Using
ADOBE® FLASH® BUILDER 4

Last updated 3/21/2010

Using Adobe® Flash® Builder™ 4.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA.

Contents

Chapter 1: About Flash Builder

Adobe® Flash® Builder™ is an integrated development environment (IDE) for building cross-platform, rich Internet applications (RIAs). Using Flash Builder, you can build applications that use the Adobe Flex® framework, MXML, Adobe Flash Player, Adobe AIR®, ActionScript 3.0, Adobe® LiveCycle® Data Services ES, and the Adobe® Flex® Charting components. Flash Builder also includes testing, debugging, and profiling tools that lead to increased levels of productivity and effectiveness.

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

What you can do with Flash Builder

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

- Create Flex projects that can work with any backend server technology, including Adobe ColdFusion® and Adobe LiveCycle® Data Services. with or without using the Flex server. See "Creating Flex projects" on page 40.
- Create applications that access remote data services. Flash Builder provides tools and wizards that facilitate access to data services. See Accessing data services overview.
- Create ActionScript projects. See "ActionScript projects" on page 63.
- Write and edit your application source code using editors that provide features such as code refactoring, code hinting, streamlined code navigation, and automatic syntax error checking. See "About code editing in Flash Builder" on page 96.
- Use the MXML editor in Design mode to design applications using layout options, to drag components onto the design canvas and then reposition and resize them as needed, to simplify using view states, and so on. See "Building a user interface with Flash Builder" on page 177.
- 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. See "Creating Custom MXML Components" on page 126.
- 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. See "Managing projects" on page 47 and "Creating folders and files in a project" on page 60.
- Build your applications in a variety of ways using the built-in builders or create custom builders using Apache Ant. See "Building projects" on page 69.
- Run your applications in a web browser or the stand-alone Flash Player. Create custom launch configurations to control how your applications run. See "Running your applications" on page 87 and "Managing launch configurations" on page 88.
- Debug your applications using the integrated debugging tools in Flash Builder. See "Debugging your applications" on page 132.

- Use the Network Monitor to generate a detailed audit trail of all data passed between the local Flex application and the back end. See "Monitoring applications that access data services" on page 145.
- Publish your application source code so it can be viewed by users and other developers. See "Publishing source code" on page 80.
- Create library projects that generate shared component library (SWC) files for code reuse and distribution. See "Library projects" on page 66.
- Customize the IDE. For example, you can arrange the interface to include your favorite tools in the specific layout. See "Navigating and customizing the Flash Builder workbench" on page 23.

Flash Builder versions

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

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

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

Flash Builder installers

Flash Builder provides two installers:

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

Stand-alone installer This installer is a customized packaging of Eclipse and the Flash Builder plug-ins created specifically for developing applications that use the Flex framework and ActionScript 3.0. The user interface of the stand-alone installation is more tightly integrated than in the plug-in installation. The standalone installation eliminates much of the potential confusion that the open, multipurpose plug-in installation might create. The stand-alone installation is ideal for new users and users who intend to only develop applications using the Flex framework and ActionScript 3.0.

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

- If you already have Eclipse 3.4.2 (or later) installed, use the Flash Builder plug-in installer to add Flash Builder features to your existing copy of Eclipse.
- If you don't have Eclipse installed and your primary focus is on developing Flex and ActionScript applications, use the Flash Builder stand-alone installer. This installer also allows you to install other Eclipse plug-ins to expand the scope of your development work in the future.

  The Flash Builder plugin and standalone installers provide the same functionality.

Adobe Community Help Client (CHC)

In this release of Flash Builder, Adobe introduces the Adobe Community Help Client (CHC). The CHC is an AIR-based application that replaces the Eclipse help engine for Flash Builder and is the platform for the next generation of Adobe help delivery. CHC features include:

- Always online

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

- Search-centric

  Use Community Help search, adobe.com search, or local search. Community Help search aggregates resources, including those from 3rd party sites. adobe.com search includes refinements to narrow your scope.

- In-context navigation

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

Chapter 2: Flash Builder Workbench

Flash Builder Workbench Basics

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

About the workbench

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

The information you need to use Flash Builder is contained in the Flash Builder documentation. Unless you are using other Eclipse plug-ins (such as CVS or Java) with Flash Builder, or you want to extend the Flash Builder plug-ins (see "Extending the Flash Builder workbench" on page 23), you do not need to be concerned with the underlying Eclipse framework.

Workbench The term *workbench* refers to the Flash Builder development environment. The workbench contains three primary elements: *perspectives*, *editors*, and *views*. You use all three in various combinations at various points in the application development process. The workbench is the container for all of the development tools you use to develop applications. You might equate it to Microsoft Visual Studio, which provides a framework and core functionality for a variety of development tools.

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

If you use the Flash Builder plug-in configuration (see "Flash Builder installers" on page 2), 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 Flash Builder perspectives" on page 7.

Editor An editor allows you to edit various file types. The editors available to you depend on the number and type of Eclipse plug-ins installed. Flash Builder contains editors for writing MXML, ActionScript 3.0, and Cascading Style Sheets (CSS) code. For more information about Flash Builder code editing, see "About code editing in Flash Builder" on page 96.

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

Some views support the core functionality of the workbench itself. For example, the Package Explorer 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, Adobe Dreamweaver®, and other Adobe development tools.

Workspace Not to be confused with *workbench*, a *workspace* is a defined area of the file system that contains the resources (files and folders) that make up your application projects. You can work with only one workspace at a time; however, you can select a different workspace each time you start Flash Builder. For more information, see "Managing projects" on page 47.

Resource The term *resource* is used generically to refer to the files and folders within the projects in a workspace. For more information, see "Creating folders and files in a project" on page 60.

Project All of the resources that make up your applications are contained within projects. You cannot build an application in Flash Builder without first creating a project. You can create three types of projects in Flash Builder: Flex, ActionScript, and Library projects. For more information, see "Working with projects" on page 34.

Launch configuration A *launch configuration* is created for each of your projects, and it defines project settings that are used when running and debugging your applications. For example, the names and locations of the compiled application SWF files are contained in the launch configuration, and you can modify these settings. For more information, see "Running your applications" on page 87.

About Flash Builder editors

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

MXML editor You use the MXML editor to edit MXML and to embed ActionScript and CSS code in `<fx:Script>` and `<fx:Style>` tags. The MXML editor has two modes: Source and Design. Source mode is used for editing code. Design mode is used for visually laying out and designing your applications. The two modes are synchronized and changes in one mode are immediately reflected in the other. For more information, see "About code editing in Flash Builder" on page 96.

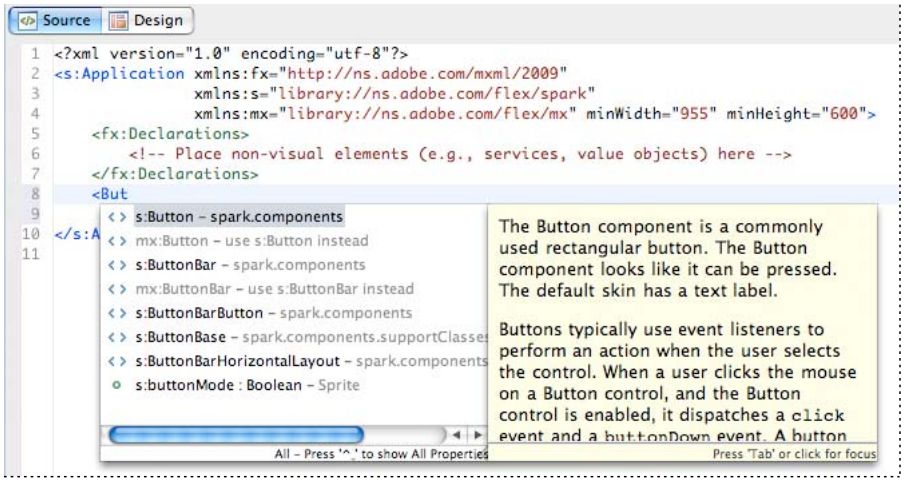
ActionScript editor You use the ActionScript editor to edit ActionScript class and interface files. Although you can embed ActionScript functions into an MXML file by using the `<fx:Script>` tag, it is a common practice to define classes in separate ActionScript files and then import the classes into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

CSS editor You use the CSS editor to display and edit Cascading Style Sheets. You can then apply styles to the visual elements of your applications. For more information, see "Working with components visually" on page 184 and Using Styles and Themes.

Code hinting

Editors contain many features that simplify and streamline code development. Foremost among these features is Content Assist, which displays code completion hints and ASDoc content as you enter MXML, ActionScript, and CSS code. ASDoc content also displays when you hover over a class in the editor.

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 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 `<fx: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 displays ASDoc content for the ActionScript classes and interfaces.

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

For more information, see “About Content Assist” on page 98.

Generating Event Handlers

Flash Builder provides automatic generation of event handler code to simplify associating events with components in your application. In Design mode, select a component and then click the Generate Event Handler button for an event listed in the Events View of the Property Inspector. The editor switches to Source mode, and generates an event handler in the Script section of the source code. The editor also places an event property in the component tag that references the generated event handler.

You can also generate event handlers from Source mode using content assist.

For more information, see “Generating event handlers” on page 194“Generating event handlers for components” on page 102.

Code navigation

Code navigation simplifies the burden of working with code, especially in large projects with many resources. Code navigation includes the ability to select a code element (a reference to a custom component in an MXML application file, for example) and go to the source of the code definition, wherever it is located in the project, workspace, or path.

Other code navigation features include code folding, which allows you to collapse and expand multiline code statements. Another feature is the Outline view, which hierarchically presents, and allows you to navigate to, all user interface and code elements in a file. For more information, see "Navigating and organizing code" on page 103.

Code formatting

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

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

For more information, see "Formatting and editing code" on page 109.

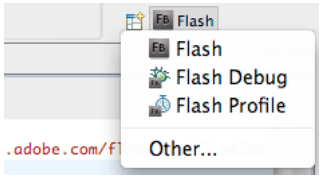
Find references and code refactoring

Flash Builder lets you find all references and declarations to identifiers in a given file, project, or workspace, including references found in elements linked from SWC files and other entries on a library path (for example, classes, interfaces, functions, variables, and some metadata). You use refactoring to rename identifiers in your code while updating all references to them in your entire code base. For more information, see "Finding references and refactoring code" on page 111.

About Flash Builder perspectives

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

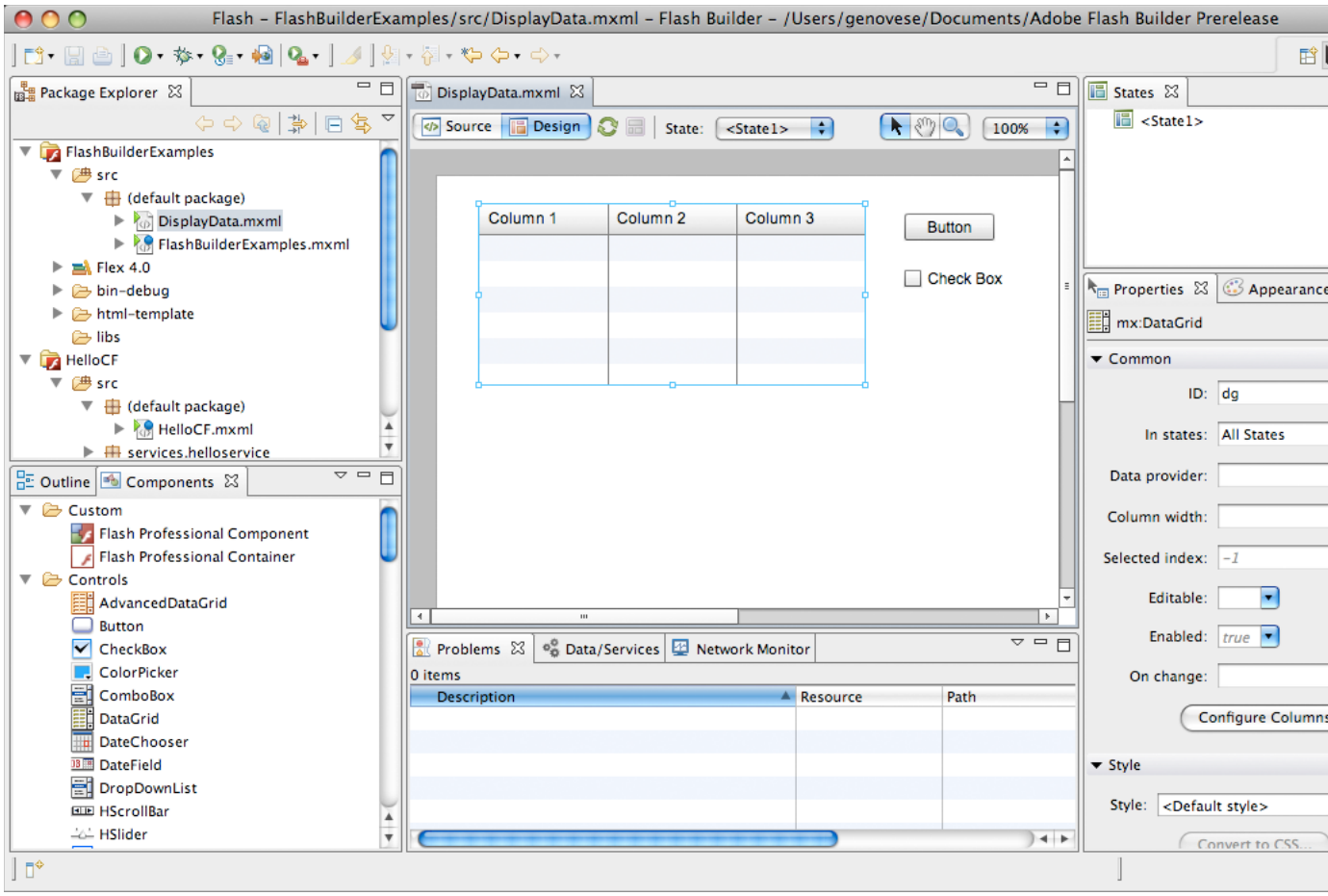
Perspectives change automatically to support the task at hand. For example, when you create a Flex project, the workbench switches to the Development perspective; when you start a debugging session, the Flash Debug perspective is displayed when the first breakpoint is encountered. You can also manually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



If you use the plug-in configuration of Flash 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 23.

The Flash Development perspective

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



The focal point of the perspective (and the workbench generally) is the editor area. The editor area contains all of the currently open documents in a multitab interface. The supporting views are placed around the editor area. Perspectives predefine the layout of all the elements within it, but you may rearrange them to your liking. For more information, see "Navigating and customizing the Flash Builder workbench" on page 23.

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

Flash Builder editors

The editor area contains all of the open documents. When you create a Flex project, the main MXML application file is opened in the editor area. You can then open and switch between any of the MXML, ActionScript, and CSS documents you are working in.

The MXML 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.

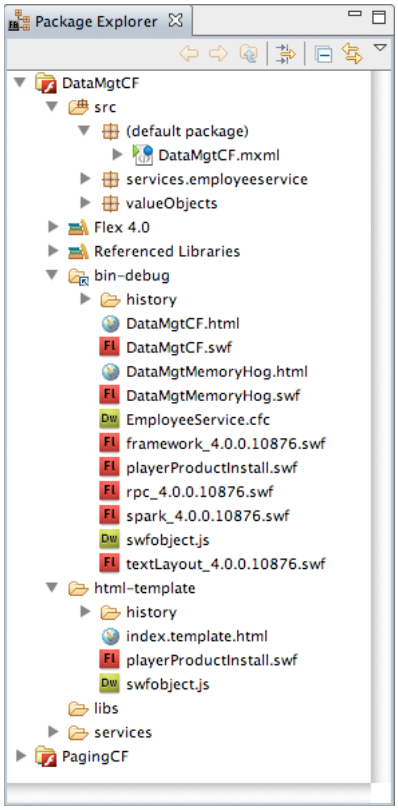
The ActionScript editor is a single-purpose editor that you use to create ActionScript files.

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

For more information about using the MXML editor, see "About code editing in Flash Builder" on page 96 and "Working with components visually" on page 184.

