default launch configuration for Flex and ActionScript 3.0 applications. When you first run or debug a project, a project-specific launch configuration is created. You can edit the launch configuration to change the default main application file or to modify the default launch path to run or debug in the Flash desktop player rather than in a web browser.

For more information about creating and customizing launch configurations, see the following topics:

- "Creating or editing a launch configuration" on page 210
- "Running the application SWF file in the stand-alone Flash Player" on page 212
- "Changing the default web browser" on page 212
- "Debugging your applications" on page 214

Creating or editing a launch configuration

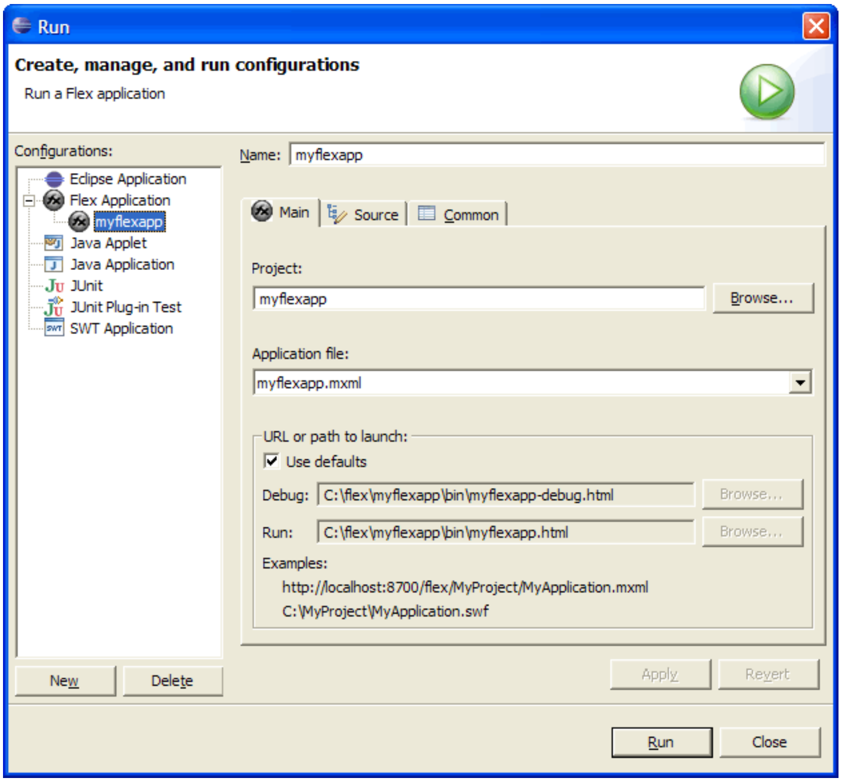
When you create and then build a project, it is ready to be run or debugged. Both running and debugging of the applications in your project are controlled by a launch configuration. By default, Flex Builder creates a launch configuration for each of the application files in your project the first time you run or debug them. The configurations are based on the default Flex application configuration and you can edit them as you see fit.

Launch configurations are managed in the Create, Manage, and Run Configurations dialog box.

To access and edit a launch configuration:

1. In the Navigator view, select a project.
2. With a project file open in the code editor, right-click (Control-click on Macintosh) to display the pop-up menu and select Run As > Run or Debug As > Debug.

   The Create, Manage, and Run Configurations dialog box appears:



Run

Create, manage, and run configurations

Run a Flex application

Configurations:
- Eclipse Application
- Flex Application
  - myflexapp
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: myflexapp

Main | Source | Common

Project: myflexapp

Application file: myflexapp.mxml

URL or path to launch:

☑ Use defaults

Debug: C:\flex\myflexapp\bin\myflexapp-debug.html

Run: C:\flex\myflexapp\bin\myflexapp.html

Examples:

http://localhost:8700/flex/MyProject/MyApplication.mxml

C:\MyProject\MyApplication.swf

3. Select the launch configuration to edit.

   A number of configurations may be listed if you have other projects in the workspace, if you've 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

By default, your applications are run or debugged in your system's default web browser. You can change which web browser is used to run and debug applications (see "Changing the default web browser" on page 212). You can also choose to run and debug your applications in the stand-alone Flash Player by making a simple change to the application's launch configuration.

To run and debug applications in the Flash desktop player:

1. Access the Create, Manage, and Run Configurations dialog box (see "Creating or editing a launch configuration" on page 210).
2. Select the launch configuration you want to modify.
3. In the Main tab, deselect Use Defaults.
4. In either or both of the Run and Debug paths, click Browse.

   The file selection dialog box appears and lists the contents of the project's build output folder.
5. Select the application SWF file (for example, myflexapp.swf).

   If you're modifying the debug path, select the debug version of the application SWF file (for example, myflexapp-debug.swf).
6. Click Open to select the file and return to the configuration dialog box.
7. Apply your changes and use the modified configuration to run or debug the application.

Changing the default web browser

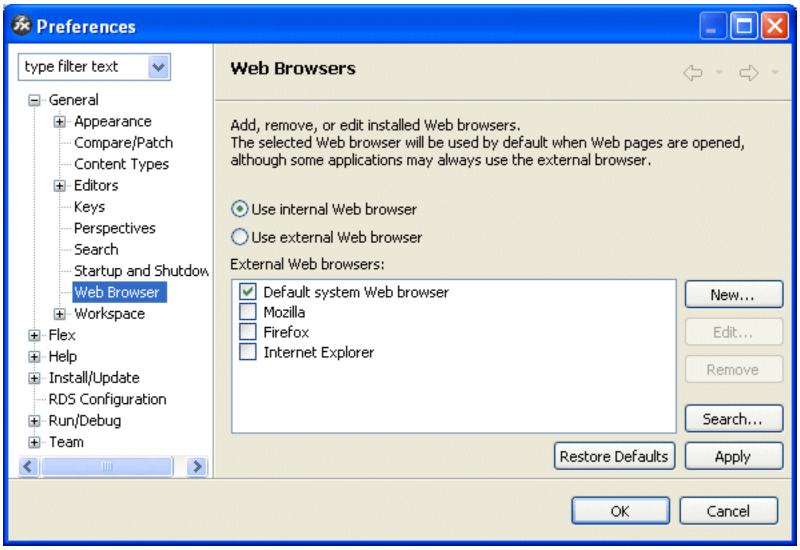
Flex Builder uses the system default web browser when running and debugging applications. While you can't set each launch configuration to use a specific web browser, you can change the workbench web browser setting, which affects how all of your applications are run and debugged.

To change the default web browser:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the General > Editors > Web Browser preferences page.



Web Browsers

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
- Mozilla
- Firefox
- Internet Explorer

3. 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.

4. Click OK to apply your changes.

Debugging your applications

Debugging is similar to running your applications, except that when debugging you can control the execution of the application to stop at specific points in the code, to monitor important variables, and to test fixes to your code. Both running and debugging use a configuration to control how applications are launched. When debugging, 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 37.

In some cases, you will be prompted to look at the Exclipse log file. For more information, see "Eclipse environment errors in the log file" on page 193.

For more information about debugging your applications, see the following topics:

Starting a debugging session

To begin a debugging session, you run the application's launch configuration in debugging mode.

To debug an application:

1. In the Navigator view, select the project to debug.
2. Select the Debug button from the main workbench toolbar.

   If you project isn't already built, Flex Builder builds and then runs it in debug mode.
3. Your application appears in your default web browser or the Flash desktop 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.

Starting a debugging session in the plug-in configuration of Flex Builder

If you're using the plug-in configuration of Flex Builder, the debug command works a bit differently. Instead of running the currently 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 that you can inspect your code and use the Flex Builder debugging tools to explore options for fixing errors. You add breakpoints in the code editor and then manage them in the Breakpoints view when you're debugging 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're coding or while you're debugging.

To set a breakpoint in the code editor:

1. Open a project file that contains ActionScript 3.0 code.
2. Locate the line of code on which you want to set a breakpoint and in the marker bar double-click 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 debugging perspective is displayed, and the line of code marked with a breakpoint 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 219).

To 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 217).

Managing breakpoints in the Breakpoints view

The Breakpoints view allows you to manage breakpoints during a debugging session by removing them, disabling and enabling them, skipping them, and so on.

The following commands are available from the Breakpoints view toolbar and pop-up menu.

Command	Description
Go to File	Opens the file (if it isn't 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.
Enable	Enables the selected breakpoints.
Disable	Disables the selected breakpoints. A disabled breakpoint does not cause the execution of an application to be suspended.
Remove	Removes the selected breakpoints.
Remove All	Removes all breakpoints.
Show Breakpoints Supported by Selected Target	Displays breakpoints that are applicable to the select debug target.
Skip All Breakpoints	Skips all breakpoints.