Package Explorer view

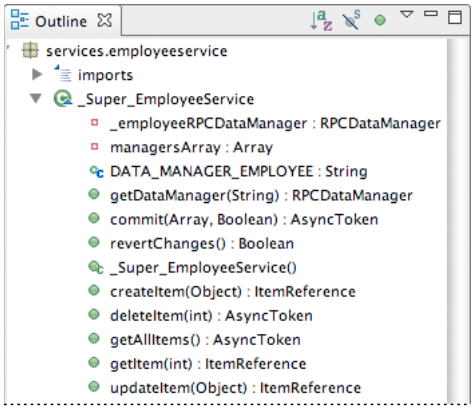
The Package Explorer view contains all of the projects and resources in the workspace and is therefore an essential element of the Flash Builder workbench. It is always displayed in the Development and Debug perspectives.



For more information about the Package Explorer and working with projects, see "Working with projects" on page 34.

Outline view

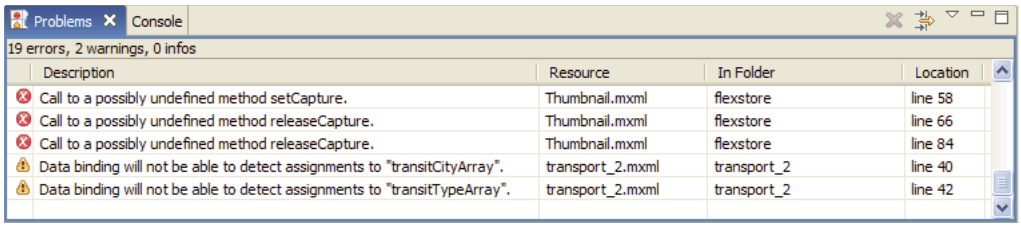
In Source mode, the Outline view presents a hierarchical view of the code structure of the selected MXML or ActionScript document so that you can inspect and navigate the sections or lines of code in the document. The Outline view also displays syntax error alerts that the compiler generates. This view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



For more information about using the Outline view in Source mode, see "Using the Outline view to navigate and inspect code" on page 104.

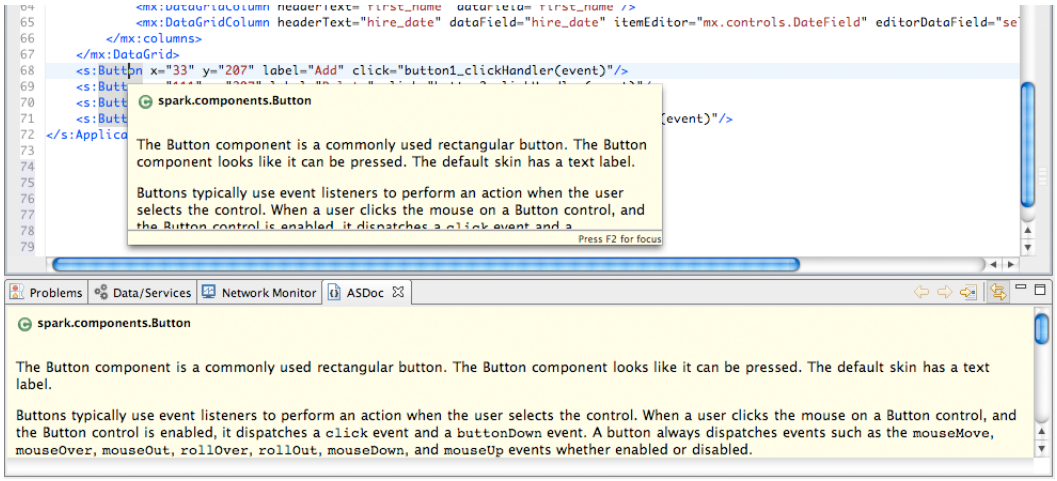
Problems view

As you enter code, the Flash 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.



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 115. For an introduction to the optional views that are available in Flash Builder, see "Other useful workbench views" on page 20.*

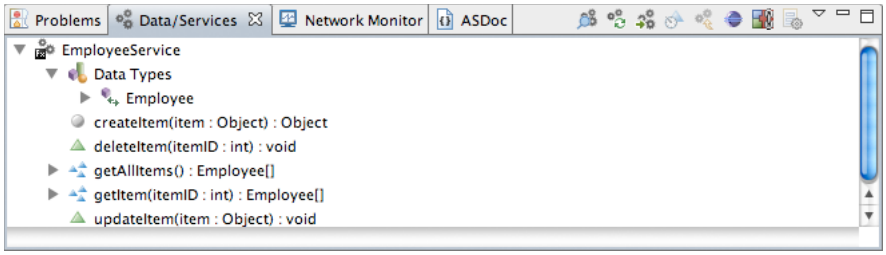
ASDoc view



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

For more information, see "Flash Builder coding assistance" on page 97.

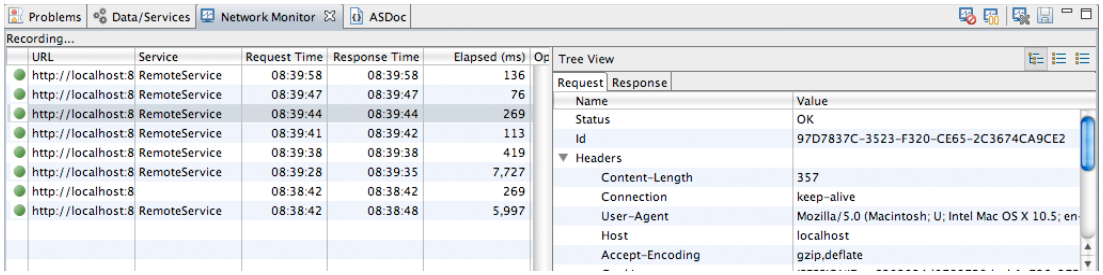
Data/Services view



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

For more information, see Building data-centric applications with Flash Builder.

Network Monitor view



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

For more information, see "Monitoring applications that access data services" on page 145.

Flash Development perspective in Design mode

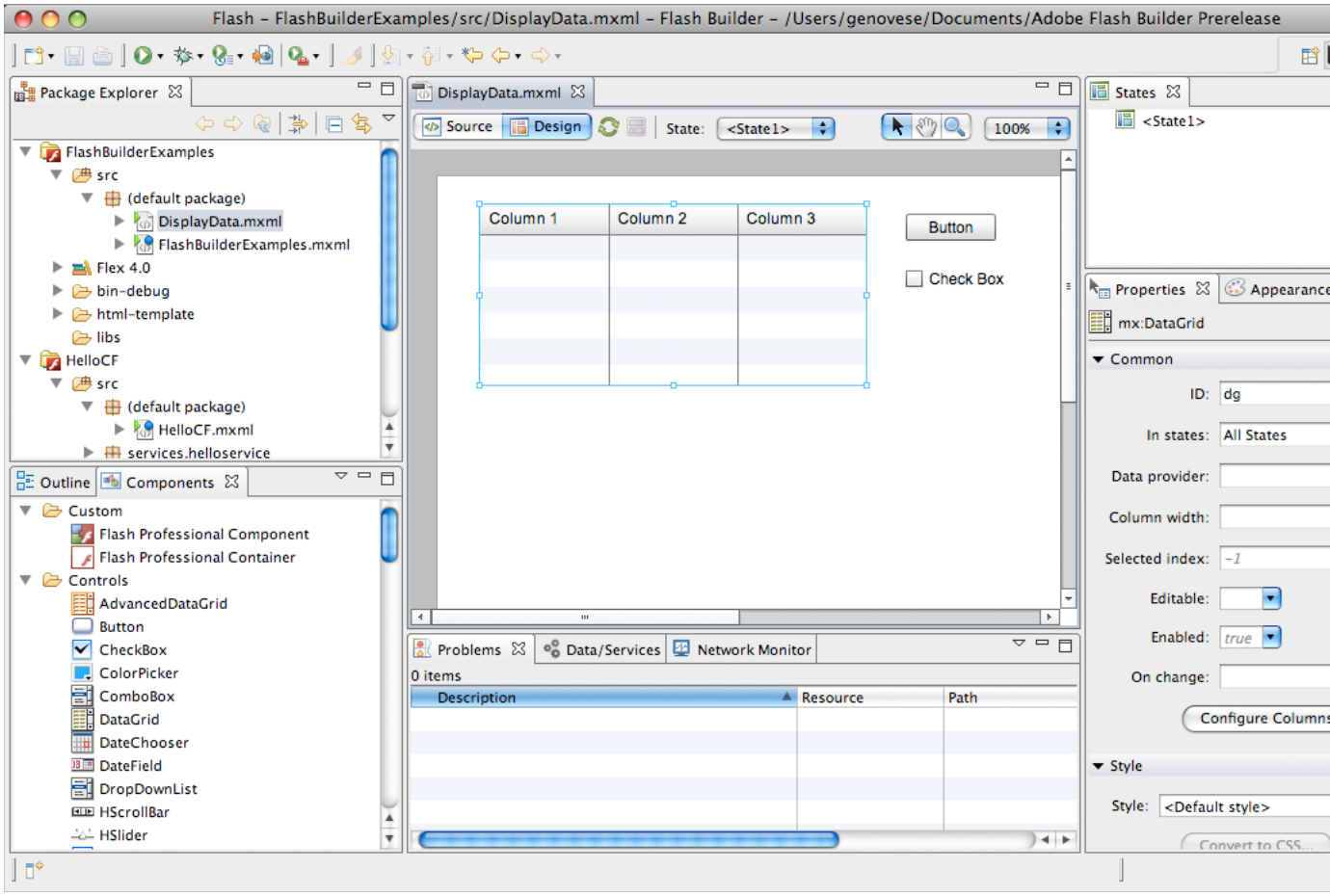
You visually lay out and design your applications in Design mode of the MXML editor. Design mode is the visual representation of the code that you edit in Source mode. In Design mode, however, additional views are added to support design tasks. These are the Components, Properties, Appearance, and States views. In addition, when you are in Design mode, the Outline view displays the MXML structure of your applications. For more information about designing Flex applications in Flash Builder, see "Building a user interface with Flash Builder" on page 177.

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 the applications. For more information about working with ActionScript, see "ActionScript projects" on page 63 and "Running applications" on page 85.*

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

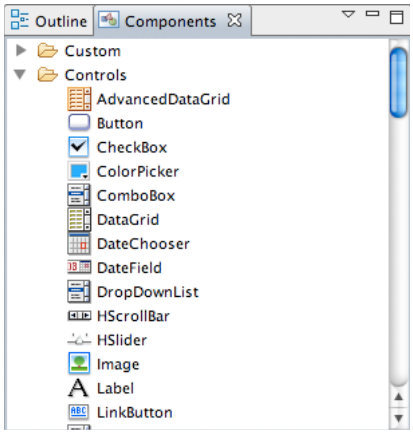
The MXML editor

In MXML Design mode, you interact with your applications visually; dragging and dropping components on to the design area, selecting and resizing components, and so on. You can also expand the MXML editor in Design mode to clearly see and select individual components, and use pan and zoom to get a closer look at items; this is especially useful when you have embedded container components. For more information about working in Design mode, see "Building a user interface with Flash Builder" on page 177.



Components view

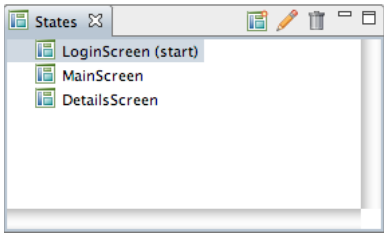
The Components view contains all of the standard Flex components, which you can select and add to the design area. As you create your own custom components, they are also displayed in the Components view.



For more information, see "Adding and changing components" on page 180.

States view

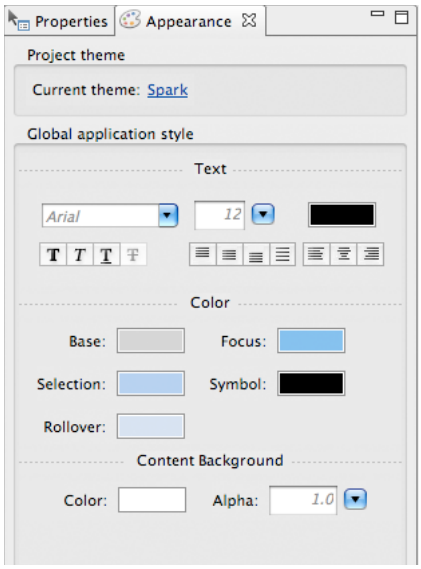
Flex allows you to create applications that change their appearance based on events that are triggered directly by the user or events that are generated programmatically. These user interface transformations are referred to as *view states*. You create and manage view states in the States view.



For more information about view states, see "Adding View States and Transitions" on page 212.

Appearance view

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

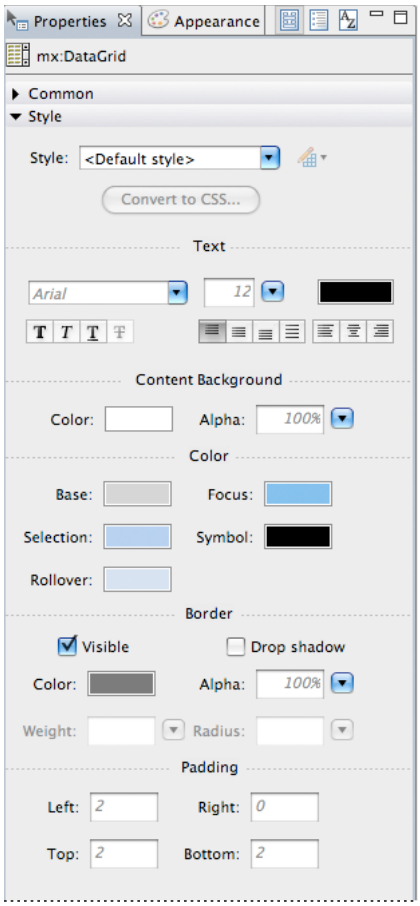


Appearance panel

For more information, see "Apply styles to an application" on page 200.

Styles view

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

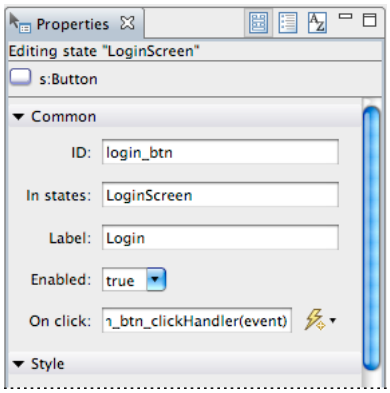


Style panel

For more information, see "Apply styles to an application" on page 200.

Flex Properties view

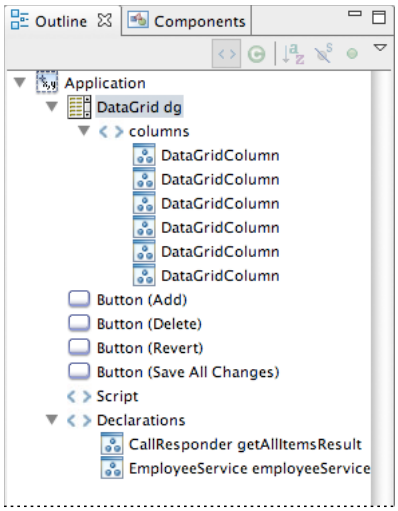
When a Flex component is selected, its properties are displayed in the Properties view. You can set and edit properties as appropriate. You can view a component's properties graphically (as shown in the following example) and as a categorized or alphabetical list.



For more information, see "Setting component properties" on page 188.