Removing breakpoints

You can remove one, a few, or all of the breakpoints in the Breakpoints view.

To remove breakpoints in the Breakpoints view:

- Select a breakpoint from the list of breakpoints, and then click Remove in the Breakpoints view toolbar.

You can also remove all the breakpoints in the Breakpoints view in a single action.

To remove all breakpoints from the Breakpoints view:

- In the Breakpoints view, click Remove All in the Breakpoints view toolbar.

Disabling breakpoints

To bypass some or all of the breakpoints without removing them, you can disable them instead.

To disable breakpoints:

- In the Breakpoints view, deselect the breakpoint selection box.
- Select one or more breakpoints, right-click (Control-click on Macintosh) to display the pop-up menu, and select Disable.

To enable breakpoints that are disabled, select them again, or use the pop-up menu and select the Enable command.

Managing the debugging session in the Debug view

The Debug view is the control center of the debugging perspective. You use it to control the execution of the application, to suspend, resume, or terminate the application, to step into and over code, and so on.

The Debug view provides the following debug commands, which are available from the Debug view toolbar and pop-up menu:

Command	Description
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.
Terminate and Relaunch	Terminates the current debugging session and relaunches the application into a new debugging session.
Terminate and Remove	Terminates the debugging session and removes it from the Debug view.
Terminate All	Terminates all active application debugging sessions. You can only debug one Flex application at a time; however you can simultaneously debug other types of applications (for example, a Java application).
Disconnect	Disconnects the debugger when debugging remotely.
Remove All Terminated	Clears all terminated debugging sessions.
Use Step Filters	This command is not supported in Flex Builder.
Step Into	Steps into the called function and stops at the first line of the function.
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.
Show Qualified Names	This command is not supported in Flex Builder.
Copy Stack	Copies the selected stack to the clipboard.
Drop to Frame	This command is not supported in Flex Builder.
Relaunch	Re-launches the application into a new debugging session.
Properties	Displays the properties of the selected process in the stack.

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 and pop-up menu:

Command	Description
Open Link	Opens the linked file in the appropriate editor.
Clear Console	Clears all content from the Console view.
Terminate	Terminates the debugging session.
Remove All Terminated	Clears all terminated debugging sessions.
Scroll Lock	Prevents the Console view from scrolling.
Pin Console	Prevents the console from refreshing its contents when another process is selected.

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.

The Variables view provides the following actions, which are available from the Variables view toolbar and pop-up menu:

Command	Description
Show type names	Displays the type names of variables.
Show logical structure	This command is not supported in Flex Builder.
Change value	Change the value of a selected variable.
Watch	Adds the selected variables to the Expressions view so that they can be monitored.

To watch a variable:

1. Select the variables or variable members to watch.
2. Right-click (Control-click on Macintosh) to display the pop-up menu and select Watch.
3. The variables are added to the Expressions view.

To change the value of a variable:

1. Select the variable to modify.
2. Right-click (Control-click on Macintosh) to display the pop-up menu and select Change Value.

   The Set Value dialog box appears.
3. Enter the new value and click OK.

   The variable contains the new value.

Modified variables are displayed in red.

To 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").

  The list of variables scrolls to highlight the appropriate variable names as you type.

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 and pop-up menu:

Command	Description
Select All	Selects all items in the Expressions view.
Copy Variables	Copies the selected variable to the clipboard.
Show Type Names	Shows the object types for items in the Expressions view.
Show Logical Structure	This command is not supported in Flex Builder.
Remove	Removes the selected variable or watch expression.
Remove All	Removes all variables and watch expressions from the Expressions view.
Change Value	Change the value of a selected variable or watch expression. Changes are valid during the debugging session only.
Add Watch Expression	Displays the Add Watch Expression dialog box, which you can use to add a new watch expression.
Reevaluate Watch Expression	Reevaluates the selected watch expression.
Disable	Disables the selected variable or watch expression.
Enable	Enables the selected variable or watch expression.
Edit Watch Expression	Displays the Edit Watch Expression dialog box, which you can use to edit the selected watch expression.

CHAPTER 12

Working with Data in Flex Builder

12

In Adobe Flex Builder, you can drag data-driven controls from the Components view into your application in Design mode. Then, you interact with data and the data-driven controls directly in your MXML and ActionScript code in Source mode. This topic provides you with an overview of how to work with data and also contains information about managing Flash Player data access security issues and using Flex Builder to work with a proxy service.

This topic contains the following sections:

About working with data in Flex Builder

You work with data in Flex Builder by directly modifying your Flex and ActionScript application code. This section provides an overview of working with data and includes references to in-depth information in *Flex Developer's Guide*.

Data-driven controls and containers

Flex provides you with control and container components from which you build your Flex application user interface. A number of these components present data to users, which they can select and interact with when using the application. A few examples of how data-driven controls are used include the following:

- On an address form you provide a way for users to select their home country (or other typical form input) using the ComboBox or List controls.
- In a shopping cart application, you use the TileList or HorizontalList controls to present product data that includes images.
- You provide standard navigation options using containers such as the TabBar, LinkBar, and ButtonBar controls.

Common to all of the data-driven controls is that you provide data input to them with a *data provider* (discussed in the next section).

For information about using the data-driven controls, see Chapter 13, "Using Data-Driven Controls" in *Flex Developer's Guide*.

Data providers and collections

A *data provider* is a collection of objects that contains data required by a component. Because the data provider is separate from the component, it can be used by multiple components and is in this sense a "model" that can be used by many views.

The following is a simple example of how a data provider is defined (as an ActionScript array) and used by a control:

```
<mx:Application>
  <mx:Script><![CDATA[
    public var stateArray:Array = ["AL", "AK", "AR"];
  ]]></mx:Script>
  <mx:ComboBox id="stateCBO" dataProvider="{stateArray}" />
</mx:Application>
```

For more information about data providers and collections, see Chapter 8, "Using Data Providers and Collections" in *Flex Developer's Guide*.

Data binding

In the code example from the previous section, you may have noticed that the value of the ComboBox control's `dataProvider` property is `"{stateArray}"`. This is an example of data binding.

A 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 brace ({ }) syntax (as shown in the example) or the `<mx:Binding>` tag. For more information, see Chapter 42, "Using data binding with data models" and Chapter 42, "Binding Data" in *Flex Developer's Guide*.

Data models

A data model is an object 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 using a tag such as `<mx:Model>`, or 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">
  <name>
    <first>Jennifer</first>
    <last>Nadeau</last>
  </name>
  <age>30</age>
  <work_tel>555-555-5555</work_tel>
</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 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 Chapter 43, "Storing Data" in *Flex Developer's 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 Chapter 44, "Validating Data" in *Flex Developer's 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 properly format data after it has been entered by the user. For example, a user may 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 may work with either as MXML tags or ActionScript classes.

For more information, see Chapter 45, "Formatting Data" in *Flex Developer's Guide*.

Data services

When your applications have complex requirements for accessing and managing data, you use advanced data services that help you move data to and from your applications.

In Flex, data services are included as part of Flex Data Services, a separate product (see "About Flex Data Services" in *Getting Started with Flex*). When you install Flex Data Services, you can use Flex Builder to create projects that implement the following data services:

RPC Services allow you to send and receive data using the following remote procedure call (RPC) services: HTTPService, WebService, and RemoteObject. Both HTTPService and WebService can be used by any Flex application. The RemoteObject service requires an additional server, such as Flex Data Services, to process the RemoteObject requests. Each enables you to make asynchronous requests to remote services that process the requests and then returns data directly to your Flex application. For more information, see Chapter 40, "Accessing Server-Side Data with Flex" in *Flex Developer's Guide*.

LiveCycle Data Services ES Data Management Service allows you to create applications that work with distributed data and to also manage large collections of data and nested data relationships, such as one-to-one and one-to-many relationships.

LiveCycle Data Services ES 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.

NOTE

When using Flex Builder to develop applications using data services, you must configure 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 225.

Learn more about working with data

If you haven't already done so, review the getting started lessons Chapter 9, "Retrieve and Display Data" and Chapter 18, "Use Web Services" in *Getting Started with Flex*. For more information about working with data, see the *Flex Developer's Guide* topics noted above.

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.)

When you need 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 226.
- 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. "Setting up Flex Builder to use a proxy for accessing remote data" on page 227.

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 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 you want to access. The following is an example of 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. Flex Data Services provides a complete proxy management system for Flex applications. You can also create a simple proxy service 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).