Outline view

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

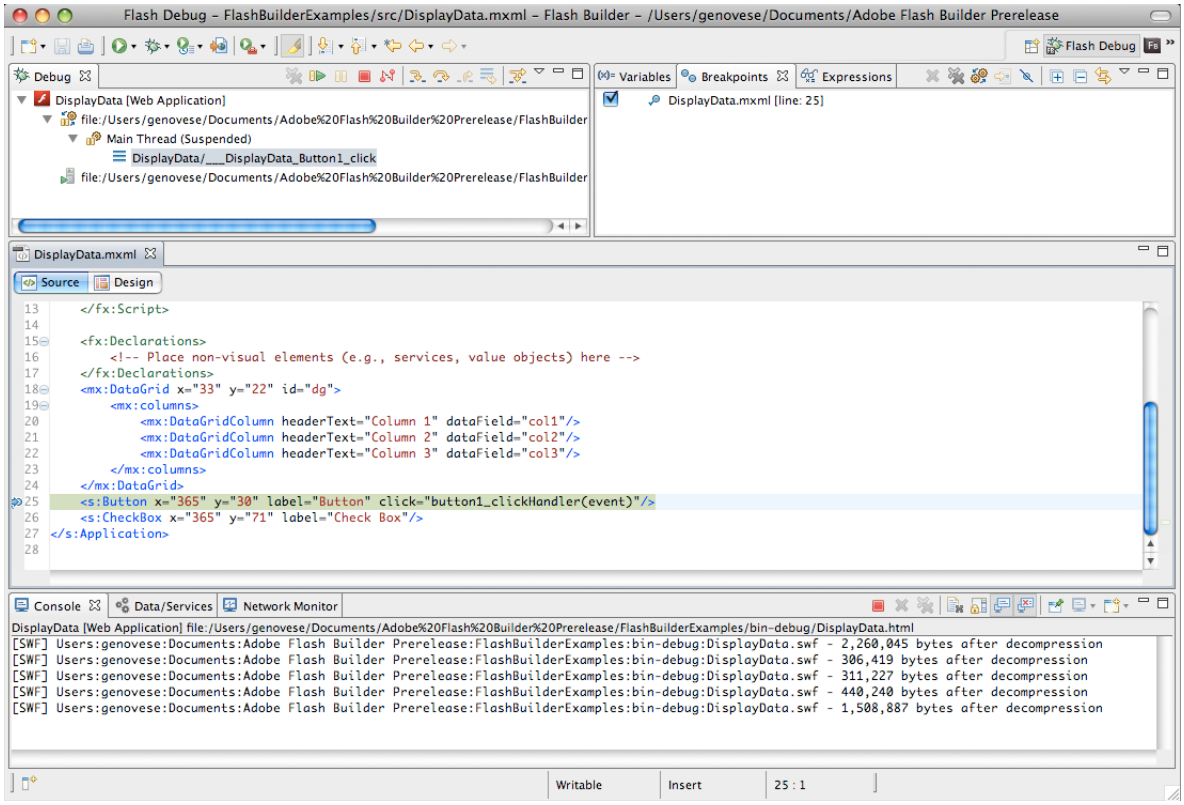


For more information about working with the Outline view in Design mode, see "Inspecting the structure of your MXML" on page 190.

The Flash Debug perspective

The Flash Debug perspective contains the tools you need to debug your applications. Like the Development perspective, the primary tool within the Debug 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 code to stop execution 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.

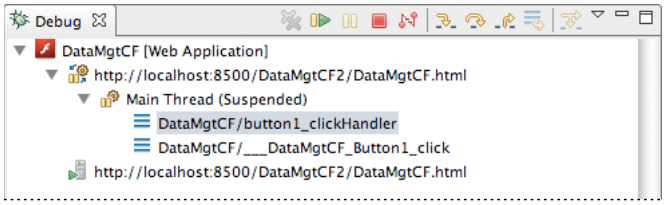


The Debug perspective appears automatically when the first breakpoint is reached. You can also switch to the Debug perspective manually by selecting it from the Perspective bar, which is located at the right edge of the main workbench toolbar.

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

Debug view

The Debug view (in other debuggers this is sometimes referred to as the *callstack*) displays the stack frame of the suspended thread of the 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.



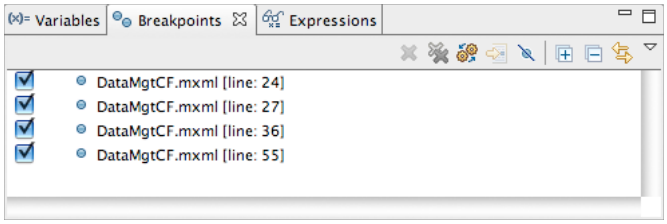
For more information about working with the Debug view, see "Managing the debugging session in the Debug view" on page 135.

Applications built with Flex are single-threaded (not multithreaded like Java, for example) and you can debug only one application at a time. Therefore, when you debug an application, you see only the processes and Debug view for a single thread of execution.

The Debug view shows a list of all the functions called to that point, in the order called. For example, the first function called is at the bottom of the list. You can double-click a function to move to it in the script; Flash Builder updates the information in the Variables view to reflect the new location in the script.

Breakpoints view

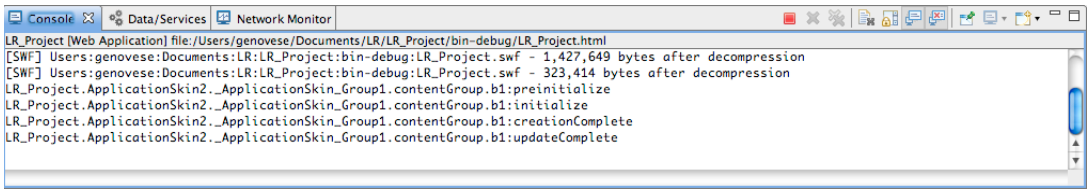
The Breakpoints view lists all of the breakpoints you set in your project. You can double-click a breakpoint and display its location in the editor. You can also disable, skip, and remove breakpoints.



For more information, see "Managing breakpoints in the Breakpoints view" on page 134.

Console view

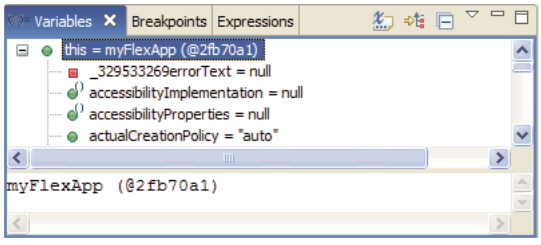
The Console view displays the output from trace statements placed in your ActionScript code and also feedback from the debugger itself (status, warnings, and errors).



For more information, see "Using the Console view" on page 135.

Variables view

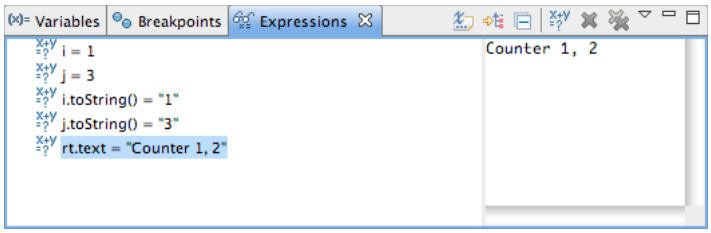
The Variables view displays information about the variables in a selected stack frame. You can select variables to monitor (in the Expressions view) and also change variable values during the debugging session. During the debug session you can see the changes in the currently running SWF file and experiment with fixes for the problem you need to resolve.



For more information, see "Managing variables in the Variables view" on page 136.

Expressions view

The Expressions view is used to monitor a set of critical variables. You can choose the variables you consider critical in the Variables view and add them to the Expressions view for monitoring. You can also add and evaluate watch expressions.



When you debug your application, you can monitor variables and, if needed, modify the values. You can add and remove variables in the Expressions view. Watch expressions are code expressions that are evaluated whenever debugging is suspended. For more information, see "Using the Expressions view" on page 137.

For more information about debugging Flex and ActionScript applications, see "Debugging your applications" on page 132.

The Flash Profiling perspective

Flash Builder Premium contains an additional perspective. The Adobe Flex profiler helps you identify performance bottlenecks and memory leaks in your applications. The Profiling perspective displays several panels (or views) that present profiling data in different ways. As you interact with your application, the profiler records data about the state of the application, including the number of objects, the size of those objects, the number of method calls, and the time spent in those method calls. For more information about the profiler, see "About profiling" on page 148.

Other useful workbench views

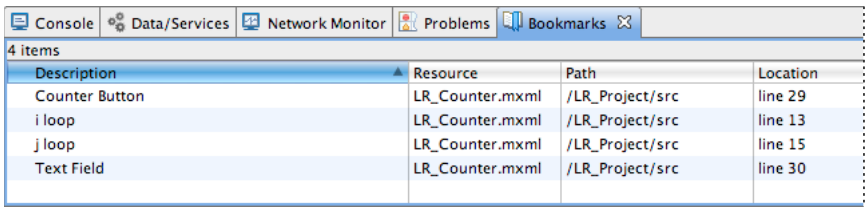
In addition to the editors and views associated with Flash Builder's default development, debugging, and profiling perspectives, the workbench contains other views that help you streamline the application development process.

You can access views that are not already displayed with a perspective and add them to the workbench by selecting Window > Other Views > General (Window > Show View Other in plugin version). These optional views are categorized by type and are associated with distinct workbench functionality or with specific Eclipse plug-ins. For more information about working with views, see "Working with editors and views" on page 25.

Several workbench views in particular are valuable aids as you develop your applications in Flash Builder. These include the Tasks, Bookmarks, and Search views.

Bookmarks view

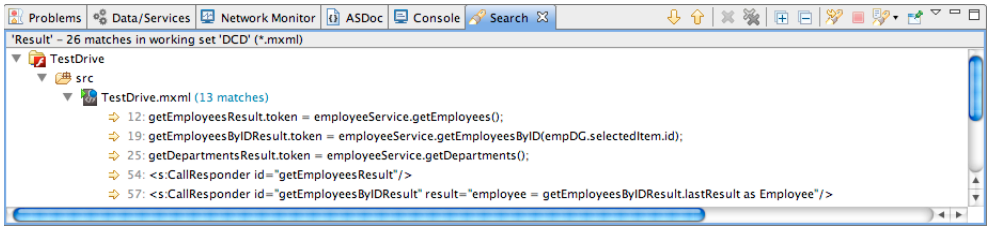
The Bookmarks view is used to manage the bookmarks that you add to specific lines of code or to resources. As in a web browser, bookmarks are used as a convenience for keeping track of noteworthy items. Selecting a bookmark locates and displays it in the workbench.



For more information about the Bookmarks view, see "About markers" on page 115.

Search view

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



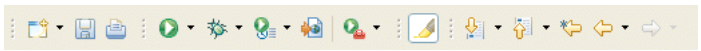
For more information about the Search view, see "Searching in the workbench" on page 31.

Workbench menus, toolbars, and shortcuts

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

The workbench toolbar

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



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

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

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

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

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

Run Opens the main application SWF file in your default web browser or directly in stand-alone Flash Player. You can also select other application files in the project from the attached pop-up menu. For more information, see "Running your applications" on page 87.

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 132.

Profile Creates, manages, and runs configurations. For more information, see "About profiling" on page 148.

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

External Tools Selects a custom launch configuration.

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

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

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

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

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

The MXML editor toolbar

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

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

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

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

Refresh Reloads the visual elements (images, SWF files, or class files containing drawing API methods) that define the visual design of your application. Collectively, these elements are referred to as a *skin*. For more information, see Creating Skins

State Pop-up menu displays all the defined views states. Selecting view states updates the display of the visual design. For more information, see "Adding View States and Transitions" on page 212.

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

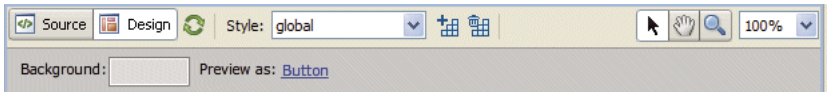
Pan Mode Allows you to pan and scroll around in design area; items cannot be selected or moved in Pan mode.

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

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

The CSS editor toolbar (Flex 3 only)

If you are editing a CSS file for an application project that uses the Flex 3 SDK, then the CSS editor is available in Design mode. The CSS editor contains several buttons that allow you to control the editor in Source and Design modes. To see the CSS editor toolbar, create a project that uses the Flex 3 SDK and open a CSS file in Design mode.



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

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

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

Refresh Reloads the visual elements (images, SWF files, or class files containing drawing API methods) that define the visual design of your application. Collectively, these elements are referred to as a *skin*. For more information, see Creating Skins.

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

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

Delete Style Deletes the selected style for your CSS file.

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

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

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

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

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

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

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

Using keyboard shortcuts

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

Display the list of all keyboard shortcuts in Flash Builder

❖ Select Help > Key Assist.

You can use Key Assist as a reference to all the Flash 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 30.

Extending the Flash Builder workbench

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

Navigating and customizing the Flash Builder workbench

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

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

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 Flash Debugging perspective to debug your application.

For an overview of perspectives, see "About Flash Builder perspectives" on page 7.

Opening and switching perspectives

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

- Flash Development
- Flash Debugging
- Flash Profiling

The Flash Profiling perspective is available with Flash Builder Premium.

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

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

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

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

Setting the default perspective

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

1 Open the Preferences dialog and select General > Perspectives.

2 Under Available Perspectives, select the perspective to define as the default, and click Make Default.

3 Click OK.

Opening perspectives in a new window

You can specify to open perspectives in a new window.

1 Open the Preferences dialog and select General > Perspectives.

2 Under Open a New Perspective, select In A New Window.

To switch back to the default, select In The Same Window.

3 Click OK.

Customizing a perspective

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

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

Create a new perspective

1 Open an existing perspective.

2 Show views and editors as desired.

For more information, see "Opening views" on page 26, and "Opening files for editing" on page 28.

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

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

Configure a perspective

1 Open the perspective to configure.

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

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

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

Deleting a customized perspective

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

1 Open the Preferences dialog and select General > Perspectives.

2 Under Available Perspectives, select the perspective you want to delete.

3 Click Delete, then click OK.

Resetting perspectives

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

1 Open the Preferences dialog and select General > Perspectives.

2 Under Available perspectives, select the perspective to reset.

3 Click Reset, then click OK.

Working with editors and views

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

For an overview of editors and views, see "About the workbench" on page 4.

Opening views

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

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

After you add a view to the current perspective, you can save that view as part of the perspective. For more information, see "Customizing a perspective" on page 24.

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 26.

Moving and docking views

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

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

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

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

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

2 (Optional) Save your changes by selecting Window > Perspectives > Save Perspective As (Window > Save Perspective As in the plug-in configuration of Flash Builder).

Rearranging tabbed views

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

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

Switching between views

There are several ways to switch to a different view:

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

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

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

Creating and working with fast views

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

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

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

Create a fast view

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

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

Restore a fast view to normal view

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

Filtering the Tasks and Problems views

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

1 In Tasks or Problems view taskbar, click Filter.

2 Complete the Filters dialog box and click OK.

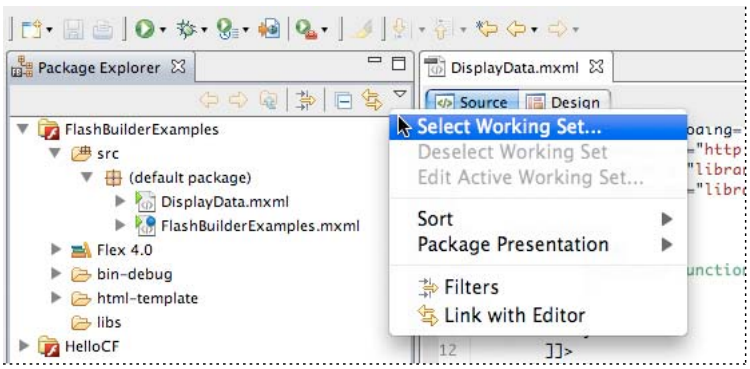
For more information about views, see "Flash Builder Workbench Basics" on page 4.

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

Create a working set

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



2 Select New.

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

3 Select the resources type and click Next.

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

5 Click Finish.

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

Display all projects in the workspace

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

Opening files for editing

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

❖ Do one of the following:

- From the context menu for the file in one of the navigation views, select Open.
- 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, select Open With from the context menu for the file. Select the editor to use.

Associating editors with file types

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

1 Select Window > Preferences.

2 Click the plus button to expand the General category.

3 Click the plus button to expand the Editors category, and then select File Associations.

4 Select a file type from the File Types list.

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

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

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

6 Click OK.

You can override the default editor preferences from the context menu for any resource in one of the navigation views. Select 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 Flash Builder. The workbench performs any necessary build or update operations to process the changes that you made to the file outside the workbench.

Refresh an MXML or ActionScript file edited outside the workbench

1 Edit the MXML or ActionScript file in the external editor of your choice.

2 Save and close the file.

3 Start Flash Builder.

4 From one of the navigation views in the workbench, select Refresh from the context menu.

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

Tiling editors

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

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

   The pointer changes to a drop cursor, indicating where the editor will appear when you release the mouse button.
4. (Optional) Drag the borders of the editor area of each editor to resize the editors as desired.