TIP 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've 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 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 machine, you can modify the project build settings to automatically copy the compiled application files to the appropriate location on your Web server.

To modify the project build path:

1. In the 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 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 need to modify the project's launch configuration.

To modify the launch configuration:

1. With the project's main application file open in Flex Builder, right-click (Control-click on Macintosh) in the editor and select Run As > Run from the context menu. The Launch Configuration dialog box appears.
2. Select the project's launch configuration from the list of configurations.
3. On the Main tab you can modify the launch path by deselecting the Use Defaults check box.
4. In the Run input box, enter the URL to the main application HTML wrapper file (for example: http://localhost/myApp/myApp.html). If you're running the application directly in the stand-alone Flash Player, instead of a web browser, enter the main application SWF file.
5. In the Debug input box, enter the URL to the debug version of the application (for example: http://localhost/myApp/myApp-debug.html or http://localhost/myApp/myApp-debug.swf).
6. Click Apply and then Close.

CHAPTER 13

Flex Builder User Interface Reference

13

This topic contains descriptions of the Adobe Flex Builder 2 user interface elements (views, editors, dialog boxes, preferences, and properties).

NOTE

You may find in-depth information about the Eclipse workbench features by referring to the Eclipse Workbench User's Guide at http://help.eclipse.org/help31/index.jsp.

This topic contains the following sections:

- "Setting project properties"
- "Using Flex Builder views"
- "Creating project resources"
- "Setting editor preferences"
- "Setting running and debugging preferences"

Setting project properties

This section provides information about setting project properties.

- "Setting project text encoding properties"
- "Setting project compiler properties"
- "Setting project application file properties"
- "Setting Flex and ActionScript project build path properties"
- "Setting Flex library project build path properties"
- "Setting Flex server properties"
- "Setting project builder properties"
- "Setting project references"

Setting project text encoding properties

Use the Info properties page to specify Flex, ActionScript, and Flex Library project text encoding properties.

To specify text encoding properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) to display the context menu and select Properties > Info.
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.

Setting project compiler properties

Use the Compiler properties page to modify how Flex, ActionScript, and Flex Library projects are built.

To set project compiler properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) 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 Directory 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's compiled into the SWF file.

   Generate Accessible SWF File enables accessibility features when compiling the application. For more information about creating accessible applications, see Chapter 39, "Creating Accessible Applications" in *Flex Developer's Guide*.

Enable Compile-time 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" in *Building and Deploying Flex 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 Chapter 11, "Running and Debugging Applications" in *Using Flex Builder*. For more information about warnings, see "Viewing errors and warnings" in *Building and Deploying Flex 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 the component compiler" in *Building and Deploying Flex Applications*.

Additional Compiler Arguments allows you to add optional compiler arguments. For more information about using compiler arguments, see Chapter 9, "Using the Flex Compilers" in *Building and Deploying Flex Applications*.

Setting 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.

Detect Flash Version checks to see if the user has the correct 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 History Management (browser Back button) enables your application to use the web browser's back button.

■ Modify the compiler properties as needed, and click OK.

Related Topics

- Chapter 39, “Creating Accessible Applications” in *Flex Developer’s Guide*
- “Viewing errors and warnings” in *Building and Deploying Flex Applications*
- Chapter 11, “Running and Debugging Applications” in *Using Flex Builder*
- Chapter 9, “Using the Flex Compilers” in *Building and Deploying Flex Applications*

Setting project application file properties

Use the Applications properties page to specify the Flex and ActionScript project files that should be compiled as applications.

To set applications properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) to display the context menu and select Properties > (Flex or ActionScript) Applications.
3. You can set the following compiler properties:

   Select the Runnable Application Files is 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.

Related Topics

- “Switching the main application file” on page 64
- “Managing project application files” on page 64

Setting 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 “Setting Flex library project build path properties” on page 233.

To specify build path properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) to display the context menu and select Properties > (Flex or ActionScript) Build Path.
3. Specify the following properties:

   Source path is used 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. 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 is used to 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 is 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 is 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 a project's output folder" on page 195.

   Output folder URL is used to specify the server location of the compiled application files. (Optional)
4. Click OK.

Related topics

- "Adding resource folders to the project source path" on page 72
- "Using SWC files in your projects" on page 82

Setting Flex library project build path properties

Use the Flex Library Build Path properties page to specify Flex Library project build path properties. For information about setting Flex and ActionScript build path properties, see "Setting Flex and ActionScript project build path properties" on page 232.