Maximizing a view or editor

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

Maximize (restore) a view or editor

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

Switching the workspace

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

Customizing the workbench

You can customize the workbench to suit your individual development needs. For example, you can customize how items appear in the main toolbar, create keyboard shortcuts, or alter the fonts and colors of the user interface.

Rearranging the main toolbar

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

1. Ensure that the toolbar is unlocked. From the context menu for the toolbar, deselect Lock the Toolbars.
2. Move the mouse pointer over the vertical line "handle" that is on the left side of the toolbar section you want to rearrange.
3. Click the handle and drag the section left, right, up, or down. Release the mouse button to place the section in the new location.

To prevent accidental changes, lock the toolbar again from the toolbar context menu.

Changing keyboard shortcuts

1 Open the Preferences dialog and select General > Keys.

2 In the View screen of the Keys dialog box, select the command you want to change.

3 In the Binding field, type the new keyboard shortcut you want to bind to the command.

4 In the When pop-up menu, select when you want the keyboard shortcut to be active.

5 Click Apply or OK.

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

Changing fonts and colors

By default, the workbench uses the fonts and colors that your computer's operating system provides. However, you can customize fonts and colors in a number of ways. The workbench lets you configure the following fonts:

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

Dialog font Appears in widgets and dialog boxes.

Header font Appears as a section heading.

Text font Appears in text editors.

CVS Console font Appears in the CVS console.

Ignored Resource font Displays resources that CVS ignores.

Outgoing Change font Displays outgoing changes in CVS.

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

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

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

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

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

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

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

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

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

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

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

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

Changing fonts and colors

1 Open the Preferences dialog and select General > Appearance > Colors and Fonts.

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

Note: *You can also click Use System Font instead of Change to set the font to a reasonable value that the operating system chooses. For example, in Windows, selecting this option causes Flash Builder to use the font selected in the Windows Display Properties control panel.*

3 Set the font and color preferences as desired.

Changing colors

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

Change colors

1 Open the Preferences dialog and select General > Appearance > Colors and Fonts.

2 Expand the Basic, CVS, Debug, Text Compare, or View and Editor Folders categories to locate and select the color to change.

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

4 Select a new color.

Controlling single- and double-click behavior

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

1 Open the Preferences dialog and select General.

2 In the Open Mode section, make your selections and click OK.

Searching in the workbench

Flash Builder provides a search tool that lets you quickly locate resources. For more information about searching for text in a particular file, see "Finding and replacing text in the editor" on page 110.

Searching for files

Flash Builder lets you conduct complex searches for files.

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

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

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

Searching for references and declarations

Flash Builder includes advanced search features that are more powerful than find and replace. To help you understand how functions, variables, or other identifiers are used, Flash Builder lets you find and mark references or declarations to identifiers in ActionScript and MXML files, projects, or workspaces. For more information, see "Finding references and refactoring code" on page 111.

Using the Search view

The Search view displays the results of your search.

Open a file from the list

❖ Double-click the file.

Remove a file from the list

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

Remove all files from the list

❖ Click Remove All Matches.

Navigate between matched files

❖ Click Show Next Match or Show Previous Match.

View previous searches

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

Return to the Search view after closing it

1 Select Window > Other Views > General. (Window > Show View > Other in the plug-in configuration of Flash Builder.)

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

Working in the editor's Source and Design modes

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

View your file in Design mode

❖ Click Design at the top of the editor area.

View your file in Source mode

❖ Click Source at the top of the editor area.

Work in both Source and Design modes simultaneously

1 From the option menu on the editor's tab, select New Editor.

You now have two editor tabs for the same file.

2 Drag one of the tabs to the right to position the editor windows side-by-side.

3 Set one of the editors to Design mode, and set the other editor to Source mode.

Switch between the Source and Design modes

❖ Press Control+` (Left Quote).

Accessing keyboard shortcuts

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

❖ Select Help > Key Assist.

Setting workbench preferences

You can set preferences for many aspects of the workbench. For example, you can specify that Flash 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 applications.

Your Flash Builder preferences apply to the current workspace only. You can, however, export your workbench preferences and then import them into another workspace. This may be helpful if you are using multiple workspaces, or if you want to share your workbench preferences with other members of your development team.

You can also set preferences for individual projects within a workspace. For example, you can set separate compiler or debugging options for each of your Flex projects.

Set Flash Builder workbench preferences

1 Open the Preferences window.

2 Expand General and select any of the categories of workbench preferences and modify them as needed.

3 Click OK.

Chapter 3: Working with projects

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

About Flash Builder projects

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

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

Applications deployed to Flash Player

Use the New Flex Project wizard to create applications that can be deployed to the Flash Player. When creating the project, specify the application type as Web (runs in Adobe Flash Player). These applications are compiled into stand-alone SWF files. For more information, see "Working with projects" on page 34 and "ActionScript projects" on page 63.

Applications deployed to Adobe AIR

Use the New Flex Project wizard to create applications that can be deployed to Adobe® AIR®. When creating the project, specify the application type as Desktop (runs in Adobe AIR). Use the Export Release Build feature to generate a release-quality, installable AIR package. For more information, see "Developing AIR applications with Flash Builder" on page 129.

With Flash Builder you can debug, package, and manage AIR projects. Flash Builder enables you to run applications in AIR.

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

Flex Library Projects

You also use Flash Builder to build custom code libraries that you share between your applications or distribute to other developers. A library project generates a SWC file, which is an archive file for Flex components and other resources. For more information, see "Library projects" on page 66.

Applications contained in projects

To begin building an application in Flash Builder, you must first create a project. Specify whether the application is a Web application (runs in Flash Player) or Desktop application (runs in AIR). When you create a Flex project, a main application file is created for you. Then you add other resources such as MXML application files, custom MXML component files, ActionScript files, and other assets that make up your application. When you create an ActionScript project, a main ActionScript file is created; then you can build an application by using ActionScript and the Flash Player API. For more information, see "Creating Flex projects" on page 40 and "Managing projects" on page 47.

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; Flash Builder automatically links them to the workspace. When you switch workspaces, Flash Builder restarts.

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 Package Explorer, and you can manage them as needed. You can add resources, organize your projects into folders, and build projects in the workspace. For more information, see "Managing projects" on page 47 and "Creating folders and files in a project" on page 60.

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 61.

More than one application in a project

Flash Builder lets you define more than one file in your project as an application. When you create a project, Flash Builder generates a main application file that serves as the entry point into your application, and the compiler uses this file to generate the application SWF file. However, if your project is complex, you can create additional application files. All application files must reside in the src folder under the root folder of your project. For more information, see "Managing project application files" on page 50.

Support for Multiple Flex SDKs

You could have projects that are in progress or an older project code base that must be maintained. With Flash Builder, you can work with different versions of the Flex SDK. To specify the installed SDKs, you configure the Flash Builder workspace, which provides a default SDK for any project. After you set up a project, you can add, remove, or edit SDK configurations in the Preferences dialog by selecting Flex > Installed SDKs. You can also modify the SDK configurations by selecting Project > Properties > Flex Compiler. For more information, see "Using multiple SDKs in Flash Builder" on page 78.

Automatic project builds

By default, your project is automatically built any time you save changes to a file. You have complete control over how and how often your applications are built. If you have no special requirements for customizing the build, it works transparently and automatically generates the application SWF files. For more information, see "Building projects" on page 69.

Export Release Build

When your application is ready to deploy, you use the Export Release Build wizard to create a release-quality *non-debug* version of your application. The wizard copies required assets to a bin-release folder separate from the debug version. You can specify whether or not to include the application's source code. The exported application is an optimized production build that can be viewed by end users. For Adobe AIR projects, AIR applications are exported to an AIR file. You use Export Release Build to create a digitally signed AIR file, which users install before running an application.

Custom Ant scripts

Apache Ant is a Java-based build tool that you use to create custom scripts for building applications in Flash Builder. You use Ant to modify and extend the standard build process. For more information, see "Customizing builds with Apache Ant" on page 78.

Command line build

With Flash Builder Premium, you can implement command line builds. Use command line builds to synchronize a developer's individual build settings with a nightly build. For more information, see "Flash Builder command line build" on page 81.

Project types

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

Flex projects

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

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

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

ColdFusion This project configuration lets you create projects that use ColdFusion with LiveCycle Data Services, BlazeDS, or ColdFusion Flash Remoting. If none of these options is selected, a ColdFusion project is created with an output folder under web root (or virtual folder). Flash Builder provides tools for accessing remote data from ColdFusion data sources. For more informations, see Building data-centric applications with Flash Builder.

J2EE This project configuration lets you create projects that use J2EE for deployment. You specify whether to use the remote object access service and either LiveCycle Data Services or BlazeDS. If you do not select a remote object access service, an output folder is created under the Java application server root. If you select the Use Remote Object Access Service option, you can use Flex with LiveCycle Data Services or BlazeDS. Your project is deployed to a LiveCycle Data Services or BlazeDS server. With the Eclipse Web Tools Project (WTP) plug-in installed, you select the Create Combined Java/Flex Project Using WTP option to create combined Java/Flex projects with or without remote object access service. For locally compiled projects with WTP, projects are deployed on your J2EE server.

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

PHP This project configuration lets you create Flex projects that have a Flex output folder under the Apache/IIS web root (or virtual folder). Flash Builder provides tools for accessing remote data from PHP servers. For more informations, see Building data-centric applications with Flash Builder.

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

Web services and HTTP services For each application type, you can create applications that can access web services and HTTP services. Flash Builder provides tools for accessing web services and HTTP services. For more informations, see Building data-centric applications with Flash Builder.

ActionScript projects

Based on the Flash API, not the Flex framework, ActionScript projects let ActionScript developers use Flash Builder to code, build, and debug ActionScript-only applications. Because these projects do not use MXML to define a user interface, you cannot view the application layout and design in Design mode. You work exclusively in the source editor, the debugging tools as necessary, and then build the project into SWF application files to preview and test your application in a web browser or stand-alone Flash Player. For more information about ActionScript projects, see "ActionScript projects" on page 63.

Library projects

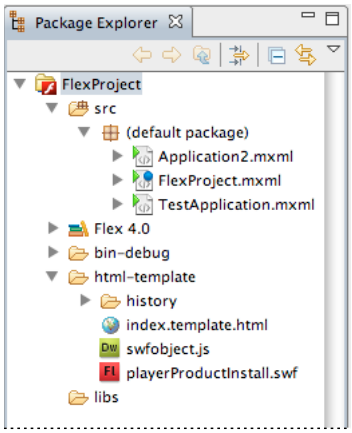
Library projects are used to package and distribute components and other resources. They generate SWC files that you add to other projects or distribute to other developers. For more information, see "Library projects" on page 66.

Flash Professional projects

Use Flash Professional projects to edit, build, or debug FLA or XFL files created in Adobe Flash Professional CS5. Flash Professional projects are only available if you have Flash Professional CS5 installed. For more information, see "Creating Flash Professional projects" on page 46.

Projects in the Package Explorer

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



Highlights of the Package Explorer include:

- Displaying ActionScript packages in either a hierarchical or flat presentation.

  Use the Package Explorer's menu to specify the package presentation.

- Project libraries are represented in two top-level nodes, one node for the Flex SDK and the other for referenced libraries.

  You can expand a library's contents and open editors to view attachments.

- Error and warning badges on Package Explorer nodes notify you of problems within a package.

- You can limit which projects and resources are visible.

  You can create a working set (a collection of resources), create display filters, and sort resources by name and type. These options are available from the Package Explorer menus. For more information about modifying views, see "Navigating and customizing the Flash Builder workbench" on page 23.

- You can expand ActionScript, MXML, and CSS files and see a tree view of their contents.

From the Package Explorer, you can open the project resources for editing. For example, you can edit MXML and ActionScript in `<fx:Script>` blocks and CSS in `<fx:Style>` blocks, or you can switch to Design mode and visually manipulate components and controls to create the application's layout and behavior. For more information about working with the Flash Builder editors, see "About code editing in Flash Builder" on page 96 and "Building a user interface with Flash Builder" on page 177.

Then you add projects, files, and folders, and organize and manage them as needed (see "Creating folders and files in a project" on page 60).

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

For more information about working with projects in the Package Explorer, see "Managing projects" on page 47 and "Creating folders and files in a project" on page 60.

Creating projects and opening resources

Flash Builder provides the wizards to help you create Flex projects, ActionScript projects, and Flex Library projects. The following table describes the projects. To create a project, select File > New.

Project type	Description
ActionScript Project	An ActionScript project based on the Flash API, not the Flex framework. ActionScript projects let ActionScript developers use Flash Builder to code, build, and debug ActionScript-only applications. For more information, see "Creating ActionScript projects" on page 64
Flex Project	A Flex project contains a set of properties that control how the application is built, where the built application resides, how debugging is handled, and the relationships to other projects in the workspace. In a Flex project you can create applications that run in Flash Player (web applications) or Adobe AIR (desktop applications). For more information, see "Setting Flex project properties" on page 47.
Flex Library Project	Flex Library Projects are used to package and distribute components and other resources. They generate SWC files that you add to other projects or distribute to other developers. For more information, see "Library projects" on page 66.

Project resources

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

Resource type	Description
ActionScript Class	An ActionScript class file. When you add this type of resource, the New ActionScript Class wizard prompts you for class definition elements, such as the superclass, interfaces, and so on. For more information about working with ActionScript in Flash Builder, see "Creating an ActionScript class" on page 64.
ActionScript File	A text file template for creating ActionScript functions.
ActionScript Interface	An ActionScript interface file. When you add this type of resource, the New ActionScript Interface wizard prompts you for interface definition elements such as extended interfaces and the package in which they reside. For more information about working with ActionScript in Flash Builder, see "Creating an ActionScript interface" on page 65.
CSS File	A text file template for creating a Cascading Style Sheets file.
File	An unformatted text file. For more information, see "Creating folders and files in a project" on page 60.
Folder	A standard file system folder for organizing the contents of your projects. For more information, see "Creating folders and files in a project" on page 60.
MXML Application	A standard application file with the `<s:Application>` tag as the root MXML element (for Flex 4 projects). You can specify which layout to use for the application. A Flex project can have more than one application file. For more information, see "Managing project application files" on page 50.
MXML Component	A standard component file with the `<s:Group>` tag as the root MXML element. The wizard allows you to specify an alternate root MXML element. For more information, see "Creating MXML components using Flash Builder" on page 126.
MXML Item Renderer	Item renderers control the appearance of a data item in a DataGroup, SkinnableDataContainer, or in a subclass of those containers. The appearance can include the font, background color, border, and any other visual aspects of the data item. For more information, see "Generating custom item renderers" on page 210.
MXML Skin	Skin classes modify the appearance of controls in a user interface. The way you create, edit, and import skins differs for Spark components and MX components. For more information, see "Modifying user interfaces using skins" on page 204.
MXML Module	A resource that can be added to an existing application project or created separately, but always associated with one application. For more information, see "Creating modules in Flash Builder" on page 90.
Test Case Class Test Suite Class	FlexUnit test cases and test suites. You can generate and edit repeatable tests that can be run from scripts or directly within Flash Builder. For more information, see "FlexUnit test environment" on page 140.
Package	Create a new package for project source files. The default location is under the `src` directory in your project folder.
Other	Other file types that are registered in Flash Builder. Select File > New > Other to add any other file types. For example, if you have a Java plug-in installed in Flash Builder, you can add new Java classes, interfaces, and packages. When a file type is registered in Flash Builder, a corresponding editor is also available in the workbench. For more information, see "Associating editors with file types" on page 28. You can always add unregistered file types to your projects by importing them. For more information see "Importing projects" on page 51

For more information about adding resources to your projects, see "Creating folders and files in a project" on page 60.

Creating Flex projects

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

For information about creating an ActionScript project, see "Creating ActionScript projects" on page 64. For information about creating library projects, see "Library projects" on page 66.

Creating a Flex project with no server

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

1. Select File > New > Flex Project.
2. Enter a project name.
3. Select the project location. The default location is the current workspace. To choose a different project location, deselect the Use Default Location option.
4. Choose the application type (web or desktop).
5. For application server type, choose None.
6. Click Finish, or click Next to select more configuration options.