To specify build path properties:

1. In the 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 81.

   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 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 82.

   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 a project's output folder" on page 195.
4. Click OK.

Related topics

- "About library projects" on page 78
- "Creating Flex component library files" on page 80
- "Selecting library project elements to include in the SWC file" on page 81
- "Using SWC files in your projects" on page 82
- "Setting a project's output folder" on page 195

Setting 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 Flex Data Services.

To specify server properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) 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.

Related topics

- "Creating a Flex ColdFusion Flash Remoting service project" on page 57
- "Creating a Flex Data Services project" on page 58

Setting 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 201.

To access project builder properties:

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) to display the context menu and select Properties > Builders.

Setting project references

Use the Project References properties page to create references to other projects in the workspace. For more information, see "Adding project references" on page 73.

1. In the Navigator view, select a project.
2. Right-click (Control-click on Macintosh) 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" on page 236
- "States view" on page 245
- "Outline view" on page 245

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.

For more information on each category in the Components view, see the following sections:

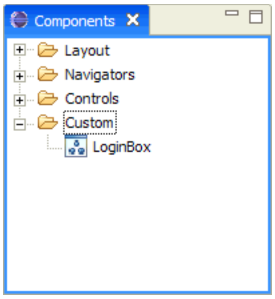
- "Custom category" on page 237
- "Layout category" on page 240
- "Navigators category" on page 242
- "Charts category" on page 243

Related topics

- "Adding components visually" on page 108
- "Setting component properties" on page 120

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 isn't in a project, the Custom category is empty.

The Components view only lists visible custom components (components that inherit from the UIComponent class).

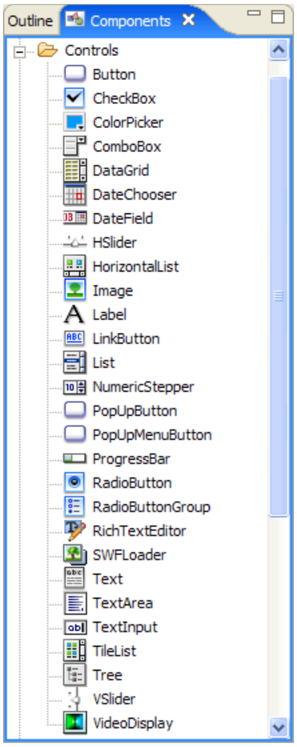
The Custom category lists all the custom components that you saved in the current project or in the source path of the current project. For example, if you create a component file called LoginBox.mxml and save it in your project, the LoginBox component appears in the Custom category of the Components view. Like any component, you can insert these components in your layout by dragging them from the Components view.

Related topics

- Chapter 8, "Creating Custom MXML Components," on page 157
- Chapter 7, "Creating Simple MXML Components" in *Creating and Extending Flex Components*

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.

Component	Description
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.
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.

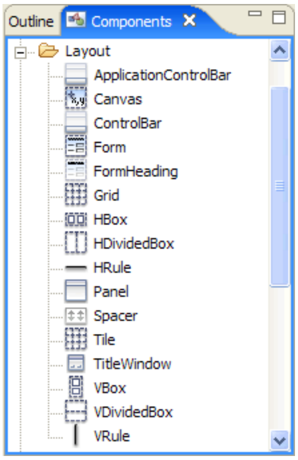
Component	Description
VideoDisplay	Control that lets you play an FLV file in a Flex application. It supports streaming video from the Flash Media Server, FLV files, and from a Camera object.
VSlider	Control that lets users select a value by moving a slider vertically.

Related topics

- "Adding data provider controls" on page 138
- Chapter 10, "Using Controls" in *Flex Developer's Guide*
- Chapter 13, "Using Data-Driven Controls" in *Flex Developer's Guide*

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.

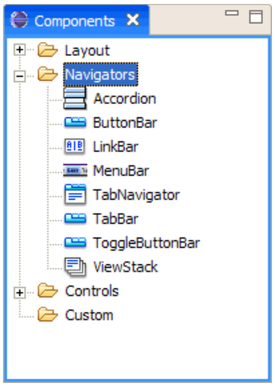
Component	Description
Form	Container that arranges its children in a standard form format. Should only be used in a Panel container.
FormHeading	Control that specifies an optional label for a group of FormItem containers. Only valid in a Form.
Grid	Container that arranges children as rows and columns 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.