   "Additional project configuration options" on page 44

Creating a Flex project with ASP.NET

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

1. Select File > New > Flex Project.
2. Enter a project name.
3. Specify the project location. The default location is the current workspace. On Windows platforms, the default workspace is C:\Documents and Settings*Flex Developer*\Adobe Flash Builder\. To choose a different project location, deselect the Use Default Location option.
4. Choose the application type (web or desktop).
5. For application server type, choose ASP.NET.
6. Click Next.
7. Select the ASP.NET server:
   - If you are using an ASP.NET Development Server, there is no need to specify a server location.
   - If you are using IIS, enter the Web Application Root and Web Application URL.
   - Specify the output folder for your Flex application.
8. Click Finish, or click Next to select more configuration options.

   "Additional project configuration options" on page 44

Creating a Flex project with J2EE

This project configuration lets you create Flex projects that use a J2EE servlet with the remote object access service option. When no option is selected, and Java server is used, an output folder is created under the server root. If you installed the Eclipse Web Tools Project (WTP) plug-in, you can create combined Java and Flex projects with or without remote object access service.

Note: *LiveCycle Data Services ES and BlazeDS support specific versions of the Flex SDK. Check the LiveCycle Data Services Compatibility Matrix to see which versions of the Flex SDK your version of LiveCycle DataService ES supports. The compatibility matrix also lists the versions of the Flex SDK that BlazeDS supports.*

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

1. Select File > New > Flex Project.
2. Enter a project name.
3. Specify the project location. The default location is the current workspace. To choose a different project location, deselect the Use Default Location option.
4. Choose the application type (web or desktop).
5. For application server type, choose J2EE.
6. Select the Use Remote Object Access Service option. LiveCycle Data Services ES is automatically selected. You can select BlazeDS. If you installed WTP, you can also choose to create a combined Java and Flex project that uses WTP (the Java source folder is selected for you).
7. Click Next.
8. Configure the J2EE server.
   - If you selected the Use Remote Access Service and LiveCycle Data Services or BlazeDS options, specify the root settings:

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

     Root URL This is the valid root URL of the Flex server (web application) that serves your application. For LCDS, the default root URL for local server instances is http://localhost:8400/lcds/. If you use a remote server, the URL might look like this: http://*myserver.com*:8400/lcds/.

     Context Root The context root typically matches the last segment of the root URL path.
   - If you selected the Create Combined Java/Flex Project Using WTP option (with or without LiveCycle Data Services):
     - Specify the names of your Java and Flex source folders and target runtime.

       When you create a Flex project with LiveCycle Data Services ES, Flash Builder either creates a directory with the same name as your project, or uses an existing directory with that name. That directory is a subdirectory of the root folder that you specified for the project.
     - With LiveCycle Data Services ES, specify a flex.war file, which is located in the server installation folder.

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

9 Specify the location to compile your project.

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

10 Click Finish, or click Next to select more configuration options.

"Additional project configuration options" on page 44

Creating a Flex project that accesses ColdFusion services

To access data that uses ColdFusion, you must have either Adobe ColdFusion® 8 or Adobe ColdFusion 9. For more information, see the ColdFusion product page.

1 Select File > New > Flex Project.

2 Enter a project name.

3 Specify the project location.

Typically you accept the default location, which is the current workspace.

4 Specify the application type (web or desktop).

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

Use Remote Object Access Service If you deselect Use Remote Object Access Service, specify your Web Root and Web Root URL in the next step.

If you select Use Remote Object Access Service, you have the following choices:

- ColdFusion Flash Remoting

  Use this option if you plan to use data-centric development tools available with Flash Builder. This option also applies if you use Flash Remoting to invoke methods in ColdFusion Components (CFCs). See Building data-centric applications with Flash Builder.

- LiveCycle Data Services

  Specify LiveCycle Data Services as a ColdFusion application type only if your ColdFusion 8 installation is configured for LiveCycle Data Services 2.6.1. See Integrating Adobe LiveCycle Data Services ES 2.6 with Adobe ColdFusion 8.

  Typically for LiveCycle Data Services, you specify J2EE as the application server type, not ColdFusion. See "Creating a Flex project with J2EE" on page 41.

- BlazeDS

  Specify BlazeDS as a ColdFusion application type only if your ColdFusion 8 installation is configured for Adobe BlazeDS 3.1. See Integrating BlazeDS with a ColdFusion 8 Installation.

  Typically for BlazeDS, you specify J2EE as the application server type, not ColdFusion. See "Creating a Flex project with J2EE" on page 41.

6 Click Next. Specify a server location, Web Root, Web Root URL, and Context Root:

If accessing a remote object service, you can configure a standalone ColdFusion configuration or a ColdFusion configuration deployed to a J2EE server:

- Standalone

  Use the Standalone option if your ColdFusion installation uses the server configuration.

  Specify the location of the ColdFusion server, location of the Web Root, and Web Root URL.

- Deployed to J2EE Server

  Use the Deployed to J2EE option if your ColdFusion installation uses either the multiserver or J2EE configurations.

  Specify a Web Root, Root URL, and Context Root. If you are using the ColdFusion multiserver configuration, you typically do not have to specify the Context Root.

  The context root typically matches the last segment of the root URL path when you deploy ColdFusion as a web application in the ColdFusion J2EE configuration.

When specifying the location of the server and web root, navigate to a local directory or specify a path to a directory on a network server. Make sure the directory is a shared directory and the account under which Flash Builder is running has write access.

Make sure you have mapped or mounted a network drive for the network server. The path to a network server is platform-specific. For example:

(Windows) `\\10.192.18.12\server\webroot`

(Windows) `Z:\webroot`

(Mac) `/Volumes/webroot`

7 Click Validate Configuration to ensure the setup is correct.

If the web root directory is not writable, then Flash Builder displays a warning.

8 Choose an output folder for the compiled application.

9 Click Finish, or click Next to select more configuration options.

"Additional project configuration options" on page 44

Creating a Flex project that accesses PHP services

To access data from PHP services, a server hosting the services must be available. The server can be either a local server or a server available from a local network.

1 Select File > New > Flex Project.

2 Enter a project name.

3 Select the project location. The default location is the current workspace. To choose a different project location, deselect the Use Default Location option.

4 Choose the application type (web or desktop).

5 For application server type, choose PHP. Click Next.

6 Specify the Web Root and Root URL for the service. Click Validate Configuration.

Typically, you specify a Web root and root URL that is local to your environment. However, you can also access network servers. Make sure the directory is a shared directory and the account under which Flash Builder is running has write access.

Make sure you have mapped or mounted a drive for the network server. Then specify a path to the server. The path is platform specific. For example:

(Windows) `\\10.192.18.12\server\webroot`

(Windows) `Z:\webroot`

(Mac) `/Volumes/webroot`

7 (Optional) Specify the output folder for your application.

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

"Additional project configuration options" on page 44

Additional project configuration options

When creating a Flex project, you can customize its configuration. All additional configuration steps are optional.

Note: *You can also change a project's configuration after the project has been created. When the Flash Builder editor is in Source mode, go to Project > Properties.*

- Main source folder, main application file, and output folder URL.

  By default, Flash Builder places source files in a projects src folder. The default name of the main MXML application file is the name of the project. You can change these defaults when creating the project.

  When you create a project, Flash Builder runs application files from a default URL based on your project settings. Specify an output folder URL to override the default settings.

  See "Setting up a project output folder" on page 74 and "Running applications" on page 85.

- Component Set

  Typically, you make all components available. In some cases, you specify MX components only. See "Component Set (MX + Spark or MX Only)" on page 73.

- Framework Linkage

  By default, application classes for the Flex 4 framework use dynamic linking. The following options are also enabled by default:

  - Verify RSL Digests
  - Use Local Debug SWF RSL When Debugging
  - Automatically Determine Library Order Based on Dependencies

  See "Application Framework Linkage" on page 74.

- Build Path Libraries

  You can add or remove project libraries, SWC library folders, or SWC files to the build path. You can also change the build path order.

  Use the Edit button to change the location of added libraries or folders.

  Use the Add Flex SDK button to restore the default SDK for a project if you removed the Flex SDK from the build path.

- Additional source folders

  Use the Source tab to add additional source folders to a project. You can reorder the source folders, edit the location of folders, and remove folders from the source path.

Changing server options of existing projects

At times you may find that the original server configuration for a project does not meet your current needs. You can reconfigure the server configuration for a web application or desktop application from the Project Properties window.

In the Project Properties window select the Flex Server option to add or change the server options for the project:

- Select None to remove the server configuration from a project.

  Removing the server configuration from a project removes any added SWCs on the library path for that server type.

- Select a server type to change or add the server configuration of a project

  All the server options for the selected server configuration are available. See "Creating Flex projects" on page 40 for details on server configuration settings.

  Changing the server type of a project can result in errors in existing code that relies on the original server type. You will need to investigate and correct any resulting errors in your code.

Creating a Flex 4 project that uses only MX components

You can create a Flex project that is compatible with MX components that were available with Flex 3. This MX Only option is useful for creating applications that are similar in design to applications created with the previous release of Flex, but still have access to Flex 4 and Flash Builder 4 features (such as the new states syntax, advanced CSS, compiler improvements, and other language features).

If you specify MX Only for a project, then Spark components available with Flex 4 are not available to applications in the project.

You can convert a Flex 4 project to be an MX Only project. However, Flash Builder does not rewrite any code in the project. You must manually update your code to remove any reference to Spark components.

Creating an MX Only Flex project

1. Select File > New > Flex Project.
2. Enter a project name.
3. Select the project location. The default location is the current workspace. To choose a different project location, deselect the Use Default Location option.
4. Choose the application type (web or desktop).
5. Specify an application server type (or specify none if an application server type is not required). Click Next.
6. (Optional) Specify the output folder for your application.
7. Click Finish, or click Next to select more configuration options.

   "Additional project configuration options" on page 44

Converting a Flex project to an MX Only Flex project

1. Make the project you want to convert the active project in Flash Builder:

   Typically, you open a source file in the project to make the project active.

2. Select Project > Properties > Flex Build Path.
3. For Component Set, select MX Only. Click OK.
4. Modify any application code in the project that accesses Spark components.

   You cannot reference Spark components in an MX Only project.

Creating Flash Professional projects

Use Flash Professional projects to access Flash FLA or XFL files created with Flash Professional CS5. This feature allows Flash Professional developers to take advantage of the editing and debugging environment available with Flash Builder. The features of Flash Professional projects are only available in Flash Builder if you have installed Flash Professional CS5.

Typically, you create a project and files in Flash Professional. Then you create a corresponding project in Flash Builder to edit and debug the files. When editing the files in Flash Builder, you can set breakpoints in the project's ActionScript files. Breakpoints set in files that are in the Flash Professional project are recognized by the Flash Professional debugger when you call Debug Movie.

You can launch Flash Professional from Flash Builder to publish and run the files. You can also launch the Flash Professional debugger from Flash Builder.

Creating a Flash Professional project

1 Select File > New Flash Professional Project.

2 Navigate to the target FLA or XFL file for the project.

The name of the file becomes the name of the project.

3 Specify a project location:

You can use either the default project location in the workspace or navigate to a new project location.

4 Click Finish.

Flash Builder opens the new project in the Package Explorer. The folder containing the target FLA file is accessible. The selected FLA file becomes the target file in the project. ActionScript files that are dependent to the target files are available for editing.

If Flash Professional is not running, Flash Professional starts.

Working with Flash Professional projects in Flash Builder

You can do the following with source files in a Flash Professional project:

- Edit the ActionScript files that are dependent to the target FLA file.
- Debug the file in the Flash Builder debugger or Flash Professional Debugger:

  To debug in Flash Builder, select Run > Debug *file* or click the Debug button from the toolbar.

  To debug the file in Flash Professional, select Run > Debug Movie or click the Debug Movie button in Flash Builder. Breakpoints set in Flash Builder are recognized in the Flash Professional debugger.
- Publish the file in Flash Professional CS5:

  Select Project > Publish Movie or click the Publish in Flash Professional button from the toolbar.
- Run the file in either Flash Builder or Flash Professional:

  To run the file in Flash Builder, select Run > Run *file* or click the Run button from the toolbar.

  To run the file in Flash Professional, select Run > Test Movie or click the Test Movie button from the toolbar.

Setting project properties for Flash Professional projects

1 Select Project > Project Properties > Flash Professional.

2 Select Add to add additional files to the project.

A project can have only one target FLA or XFL file as the default target file. Use the Set as Default button to specify the default target file for the project.

3 Click OK.

Managing projects

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

Setting Flex project properties

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

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

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

Builders Specifies the build tool to use. A standard builder is included in Flash Builder. You can use Apache Ant (an open-source build tool) to create build scripts or import existing Ant build scripts. (See "Customizing builds with Apache Ant" on page 78.)

Data Model Available only with LiveCycle Data Services 3 and higher. Specifies the location of the data model file, which contains service and data type information for LiveCycle Data Services.

Data/Services For projects that access data services, specifies whether to use the default code generator for accessing services. You can also specify whether to use a single server instance when accessing services. (See Extending service support in Flash Builderfor information on extending Flash Builder to use custom code generation. See Using a single server instance for information on using a single server instance when accessing services.)

Flex Applications Displays the names of the project files that are set as application files, which can be compiled, debugged, and run as separate applications. (See "Managing project application files" on page 50.)

Flex Build Path Specifies the build path, which specifies where external source and library files are located. You can modify the build path and also change the name of the output folder. (See "Setting up a project output folder" on page 74 and "Building projects manually" on page 76.)

Flex Compiler Specifies optional compiler preferences, such as generating an accessible SWF file, enabling compiler warnings and type checking, specifying additional compiler arguments, Flex SDK version, and sets HTML wrapper settings. (See "Advanced build options" on page 76.)

Flex Modules Specifies modules to build and optimize for the project. For more information about using modules in Flash Builder, see "Creating modules in Flash Builder" on page 90.

Flex Server Specifies the application server type for the project. When you create a project, you specify the application server type. You can change the application server type for a project here. If you change the application server type for a project, you may not be able to access data services previously configured. See "Creating Flex projects" on page 40 and Creating a Flex project to access data services.

Flex Theme Specifies the theme to use for all applications in the project. You can specify one of the themes available with Flash Builder or import a theme. For more information, see "Applying themes" on page 196.

Project References Lists the projects that the current project references.

Run/Debug Settings Manages launch configuration settings. See "Managing launch configurations" on page 88.

Changing Flex projects to Adobe AIR projects

You can change the application type of a Flex project from Web (runs in Flash Player) to Desktop (runs in Adobe AIR). The following changes are made during the conversion:

- An AIR descriptor file is created for each application in the project.
- The launch configurations for the project are updated to properly launch in the AIR runtime.
- Settings for HTML wrapper are removed.
- Custom Flash Player settings are removed.
- The library path is modified to include airglobal.swc instead of playerglobal.swc.

During conversion, you can specify whether to change the base Application tags to WindowedApplication tags for each application in the project. If you choose to convert these tags, this is the only change to application code that occurs during conversion. You should inspect the attributes to the base tags after the conversion to make sure the application runs as intended in Adobe AIR.

To change a web application project to a desktop application

Note: *This procedure cannot be undone.*

1. Select the project that you want to convert.

   The project should be a Flex project with the Web application type (runs in Flash Player)

2. From the context menu for the project select:

   Add/Change Project Type > Convert to Desktop/Adobe AIR project.

3. In the Convert to Desktop/Adobe AIR Project dialog, specify whether to rewrite code:

   - Convert Application Tags to WindowedApplication Tags

     For existing applications in the project, all Application tags are rewritten to WindowedApplication tags. No other change to your code occurs. Inspect attributes to the base tags to make sure the application runs as intended in Adobe AIR.

     New applications you create in the project are desktop applications and can run in Adobe AIR.

   - Do Not Rewrite Any Code

     No changes are made to your code. You must edit any applications in the project before they can run in Adobe AIR.

     New applications you create in the project are desktop applications and can run in Adobe AIR.

Moving a project from one workspace to another

You use a combination of deleting and importing operations to move a project from one workspace to another. When you delete a project from a workspace, you can remove it from the workspace but leave it in the file system (see "Deleting projects" on page 49). After you remove a project from one workspace you can import it into another.

Specifying an SDK for a project

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

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

Importing projects that use remote server compilation

Importing projects that use server compile is not supported. You can import a project that specifies server compilation, however the project is imported with errors in the Problems View. The error provides a link with information on how to convert a server compile project to a tool compile project.

Deleting projects

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

Instead of deleting the project from the workspace, you can close the project. Closing the project lets you keep a reference to it in your workspace and also free some system resources. For more information, see "Closing and opening projects" on page 49.

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

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

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

Closing and opening projects

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

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

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 50.

1. In the Package Explorer, select the MXML application file that you want to make the main application file.
2. From the Package Explorer context menu, select Set as Default Application.

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

Managing project application files

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

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

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

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

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

Exporting and importing projects

Flash Builder exports Flex projects and Flex library projects in the FXP format. ActionScript projects can only be exported as an archive file, typically in ZIP format.

The FXP format is an archive format that includes project folders, files, and metadata about the project. An exported project includes all dependent Flex library projects.

Note: *You can also use the Eclipse Export wizard to export Flex projects and Flex library projects in ZIP format or other archive formats.*

When exporting a Flex project to an FXP file, some contents of the project require special handling.

- service files

  Flex projects that connect to data services contains a `services` folder that contains links to deployed service files. When exporting the project, Flash Builder exports the links but not the deployed services. Upon import, manually deploy the service files and update the links as necessary.

  For projects that connect to services using LiveCycle Data Service or BlazeDS, make sure that service destinations are available on the target server.

  For projects that reference local files, upon import deploy the local files using the same path in the original project. This applies to projects accessing static XML service files or local files for HTTP services or web services.

- Zend Framework

  Flex projects that connect to data services using PHP and the Zend Framework contain two configuration files. Upon import, examine these files to make sure that they are configured properly for your system:

  `amf-config.ini`

`gateway.php`

See Installing Zend Framework for information on installing, configuring, and troubleshooting your Zend Framework installation.

- Data model files (LiveCycle Data Services)

  A Flex project that uses LiveCycle Data Services (LCDS) links to a data model file.

  Upon export and subsequent import, Flash Builder references the actual data model file and not a link to it. If you want to use a linked file, and not the one packaged with the exported project, then change the data model file using project properties. Select Project > Properties > Data Model and make the changes.

Export a Flex project or Flex library project as an FXP file

The menus for this procedure vary slightly between the standalone and plug-in configurations of Flash Builder.

Some Flex projects require special handling upon import. See "Exporting and importing projects" on page 50.

1. In Flash Builder, select File > Export Flex Project (FXP).

   If you have the plug-in version of Flash Builder, select File > Export > Flash Builder > Flash Builder Project.

   You can also use the context menu for a project in the Package Explorer. Select Export > Flash Builder > Flash Builder Project.

2. In the Export Flex Project wizard, select the project to export.

   All available projects for export are listed in the Project pop-up menu.

3. (Optional) Enable Validate Project compilation.

   Use this option to confirm that your project compiles without errors. If there are errors, you can still export the project.

4. Click Finish.

   For server projects, absolute paths to server resources are saved as path variables. When you later import the project, you specify values for the path variables.

Export an ActionScript project in ZIP format (or other archive format)

1. In Flash Builder, select File > Export > Other.
2. In the Export wizard, select General > Archive File. Click Next.
3. In the Export wizard select the project and files to export:
   - In the leftmost panel, expand the project to specify project folders to include
   - In the rightmost panel, for each selected folder, specify the files to include.
4. Browse to a location to save the exported project and specify a filename.
5. Specify archive file options and click Finish.

Importing projects

Flash Builder can import Flex projects, Flex library projects, and ActionScript projects. Projects can be imported from existing project folders or from files that were previously exported from Flash Builder. You can import multiple versions of the same Flex project or Flex library project. After importing multiple versions, you can compare the versions and copy or merge differences.

Some Flex projects require special handling upon import. See "Exporting and importing projects" on page 50.

Support for Catalyst projects

Flash Builder provides development support to application designers using Adobe® Flash® Catalyst™. Catalyst exports a project as an FXP file. Catalyst exports components in as an FXPL file. An FXPL file is a library package. The FXP and FXPL files can then be imported into Flash Builder for development. For FXP files, the resulting project is a Flex web project that runs in Adobe Flash Player. An FXPL file contains a library file. You can import an FXPL files as a Flex library project or you can import the contents into an existing Flex project.

You can create an Adobe AIR project from a Catalyst project. Import the FXP file for the Catalyst project into Flash Builder. Convert the application type for the project from Web (runs in Adobe Flash Player) to Desktop (runs in Adobe AIR). See "Changing Flex projects to Adobe AIR projects" on page 48.

Importing a Flex project or Flex library project

You can import a project from an exported FXP file or by navigating to a folder containing the project.

Note: *See "Importing a Catalyst FXPL project" on page 53 for information on importing the contents of a library project into another Flex project. An FXPL project is a library project created by Adobe Catalyst.*

The menus available for this procedure vary slightly for the plug-in configuration of Flash Builder.

1. From the Flash Builder menu, select File > Import FXP.

   If you have the plug-in version of Flash Builder, select File > Import > Flash Builder > Flash Builder Project.

   You can also use the context menu for the Package Explorer to import a project.

2. (Project folder) If you are importing from an existing project folder, select Project Folder, and navigate to the folder containing the project.

3. (FXP file) If you are importing from an FXP file, select File and navigate to the location of the file.

   If an FXP file contains more than one project, you can select individual projects to import.

4. (Library project or FXPL project) If you are importing a library project or a Catalyst FXPL project, you have the option to import the contents into an existing project. See "Importing a Catalyst FXPL project" on page 53

5. (FXP file) If a project by the same name exists in the workspace, specify the import method:

   - Import as New Project: Flash Builder appends a numeric identifier to the project name. Previous versions of the project are preserved.

     In the Extract To field, specify a location in which to extract the file. Typically, this location is a directory in your Flash Builder workspace representing a project folder. You can specify a new project folder or overwrite an existing project folder.

   - Overwrite existing project: Select the project to overwrite. The previous version of the project is permanently removed.

6. (Path variables) If you are importing a project that defines path variables, update path variables for the project.

   Projects compiled for ColdFusion, PHP, LiveCycle Data Services, or other server technologies use path variables to access a web server and server resources. Other projects can have user-defined path variables.

   Select each path variable and provide a valid value for the variable.

7. (Font references) If you are importing an FXP exported by Catalyst, the project can contain font references. You have the option to resolve references to fonts.

   See "Resolving font references when importing Catalyst projects" on page 54.

8 Click Finish.

9 (PHP server projects) If you are importing a project of application server type PHP, then install or update your Zend installation.

The Zend dialog guides you through the process.

Note: *If you cancel the process in the Zend dialog, manually install or update your Zend framework. You cannot access PHP services unless the Zend Framework is properly installed and configured. See Installing Zend Framework for information on installing, configuring, and troubleshooting your Zend Framework installation.*

10 (Server projects) Deploy services.

a Manually place services in the web root of the server. Use the same directory structure that was used in the original project.

b In the Data/Services view, from the context menu for a service, select Refresh.

Importing a Catalyst FXPL project

A Catalyst FXPL project is a library project created by Adobe Catalyst. When you import an FXPL project, you have the option to import the contents into another Flex project or Flex library project.

This feature is designed to help developers working with Catalyst application designers. However, you can use this feature to import the contents of any library project into another Flex project or Flex library project.

The menus available for this procedure vary slightly for the plug-in configuration of Flash Builder. This procedure assumes that you are importing a library project.

1 From the Flash Builder menu, select File > Import FXP.

If you have the plug-in version of Flash Builder, select File > Import > Flash Builder > Flash Builder Project.

You can also use the context menu for the Package Explorer to import a project.

2 Select File and navigate to the location of the file.

3 Specify the import method:

- Import a new copy of project: Flash Builder appends a numeric identifier to the project name. Previous versions of the project are preserved.

  In the Extract To field, specify a location in which to extract the file. Typically, this location is a directory in your Flash Builder workspace representing a project folder. You can specify a new project folder or overwrite an existing project folder.

- Import contents into existing project.

  For Source Folder, browse to a `src` folder of an existing project. For Package, browse to an existing package or specify a new package name for the contents.

- Overwrite existing project: If a project by the same name exists in the workspace, you can overwrite the existing project.

  Select the project to overwrite. The previous version of the project is permanently removed.

4 Click Finish.

When importing FXPL files, Flash Builder attempts to resolve references to fonts in the FXPL file. See "Resolving font references when importing Catalyst projects" on page 54.

Resolving font references when importing Catalyst projects

When importing an FXP project created with Adobe Catalyst, the imported project can contain references to fonts that are not available on your system.

The Import wizard provides the option to fix font references using CSS. If you select this option, Flash Builder imports the Catalyst style sheet `Main.css`. `Main.css` contains references to the fonts used in the project.

If you get compile errors from the fonts referenced in the style sheet, fix the references in the style sheet with fonts available on your system.

Catalyst FXPL projects do not contain style sheets. Flash Builder attempts to correct any references to fonts when importing an FXPL file. If Flash Builder cannot find a corresponding font on the target system, the original font references are left intact. For FXPL projects, font references that Flash Builder cannot resolve are discovered at runtime. There is either a font substitution or a runtime error for unresolved font references.

Note: *For FXPL files, Flash Builder modifies the* `fontFamily` *attribute in MXML files when it attempts to resolve font references.*

Importing a Flex 3 project

You can import a Flex 3 project into Flash Builder using Flex 3 Compatibility Mode. In this case, the namespaces and components are unchanged from Flex 3. However, you can take advantage of the compiler available with Flex 4.

New documents created in Flex 3 Compatibility Mode use MX components and the following namespace:

```
mx="http://www.adobe.com/2006/mxml"
```

1. In Flash Builder, select File > Import Flex Project.
2. Navigate to the previously exported Flex 3 project ZIP file, or browse to the Flex 3 Project folder.
3. Click Finish.
4. In the Choose Flex SDK Version dialog, make sure that Flex 4 SDK is specified. Select Use Flex 3 Compatibility Mode.
5. Select OK.

Comparing changes in a project

If you import multiple versions of a project you can compare, copy, or merge the contents of the versions. You can only compare different versions of the same project.

1. In the Package Explorer, select one of the projects you want to compare.
2. Open the Package Explorer context menu and select Compare Project With Version.

   The Compare Viewer launches, allowing you to compare the project with other versions of the project.
3. Select the version for comparison, which opens the Eclipse Compare Editor.
4. In the compare editor, navigate to the file you want to compare and from the context menu, select Show Content Comparison.

   The Compare Editor displays both versions of the file, with differences highlighted.

   You can use Compare Editor options to copy or merge differences in the file. See the Eclipse documentation on the Compare Editor for details.

Importing an ActionScript project

ActionScript projects are exported in ZIP archive format. Use the Eclipse import wizard to import ActionScript projects.

1 In Flash Builder, select File > Import > Other > General > Archive File.

You can also use the context menu for the Package Explorer to import an ActionScript project.

2 In the Import Flex Project dialog box, select the ZIP file you want to import.

3 Click Finish.

Importing projects exported with the Eclipse Export wizard

If you have a project that was exported using Eclipse's Export wizard, use the Eclipse Import wizard to import the project. Select File > Import > General. Then navigate to the appropriate format for your project.

For more information, see the Eclipse documentation for importing projects. This documentation is available as Help in the Eclipse Import and Export wizards.

If the project contained services created with Flash Builder tools for accessing data services, then you manually have to add the services. Copy the server files in the services folder to an appropriate server. Use the service properties for a server from the Data/Service view to determine the service location.

If you exported a PHP project that uses the Zend Framework, the Zend Framework must be installed on the target server. Modify the `amf-config.ini` file that configures the Zend Framework. For `zend_path`, specify the absolute path to the Zend installation directory. See Installing Zend Framework for information on installing, configuring, and troubleshooting your Zend Framework installation.

Importing projects into multiple workspaces

When you import a project, you import it into a Flash Builder workspace. A project can be imported into multiple workspaces. In this scenario, the project files exist on disk in one location, but are referenced by each workspace. Changes you make to a project apply to all workspaces.

Importing source files into a new project

If you have source files and assets on your file system, but are not in a project, you can create a new project for these files.

1 From the Flash Builder menu, select File > New > *Project*.

Project can be a Flex project, Flex library project, or ActionScript project.

2 In the New Project wizard, specify the source and output folder settings to the appropriate location in your file system.

Note: *You could also accept the default wizard locations and move the source files accordingly.*

Exporting Adobe AIR application installer

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

- Sign the application using a VeriSign or Thawte digital certificate
- Create and use a self-signed digital certificate

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

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

1 Select Project > Export Release Build.

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

2 Choose the export settings for project and application.

   - If your project does not have a server web root associated with it, all assets are copied to the *project_name* folder, which is the default location.
   - If your project has server web root associated with it (for example, PHP and J2EE), all assets are copied to the *web_root/project_name*-debug folder.
   - If you want users to view source code, select Enable View Source.
   - Click Choose Source Files to select files to you want to publish, then click OK.

     Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

   - Click Next.

3 On the Digital Signature page:

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

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

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

5 Click Finish.

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

Exporting a release version of an application

You can export an optimized release-quality version (non-debug SWF file or AIR file) of your application using the Export Release Build wizard. Required assets are copied to a folder separate from the debug version. After running the wizard, additional steps are required to deploy your application on a server.

Export a release build (Web application, runs in Adobe Flash player)

1 Select Project > Export Release Build to open the Export Release Build wizard.

2 Select the project and application you want to export.

3 (Optional) Select Enable View Source to make application source files available from the exported application.

Click View Source Files to specify which source files to include. In addition to the specified source files, the wizard generates a ZIP archive file containing the source files.

Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

4 Click Finish.

5 Copy the folder containing the exported release to the web root of the server hosting the application.

6 (Server projects) If exporting a release from a project that specified an application server type, deploy the services to the web root of the target server.

Maintain the same directory structure used during development.

This step applies to ColdFusion, PHP, BlazeDS, and LCDS services. You specify the application server type when creating a project in Flash Builder.

For local files deployed on the server, a cross-domain policy file is necessary to access these services. This applies to projects accessing static XML service files or local files for HTTP services or web services. See Using cross-domain policy files.

7 (PHP server projects only) For PHP projects, perform these additional steps:

a Install the Zend framework on the server. See Installing Zend Framework.

b Modify the `amf-config.ini` file, which is in the output folder of the exported release:

For `zend_path`, specify the absolute path to the Zend installation directory.

Set `amf.production` to **true**.

Update `webroot` with the absolute path to the web root on the server.

Export a release build (Desktop application, runs in Adobe AIR)

1 (Optional) Change the server settings in the project properties.

An exported desktop application can only access services used during development. If you want to change the server for the exported desktop application, modify the project settings.

2 Select Project > Export Release Build to open the Export Release Build wizard.

3 Select the project and application you want to export.

4 (Optional) Select Enable View Source to make application source files available from the exported application.

Click View Source Files to specify which source files to include. In addition to the specified source files, the wizard generates a ZIP archive file containing the source files.

Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

5 Click Finish.

6 Copy the `.air` project to the target desktop.

7 (Server projects) If exporting a release from a project that specified an application server type, deploy the services on the target server.

This step applies to ColdFusion, PHP, BlazeDS, and LCDS services. You specify the application server type when creating a project in Flash Builder.

8 (PHP server projects only) For PHP projects, perform these additional steps:

a Install the Zend framework on the server. See Installing Zend Framework.

b Modify the `amf-config.ini` file, which is in the output folder of the exported release:

For `zend_path`, specify the absolute path to the Zend installation directory.

Set `amf.production` to **true**.

Update `webroot` with the absolute path to the web root on the server.

Export Adobe AIR application installer

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

- Sign the application using a VeriSign or Thawte digital certificate
- Create and use a self-signed digital certificate
- Choose to package the application and sign it later

Digital certificates provided by VeriSign and Thawte give users some assurance as to your identity as a publisher and verification that the installation file has not been altered since you signed it. Self-signed digital certificates serve the same purpose but they are not validated by a third-party.

You also have the option of packaging your AIR application without a digital signature by creating an intermediate AIR file (.airi). An intermediate AIR file is not valid in that it cannot be installed. It is instead used for testing (by the developer) and can be launched using the AIR ADT command line tool. This capability is provided because in some development environments signing is handled by a particular developer or team, which ensures an additional level of security.