Related topics

- Chapter 17, "Using Layout Containers" in *Flex Developer's Guide*

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.
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.

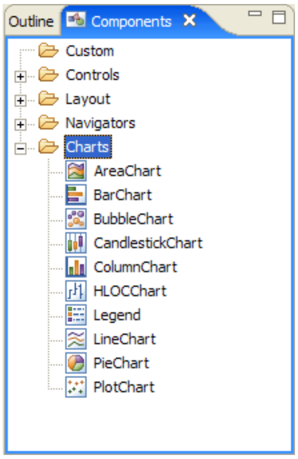
Related topics

- "Creating layouts in navigator containers" on page 135
- "Letting users select a view in a ViewStack container" on page 136
- Chapter 18, "Using Navigator Containers" in *Flex Developer's Guide*

Charts category

The Flex charts are available in Adobe Flex Builder with Charting. A trial version of the charts is included in the standard version of Flex Builder. You can also obtain the full version of the charts by purchasing Adobe Flex Charting, a separate product.

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.

Component	Description
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).

Related topics

- "Adding charting components" on page 140
- Chapter 46, "Introduction to Charts" in *Flex Developer's Guide*

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.

Related topics

- "Setting component properties" on page 120

Navigator view

The Navigator view allows you to manage the projects and resources contained within the workspace. For more information about using the Navigator view, see the following topics:

- "Projects in the Navigator view" on page 53

- “Managing projects” on page 60
- “Managing project resources” on page 65

Outline view

You use the Outline view to more easily inspect and navigate the structure of your MXML and ActionScript documents. For more information about using the Outline view, see the following topics:

- “Using the Outline view to navigate and inspect code”
- “Outline view in Class mode”
- “Outline view in MXML mode”
- “Using Quick Outline view in the editor”

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 254.

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 246.

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.

TIP Selecting Edit › Undo restores the state that you deleted.

Related topics

- “Creating a view state” on page 145
- “Creating a state based on an existing state” on page 146
- “Modifying the appearance of existing states” on page 149
- Chapter 29, “Using View States” in *Flex Developer’s Guide*

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.

Related topics

- "Creating a view state" on page 145
- Chapter 29, "Using View States" in *Flex Developer's Guide*

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.

Related topics

- "Modifying the appearance of existing states" on page 149
- Chapter 29, "Using View States" in *Flex Developer's Guide*

Creating project resources

This section provides information about dialog boxes used to create project resources.

- "Setting the New ActionScript Class dialog box options" on page 247
- "Setting the New ActionScript Interface dialog box options" on page 248
- "Setting the New MXML Component dialog box options" on page 249
- "Create Chart dialog box" on page 250

Setting the New ActionScript Class dialog box options

The New ActionScript Class dialog box lets you rapidly create an ActionScript class.

To set the dialog box options:

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 "Adding components visually".

8. Write the definition of your ActionScript class.

   For more information, see Chapter 9, "Creating Simple Visual Components in ActionScript" in *Creating and Extending Flex Components*.

Related topics

- "Creating an ActionScript class" on page 76
- "Adding components visually" on page 108
- Chapter 9, "Creating Simple Visual Components in ActionScript" in *Creating and Extending Flex Components*.

Setting the New ActionScript Interface dialog box options

The New ActionScript Interface dialog box lets you rapidly create an ActionScript interface.

To set the dialog box options:

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.

Related topics

■ "Creating an ActionScript interface" on page 77

Setting the New MXML Component dialog box options

The New MXML Component dialog box lets you rapidly create an MXML component.

To set the dialog box options:

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 "Adding components visually" on page 108.

5. Develop your custom component.

   For more information, see Chapter 7, “Creating Simple MXML Components” in *Creating and Extending Flex Components*.

Related topics

- “Adding components visually” on page 108
- Chapter 7, “Creating Simple MXML Components” in *Creating and Extending Flex 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” in *Flex Developer’s 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.

Related topics

- “Adding charting components” on page 140
- Chapter 46, “Introduction to Charts” in *Flex Developer’s Guide*

Setting editor preferences

This section provides information about setting Flex Builder editor preferences.