1 Select Project > Export Release Build.

2 In the Export Release Build wizard, choose the export settings for project, application, and folder.

- If your project does not have a server web root associated with it, all assets are copied to the *project_name* folder, which is the default location.
- If your project has server web root associated with it (for example, PHP and J2EE), all assets are copied to the *web_root/project_name*-debug folder.
- If you want users to view source code, select Enable View Source.
- Click Choose Source Files to select the files you want to publish, then click OK.

  Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

- Click Finish.

For more information about Adobe AIR files, see the *Adobe AIR Developer's Guide*.

Digitally signing your AIR applications

1 Select Project > Export Release Build.

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

2 (Optional) Select Enable View Source if you want users to see the source code when they run the application.

Click Choose Source Files to select individual files.

Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

3 On the Digital Signature page:

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

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

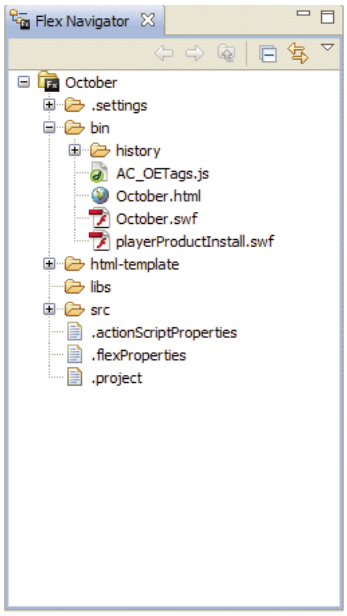
4 Click Finish.

Debug version

The debug version of your application contains debugging information and is used when you debug your application. The Export Release Build version does not include the additional debugging information and is therefore smaller in size than the debug version. An HTML wrapper file contains a link to the application SWF file and is used to run or debug your application in a web browser.

Note: *Both the Run and Debug commands will launch the development build in the bin-debug folder (not the exported release build folder, bin-release.)*

In a standard application, a typical output folder resembles the following example:



You can run or debug your Flex and ActionScript applications 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 87). For more information about running and debugging your applications, see "Running applications" on page 85 and "Debugging your applications" on page 132.

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

Managing project resources

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

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

Creating folders and files in a project

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

Create a folder

1 In Package Explorer select File > New > Folder.

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

If you create the new folder in the source path folder, it is treated like a package name and you can place source files inside that will be recognized by the compiler.

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

3 Enter the folder name and click Finish.

Create a file

1 In the Package Explorer, select File > New > File.

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

3 Enter the filename and click Finish.

You can also add folders and files that are located outside the current project; for more information, see "Linking to resources outside the project workspace" on page 61.

Deleting folders and files

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

Note: *If you delete a linked resource, you delete only the link from your project, not the resource itself (see "Linking to resources outside the project workspace" on page 61). However, if you've linked to a folder and you delete any of the files in it, they are removed from the file system.*

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

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

The resource is deleted from the file system.

Moving resources between projects in a workspace

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

1 In the Package Explorer, select the resource to move.

2 Do one of the following:

- Drag the resource to a new project.
- Cut and paste the resource to another project.

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 61.*

Refreshing resources in the workspace

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

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

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

Manually refresh the workspace

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

Turn off the automatic refresh preference

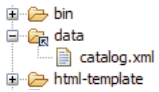
1 Open the Preferences dialog and select General > Workspace.

2 Deselect Refresh Automatically.

Linking to resources outside the project workspace

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

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



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

When you work with shared resources, the changes you make to the source folders and files affect all of the projects that are linked to them. Be cautious when you delete linked resources from your projects; in some cases you merely delete the link reference, and in others you delete the source itself. For more information, see "Deleting folders and files" on page 60.

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

Link to resources outside the project workspace

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

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 76.

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

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

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

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

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

   Click OK to create the path variable.

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

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

Adding resource folders to the project source path

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

Add an external resource folder to the source path

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

   The folder is added to the source path.

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

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

Alternatives to using project references

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

Flex Library projects The preferred way to create a reusable library. Flash Builder creates a project reference to ensure that the SWC project is built before the main project that includes it on the library path. Also, because Flash Builder adds it to the library path, code hints appear in the main project for the classes in the SWC project.

Source path The recommended way to include code in your project that is not under the same folder structure. This enables code hints in the project files and classes in related files, and the compiler knows where to find the source code. You can add any number of source paths to your project and they are displayed as linked folders in the Package Explorer.

Viewing resource properties

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

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

ActionScript projects

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

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

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

Creating ActionScript projects

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

1 Select File > New > ActionScript Project.

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

Project Location The default location is the current workspace. On Windows platforms, the default workspace location is C:\Documents and Settings*Flex Developer*\Adobe Flash Builder\. On the Macintosh, the default workspace location is /Users/*Flex Developer*/Adobe Flash Builder/. You can choose a different project location by deselecting the Use Default Location option.

Flex SDK Version Choose default or specific. You can also click the Configure SDKs link to add, edit, or remove SDKs on the main Preferences page.

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

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

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

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

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

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

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

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

Creating an ActionScript class

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

1 Select File > New > ActionScript Class.

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

After clicking Finish, Flash Builder saves the file in the specified package and opens it in the code editor.

If you saved the file in the current project or in the source path of the current project, Flash Builder also displays the component in the Components view so that you can rapidly insert it in your applications. For more information, see "Add components in MXML Design mode" on page 180.

3 Write the definition of your ActionScript class.

For more information, see Simple Visual Components in ActionScript.

Creating an ActionScript interface

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

1 Select File > New > ActionScript Interface.

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

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

Generating Accessor Functions

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

Flash Builder can generate ActionScript get and set accessor functions for class variables. When you generate getters and setters, Flash Builder provides the following options:

- Make the class variable private.

  Typically, class variables have private access.

- Rename the class variable, suggesting a leading underscore for the variable name.

  By convention, private class variables have a leading underscore.

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

Generate get or set accessor functions

1 With an ActionScript file open in the Source Editor, place the cursor on a class variable.

2 Select Source > Generate Getter/Setter from either the Flash Builder menu or the context menu.

3 In the Generate Getter/Setter dialog, specify details for the accessor functions and click OK.

Note: *If you want to view the code that will be generated, select Preview before clicking OK.*

Library projects

Library projects let you build custom code libraries that you can share between your applications or distribute to other developers. A library project generates a SWC file, which is an archive file for Flex components and other resources. For example, the Flex framework is contained in SWC files. When you create a Flex project, the Flex framework SWC files are added to the project's library path. You can view and edit the library path by accessing the project's build path properties page (for Flex projects, select Project > Properties > Flex Build Path).

Archived into a SWC file is a SWF file that contains components and resources and a catalog.xml file that is the manifest of the elements contained within the SWF file. Typically, the SWF file contains one or more components and any other required resources. Adding the library to a project lets you use those components in your application and also enables code hinting for those components.

In addition to providing a convenient way to package and distribute components, SWC libraries are used as themes, the visual appearance of Flex applications. A SWC theme file contains a CSS file and all the related graphic assets. For more information about creating and using themes, see About themes.

Libraries are useful if you create components entirely in ActionScript and use them in Design mode in Flash Builder. ActionScript components are not visually rendered in Design mode until they are compiled into a SWF file. By adding ActionScript components to a library project, you create a SWF file that is contained in a SWC file. You can add the library to a project's library path, and the ActionScript components visually render in Design mode when you add them to the application.

Configuring libraries for your applications

You use SWC libraries in your projects in the following ways:

Merged into the application When you add a SWC file to the project's library path, the components contained in the library are available to use in your application. When you build the application, only the library components you actually used are compiled into the application SWF file. In other words, all of your application code is merged into a single SWF file. This is the most common and straightforward way of using library components.

External to the application You can keep library components separate from the compiled SWF file, so they are not merged into the file. The compiler resolves all code contained in the library that is used by the application, but does not merge the code into the application SWF file. The advantage of this approach is that you make the application SWF file smaller. The components contained in the SWC file are retrieved and loaded into memory as needed, at run time.

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

There are several advantages to sharing components between applications by using an RSL. First, the library is loaded into memory once, cached, and then available to all the applications that use those components. Second, the components contained within the library are only loaded when they are needed, which reduces the application's start-up time because the size of each application is smaller. The potential problem to this approach is that the entire RSL is loaded into memory, rather than the individual components that the applications use. For more information about when to use SWC files as an RSL, see Using Runtime Shared Libraries.

Creating Flex library projects

When you create a library project, the New Flex Library Project wizard guides you through the steps, prompting you for the project name, location, and build path information. After you create the Library project, you add components, specify the library project elements to include in the SWC file, and then build the project to generate the SWC file. The first step in creating a SWC file in Flash Builder is to create a Flex Library project.

1 Select File > New > Flex Library Project.

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

Project Location The default location is the current workspace. On Windows platforms, the default workspace location is C:\Documents and Settings*Flex Developer*\Adobe Flash Builder\. On the Macintosh, the default workspace location is /Users/*Flex Developer*/Adobe Flash Builder/. You can choose a different project location by deselecting the Use Default Location option.

Flex SDK Version Choose default or specific. You can also click the Configure SDKs link to add, edit, or remove SDKs on the main Preferences page.

Include Adobe AIR libraries Select this option if your library must use AIR features, such as access to the AIR APIs. Flash Builder then changes the library path of this new Flex Library project so that it includes airglobal.swc and airframework.swc. Web-based Flex projects cannot use this library.

Do not select this option if you are writing a generic library intended to be used only in a web-based Flex application, or in either a web-based or AIR-based application.

3 Click Next.

4 (Optional) Set the build path information. For example, you can add folders to the project's source path that contains the components to include in the SWC file. You can also add other projects, folder, or library SWC files to include in your library project. See "Using SWC files in your projects" on page 68.

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 61.)
- Add a linked folder that contains components to the library project's source path. (See "Adding resource folders to the project source path" on page 63.)

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

Selecting library project elements to include in the SWC file

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

1 Select Project > Properties > Flex Library Build Path.

The components that you added to the project (either directly or by linking to them) appear in the Classes tab.

2 Select the component classes to include in the SWC file.

3 (Optional) Select the Resources tab and then select the resources to include in the SWC file.

4 After you make your selections, click OK.

Building library projects

After you select elements to include in the SWC file, and if you selected the Build Automatically option, the SWC file is immediately compiled and generated into the project's output folder. If you build your projects manually, you can build the library project when you want by selecting Project > Build Project or Build All.

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

A SWC file is an archive file. You can open the SWC file in any archive utility, such as WinZip. Inside the SWC file are the library.swf and catalog.xml files. There also are properties files and other embedded assets.

You can export the library as an open directory rather than as a SWC file. You typically export a library as an open directory when you plan on using the library.swf file inside the SWC file as an RSL.

You do this by setting the `directory` and `output` compiler options. You set the `output` option to the name of a directory to create, and set the `directory` option to `true` to indicate that you want an open directory and not a SWC file when you build the library. To edit the compiler options, select Project > Properties > Flex Library Compiler, and add the options to the "Additional compiler arguments" field; for example:

```
-directory=true -output=myOpenDir
```

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

Using SWC files in your projects

To use SWC files in your Flex projects, you add them to the project's library path. The SWC files can be located in the project, in a Flex library project, in a shared folder within the workspace, or any other location that has been linked to the project (using a shared folder that was added to the project's source path, for example).

When you use SWC files in applications, there are configuration options that determine whether they are statically or dynamically linked to the application, merged into the application SWF file, or external to it and accessed separately at run time.

Add an SWC file to the library path

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

2 Click on the Library Path tab.

3 Select any of these options to add SWC files:

Add Project Adds a Flex library project.

Add SWC Folder Lets you add a folder that contain SWC files.

Add SWC Adds a compiled SWC file.

Add Flex SDK Lets you add other Flex SDKs. If your project already has a Flex SDK in its library path, this button is disabled. If you remove the existing Flex SDK from your library path, the button is enabled. When you click this button, a Flex SDK node is added, but you are not prompted which one is added. To control which Flex SDK to use, select Project > Properties > Flex Compiler.

4 Enter or browse to and select the location of the SWC file, project, or folder. Click OK.

The SWC file, library project, or folder is added to the library path.

Merge the SWC file into the application SWF file when compiled

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

2 Click on the Library Path tab, and then select and expand the SWC file entry to display the SWC options.

3 Double-click the Link Type option. The Library Path Items Options dialog box appears.

4 Select the Merged into Code option, and click OK.

This procedure is the equivalent of using the `library-path` compiler option.

Set the SWC file as an external library file

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

2 Select the Library Path tab, and then select and expand the SWC file entry to display the SWC options.

3 Double-click the Link Type option. The Library Path Items Options dialog box appears.

4 Select the External option, and click OK.

This procedure is the equivalent of using the `external-library-path` compiler option.

Use the SWC file as an RSL

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

2 Select the Library Path tab, and then select and expand the SWC file entry to display the SWC options.

3 Double-click the Link Type option. The Library Path Items Options dialog box appears.

4 Select the Run-time Shared Library (RSL) option.

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

6 (Optional) To extract the SWF file in the SWC file when it is placed in the deploy location, select the Automatically Extract SWF to Deployment Path option.

7 Click OK.

Using the SWC files as an RSL simplifies the process for using RSLs manually. To do this, you extract the SWF file from the SWC file and set the values of the `runtime-shared-library-path` compiler option.

For more information about using SWC files as an RSL, see Using Runtime Shared Libraries in *Building and Deploying Adobe Flex3 Applications.*

Building projects

Adobe® Flash® Builder™ automatically builds and exports your projects into applications, creating application and library files, placing the output files in the proper location, and alerting you to any errors encountered during compilation.

There are several options for modifying the build settings to control how your projects are built 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.

Understanding how projects are built and exported

A typical workflow consists of building your Flex and ActionScript projects with the Build Automatically option enabled. During the development process, Flash Builder gives you errors and warnings in the Problems view. When you run your application, a debug version of the SWF file is placed in the project output (bin) folder along with required assets and an HTML wrapper. This build contains debug information and is suitable for developer use only. For more information about exporting projects, see "Exporting and importing projects" on page 50.

When your application is ready to deploy, you create an optimized, release-quality version of your application using the Export Release Build wizard. This stores the SWF file in the bin-release folder. Since debug information is removed, the file size is smaller. This version is a production build that can be viewed by end users. For Adobe AIR projects, AIR applications are exported to an AIR file. You use Export Release Build to create a digitally signed AIR file, which users must install before running an application (similar to an install.exe).

Adobe LiveCycle Data Services ES projects may instead be compiled on the server when accessed. For more information, see "Managing projects" on page 47.

For library projects, you do not have to export. The SWC file built by a Flex library project is suitable for both developer and production use. For more information see "Library projects" on page 66.

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*.

Flash Builder automatically builds your projects whenever a file in your project is changed and saved. While you have the option of building your applications manually, this should not be necessary; however, understanding the build process and the output files that are generated will help you to diagnose and repair project configuration problems that may arise.

Flex projects Source files and embedded assets (such as images) are compiled into a single output format called SWF, which is a file that can be run directly in the stand-alone Flash Player or in a web browser through an *HTML wrapper* file that is also generated by the build. These files are generated into the project's output folder (by default, this is named bin but you can name it anything you like).

LiveCycle Data Services ES projects When using LiveCycle Data Services ES you have the option of creating projects that are compiled on the server. When the MXML application file is first accessed (through a web browser), it is compiled into a SWF file.

Note: *Even though you can configure LiveCycle Data Services ES projects to be compiled on the server, Flash Builder compiles these projects as you develop your applications so that the compiler can validate code syntax and display error messages. These projects have no output folder option and Flash Builder does not generate output files.*

ActionScript 3.0 projects Like Flex projects, ActionScript 3.0 projects compile source files and embedded assets into a SWF file.

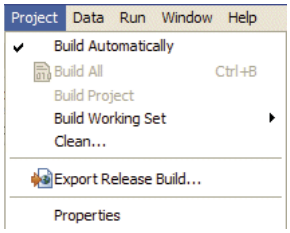
Flex library projects For library projects, source files are components and related resources. When library projects are built, a SWC file is generated into the output folder. A SWF file is archived into a SWC file containing components, resources, and a catalog.xml file that is the manifest of the elements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

Automatic builds

In the stand-alone configuration of Flash Builder, your applications are built automatically. In the plug-in configuration, you must select the Build Automatically option. While you have the option to build your applications manually, as mentioned above, this should not be necessary. Turning off automatic builds also prevents the compiler from identifying syntax errors and displaying warning and error messages as you enter code. In other words, you will not get any feedback in the Problems view until the project is compiled; therefore, it is best to set Flash Builder to build automatically.