- “Setting Flex editor preferences” on page 251
- “Setting MXML editor preferences” on page 251
- “Setting ActionScript editor preferences” on page 252
- “Setting CSS editor preferences” on page 252
- “Setting MXML Code Assist preferences” on page 253
- “Setting ActionScript Code Assist preferences” on page 254

Setting Flex editor preferences

This dialog box allows you to specify preferences that apply to the MXML, ActionScript, and CSS editors.

To specify Flex editor preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Editors preferences page and set the following preferences:

   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 with code hints while entering code.

3. Click OK.

Related topics

- "Manually indenting code blocks" on page 180
- "Folding and unfolding code blocks" on page 173

Setting MXML editor preferences

This dialog box allows you to specify MXML editor preferences.

To specify MXML editor preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Editors > MXML Editor preferences page and set the following preferences:

   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.

3. Click OK.

Setting ActionScript editor preferences

This dialog box allows you to specify ActionScript editor preferences.

To specify ActionScript editor preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Editors > ActionScript Editor preferences page and set the following preferences:

   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.

3. Click OK.

Setting CSS editor preferences

This dialog box allows you to specify ActionScript editor preferences.

To specify CSS editor preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Editors > CSS Editor preferences page and set the following preferences:

   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.

   Enable Auto-activation enables Content Assist to automatically provide with 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 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.

To specify MXML Code Assist preferences:

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:XMList, and mx:request automatically adds a namespace attribute to mx:XML, mx:XMList, 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 with 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.

Related topics

- “About Flex Builder content assistance” on page 169

Setting ActionScript Code Assist preferences

This dialog box allows you to specify ActionScript Code Assist preferences.

NOTE: The Code Assist feature is also referred to as Content Assist and these terms are used interchangeably.

To specify ActionScript Code Assist preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Editors > ActionScript Editor > Code Assist preferences page and set the following preferences:

   Enable Auto-activation enables Code 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 Code Assist.

3. Click OK.

Related topics

- "About Flex Builder content assistance" on page 169

Setting running and debugging preferences

This section provides information on setting running and debugging preferences.

- "Setting launch preferences" on page 255
- "Setting Console view preferences" on page 256
- "Setting Run/Debug preferences" on page 257
- "Setting Flex debugging preferences" on page 258

Setting launch 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.

To set launch preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Run/Debug > Launching preferences page and set the following preferences:

   Build (If Required) Before Launching is selected by default. If needed, builds the project specified in the launch configuration before launching the application.

   Save 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.

   Remove Terminated Launches When a New Launch is Created clears all of the terminated launches in the Debug view.

   Open the Associated Perspective When Launching Select 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.

   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.

3. Click OK.

Related topics

- "Running your applications" on page 207
- "Setting Run/Debug preferences" on page 257
- "Setting Flex debugging preferences" on page 258

Setting Console view preferences

You can modify preferences that affect how text in the Console view is displayed.

To set the Console preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Run/Debug > Console preferences page and set the following preferences:

   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.

   Display 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.

3. Click OK.

Related topics

- "Using the Console view" on page 218
- "Setting Run/Debug preferences" on page 257
- "Setting launch preferences" on page 255
- "Setting Flex debugging preferences" on page 258

Setting Run/Debug preferences

You can specify general settings for running and debugging your applications in Flex Builder.

To set running and debugging preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Run/Debug preferences page and set the following preferences:

   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.

   Open the Associated Perspective When an Application Suspends Select Always, Never, or Prompt.

   Skip Breakpoints During a 'Run to Line' Operation is not supported in Flex Builder.

   Variables View Changed Value Color specifies the text color of the changed value of a variable.

   Memory View Unbuffered Lines Color is not supported in Flex Builder.

   Memory View Buffered Lines Color is not supported in Flex Builder.

3. Click OK.

Related topics

- "Debugging your applications" on page 212
- "Setting launch preferences" on page 255
- "Setting Flex debugging preferences" on page 258
- "Setting Console view preferences" on page 256

Setting Flex debugging preferences

You can specify debugging settings that are specific to the Flex Builder debugger.

To set Flex debugging preferences:

1. From the main menu, select Window > Preferences.

   The Preferences dialog box appears.

2. Select the Flex > Debug preferences page and set the following preferences:

   Warn When Trying to Launch Multiple Debugging Sessions Under Mozilla Firefox 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 it 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.

3. Click OK.

Related topics

- "Debugging your applications" on page 212
- "Setting Run/Debug preferences" on page 257
- "Setting launch preferences" on page 255
- "Setting Console view preferences" on page 256

Index