Advanced project build options

With the advanced build options, you can control the timing and scope of your builds. For example, you can build a single project, all projects in the workspace, or create a working set (a collection) of projects to build. All build commands are accessible from the Project menu, as shown in the following example. For more information, see "Advanced build options" on page 76.



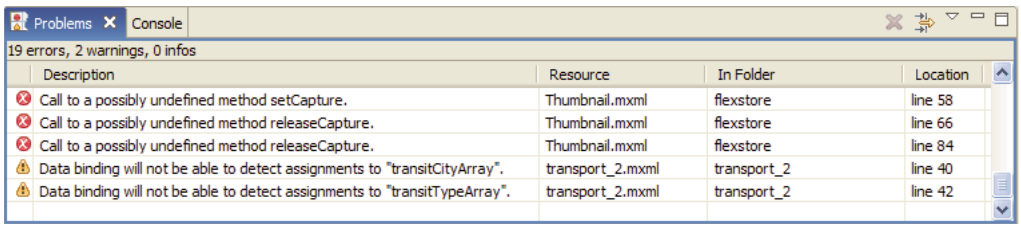
The Flash Builder compiler is incremental. It builds only those resources that have been added or affected by updates and ignores all others. This saves time and system resources. You have the option, however, to rebuild all the resources in the project. You do this by performing a *clean build*. You might do this if your application is behaving erratically during testing and you want to eliminate all potential sources of the problem by discarding and rebuilding all the files in your project. For more information, see "Advanced build options" on page 76.

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 76.

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 76, "Managing project application files" on page 50 and "Advanced build options" on page 76.

Build errors displayed in the Problems view

Errors encountered by the compiler during builds appear in the Problems view, which is included in the Development and Debugging perspectives, and in the code editor, where lines of code containing errors are marked with an x, as in the following example:



For more information about working with the Problems view, see "Using the Problems view" on page 118.

Eclipse environment errors in the log file

Sometimes you encounter errors thrown by the Eclipse environment. These errors most often occur when resources such as SWC files are not found at run time. In these cases, you can see the error messages in the Eclipse Error Log file. The default location of this log file on Windows is c:\Documents and Settings*user_name*\workspace\.metadata\.log. For Macintosh, the default location is also in the workspace directory, but files and directories that begin with a dot are hidden by default.

Custom build scripts with Apache Ant

You can modify and extend the standard build process by using Apache Ant, which is an open-source Java-based build tool. For more information about creating custom builders, see "Customizing builds with Apache Ant" on page 78.

Flex compiler options

You can modify the default Flex compiler settings that Flash builder uses. To view the default settings and modify the defaults, open the Flex Compiler properties page. From the Flash Builder menu, select Project > Properties > Flex Compiler.

Flex SDK version

The default SDK for Flash Builder 4 is Flex 4.0. However, if your project uses a specific version, such as Flex 3.4, Flash Builder compiles applications in the project using the specified Flex SDK.

You can change the default setting to use a specific Flex SDK, or to compile using Flex 3 compatibility. Specifying backward compatibility affects some behavior such as the layout rules, padding and gaps, skins, and other style settings. In addition, it affects the rules for parsing properties files. Setting the compatibility version does not enforce all differences that exist between the versions.

For more information, see Backward compatibility.

Adobe Flash Player options

The default version of Flash Player used by the compiler is the minimum version required by the Flex SDK used for compilation.

You can specify a specific version of Flash Player that you want to target for the application. Features requiring a later version of Flash Player are not compiled into the application.

Compiler options

Flash Builder provides check boxes for the following compiler options:

- Use Flash Text Engine in MX components

  The Flash Text Engine (FTE) is a library that provides text controls with a rich set of formatting options. All Spark components in the spark.components package support FTE. See Using embedded fonts.

  Some MX controls provide support for FTE. MX controls that support FTE use the same embedded fonts as Spark components using FTE. See Using FTE in MX controls.

- Copy non-embedded files to output folder
- Generate accessible SWF file

  Enables accessibility features when compiling the application or SWC file. For information on using accessibility features with Flex, see Accessible applications.

- Enable strict type checking

When strict type checking is enabled, the compiler prints undefined property and function calls. The compiler also performs compile-time type checking on assignments and options supplied to method calls.

- Enable warnings

  This option enables specified warnings. For more information, see Viewing warnings and errors.

You can also specify compiler arguments that are available with the command-line, mxmlc compiler. You can set the values of most options in the Additional Compiler Arguments field by using the same syntax as on the command line. For information about the syntax for setting options in the Flex Compiler dialog box, see About the command-line compilers.

In the Additional Compiler Arguments field, you can substitute a path to the SDK directory by using the `${flexlib}` token, as the following example shows:

```
-include-libraries "${flexlib}/libs/automation.swc" "${flexlib}/libs/automation_agent.swc"
```

HTML wrappper

In addition to generating SWF files, the Flash Builder compiler also generates an HTML wrapper that you can use when you deploy the application. The following options are available:

- Generate HTML wrapper file
- Check target player version

  When enabled, the compiled application checks for the correct version of Flash Player.

  If Express Install is enabled, the application runs a SWF file in the existing Flash Player to upgrade users to the latest version of the player.

- Enable integration with browser navigation

  This option enables deep linking. Deep linking lets users navigate their interactions with the application by using the Back and Forward buttons in their browser.

Command-line access to the Flex framework compilers

You have direct command-line access to use the Flex framework compilers (mxmlc and compc). When you install Flash Builder, the Flex framework prompt is available from the Windows start menu (Programs > Adobe). For more information, see About the command-line compilers in *Using Adobe Flex 4*.

Customizing project builds

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

Component Set (MX + Spark or MX Only)

By default Flex projects have the entire component set available to applications in the project. This includes Spark components introduced with Flex 4 as well as MX components that were available with Flex 3.

However, in some scenarios you may want to use only the MX components that were available with Flex 3. For example, suppose you have an existing Flex 3 project and you do not want to introduce the new Spark components. But you do want to take advantage of features introduced with Flex 4 and Flash Builder 4, such as the new states syntax, compiler improvements, and other language features. In this scenario, you select the MX Only component set.

When you select the MX Only component set, all Spark related libraries are removed from the build path. If you convert a Flex 4 project to MX Only, Flash Builder does not modify any of the code in the project. You must manually update your code to remove any references to Spark components and libraries.

Application Framework Linkage

By default, application classes for the Flex 4 framework use dynamic linking. Rather than compiling all classes into the application SWF file (static linking), some classes are loaded from the framework runtime shared library (RSL). Applications built with dynamic linking have smaller SWFs, which means they download faster. However, these application use more memory because all framework classes are loaded, not just the classes you need. For more information, see Runtime Shared Libraries.

You can modify a project's properties to customize this behavior for all applications in a project. After selecting a project, from the Flash Builder menu, select Project > Properties > Flex Build Path > Library Path.

By default, Flash Builder uses the SDK's default behavior for framework linkage. For Flex 4, the default behavior is dynamic linking of RSLs. For Flex 3, the default behavior is static linking. Use the Framework Linkage drop-down menu to override the default behavior.

For Flex 4 SDK, the following options are enabled by default:

- Verify RSL Digests

  Verifies that the digest of the RSL matches the digest that was stored in the application at compile time when the application was linked to the cross-domain RSL. For more information, see About RSL digests.

- Use Local Debug SWF RSL When Debugging

  Use local RSLs when debugging the application. Using local RSLs allows you to step into debug RSL files. This option is ignored when exporting a release build.

- Automatically Determine Library Order Based On Dependencies

  If enabled, Flash Builder determines the library order, based on dependencies in the libraries. To customize the library order, disable this option and use the up and down buttons tor specify a library order.

Enabling and disabling automatic builds

In the stand-alone configuration of Flash Builder, your projects are built automatically. In the plug-in configuration, you need to select this option yourself. Flash Builder is designed to automatically build your projects; turning this option off prevents the compiler from identifying syntax errors and displaying warning and error messages as you enter code. For more information about building your projects manually, see "Building projects manually" on page 76.

Do one of the following:

- Select Project > Build Automatically.
- Open the Preferences dialog and select the General > Workspace. Select or deselect the Build Automatically option.

The Build Automatically option affects all projects in the workspace.

Setting up a project output folder

When you create a project in Flash Builder, by default, the build output is generated into the output folder. This does not apply to LiveCycle Data Services ES projects that use the server compile option because the application is compiled on the server when you run it.

You can change the name of this folder when you create the project or after the project is created. You can either create a folder or select an existing folder in the workspace.

1 In the Flex Package Explorer, select a project.

2 Right-click (Control-click on Macintosh) and select Properties from the context menu.

The Project Properties dialog box appears.

3 Select the Flex Build Path properties page.

4 Change the existing output folder by entering a new name or by navigating to an existing folder in your project and selecting it.

Note: *You cannot change the output folder of a LiveCycle Data Services ES application in this manner because its location is controlled by the Flex server and is accessible only through the project's Flex-config.xml file.*

5 Click OK.

The existing output folder is replaced by the new output folder.

Important: *When you change the name of the output folder, the original output folder and all of its contents will be deleted. You will need to rebuild the project to regenerate the application SWF and HTML wrapper files.*

Modifying a project build path

Each project has its own build path, which is a combination of the source path and the library path. (Library project build paths are a little more complex. For more information, see "Library projects" on page 66.) The source path is the location of the project MXML and ActionScript source files. The library path is the location of the base Flex framework classes and any custom Flex components that you have created, in the form of SWC files.

Modify the source path

1 Select a project in the Flex Package Explorer.

2 Right-click (Control-click on Macintosh) and select Properties from the context menu. The Project Properties dialog box appears.

3 Select the Flex Build Path properties page. (If you're working with an ActionScript project, select the ActionScript Build Path properties page.)

4 Add a folder to the source path by clicking the Add Folder button.

5 Enter a name for the folder or click the Browse button to select the location of the custom classes.

You can also use path variables rather than entering the full path to the file system. You can either enter the name of an existing path variable or create a new path variable; for more information, see "Creating a path variable" on page 76.

6 Modify the source path as needed, and click OK.

Modify the library path

1 Follow steps 1 through 3 of the previous procedure to access the Flex Build Path properties page.

2 Click the Library Path tab.

The library path contains references to the Flex framework classes, which are contained in SWC files. A SWC file is an archive file for Flex components and other assets (for more information, see "Using SWC files in your projects" on page 68).

You can edit the path to the framework or, if you created custom Flex components, add new folders or SWC files to the library path. You can also remove items from the path.

3 Modify the library path as needed, and click OK.

Creating a path variable

Rather than linking to resources by entering the full path to the local or network folder where you store your files, you can define path variables. For example, you can define a path variable called Classes and then set the path to a folder on the file system. You then select Classes as the location of the new linked folder. If the folder location changes, you can update the defined path variable with the new location and all the projects that are linked to Classes continue to access the resources.

Set or create a path variable

1 Select a project in the Flex Package Explorer.

2 Right-click (Control-click on Macintosh) and select Properties from the context menu. The Project Properties dialog box appears.

3 Select the Flex Build Path properties page. (If you're working with an ActionScript project, select the ActionScript Build Path properties page.)

4 You can create a path variable for any item on the path (this includes folders in the source path and SWC folders, projects, and SWC files in the library path). As an example, on the Source Path tab select the Add Folder button. The Add Folder dialog box appears.

5 Enter a path variable using the following format: `${`*pathvariablename*`}`.

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

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

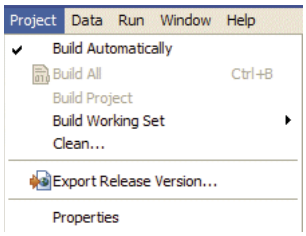
Advanced build options

Flash Builder has advanced options for customizing project builds. You can, for example, builds projects manually, change the default build order of projects in the workspace, and create custom builders using the Apache Ant utility.

Building projects manually

When you build projects manually, you can control the timing and scope of the build. For example, you can build a single project, all projects in the workspace, or create a working set of projects or selected project resources and build only those projects and resources. A working set is a collection of workspace resources (projects, files, and folders) that you can select and group together and work with as you see fit. For more information about working sets, see "Creating working sets" on page 27.

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



Build a single project

1 In the Flex Package Explorer, select the project you want to build.

2 Select Project > Build Project from the main menu.

The selected project is built, and new or updated release and debug application files are added to the project output folder.

Note: *If any of your project files need to be saved, you are prompted to do so before the build begins. To bypass this save prompt, you can set workspace preferences to save files automatically before a build begins.*

Build all projects in the workspace

❖ Select Project > Build All from the main menu.

All projects in the workspace are built and application files are added to each project output folder. You are then prompted to save files if you have not already chosen to save files automatically before a build begins.

Build a working set

Do either of the following:

- Select Project > Build Working Set > Select Working Set from the main menu. Click New to create a working set. For more information about creating a working set, see "Creating working sets" on page 27.
- Choose an existing working set by selecting Project > Build Working Set > Select Working Set from the main menu.

All projects in the working set are built and the application files are added to the project output folder.

Saving project resources automatically

When you build your projects manually, you are prompted to save all resources before the build begins. To bypass this prompt, you can set workspace preferences to automatically save project resources.

1 Open the Preferences dialog, select General > Workspace.

2 Select the Save Automatically Before Build option.

3 (Optional) You can modify how often resources are saved by entering a value (in minutes) in the Workspace Save Interval text box.

Performing a clean build

After a project has been built, subsequent builds affect only the resources that have been added or modified. To force the Flash Builder compiler to rebuild all resources in a project, you can perform a clean build. You might perform a clean build if, for example, you want to eliminate all potential sources of a problem you encountered when testing your application.

1 Select Project > Clean from the main menu.

2 Select the project (or projects) whose build files you want to discard and rebuild from scratch.

3 Click OK.

Changing the project build order

Flash Builder lets you create relationships between projects when working with multiple projects in the workspace. For example, you can import ActionScript classes from one project into another. Creating relationships between projects affects the order in which your projects are built.

By default, the compiler builds related projects in the order required to build them all properly. For example, if a project refers to classes contained in another project, the project containing the classes is built first. In most cases, relying on the compiler to build projects in the proper order is sufficient and your applications will be generated successfully.

You can, however, change the build order. For example, you might change the build order if you created a custom Ant builder and associated it with a project in your workspace, and you need to build that project before other projects are built. For more information about creating custom builders, see "Customizing builds with Apache Ant" on page 78.

1 Open the Preferences dialog and select General > Workspace > Build Order.

The Build Order dialog box displays the following options:

Use Default Build Order The default build order is dictated by the dependencies between projects and is handled by the compiler.

Project Build Order You can manually set the build order for all the projects in the workspace. You can also remove a project from the build order list; it will still be built, but only after all the projects in the build order list.

Max Iterations When Building With Cycles If your projects contain cyclic references (something you should avoid), you can set the number of build attempts so that the compiler can properly build all the projects. The default maximum number of iterations is 10.

2 Modify the build order as needed, and click OK.

Customizing builds with Apache Ant

By creating a custom builder, you can modify and extend the standard build process. Flash Builder contains a standard build script that is used to compile your applications. If needed, you can create custom build scripts using Apache Ant, which is an open-source Java-based build tool.

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 Flash Builder project types.

Create a builder

1 In the Flex Package Explorer, select a project and then right-click (Control-click on Macintosh) to display the context menu and select Properties.

2 Select the Builders properties page. If you're using other Eclipse plug-ins, there may be more than one builder listed. Flash Builder provides a builder named Flex, which you cannot modify.

3 Select New.

4 In the Choose Configuration Type dialog box, select the appropriate configuration type. Flash Builder supports the program type. Select it and click OK to continue. From the new builder properties page you define the builder properties and reference the Ant script (an XML file).

5 Click OK to apply it to the project.

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

Using multiple SDKs in Flash Builder

Flash Builder lets you change the version of the SDK that you use to compile your projects. You can select the SDK when you first create a project or at any time you are working on a project.

The combination of a framework and the compiler make up the SDK. If you select the Flex 2.0.1 SDK, then you are using the 2.0.1 version of the Flex framework SWC files, and the 2.0.1 version of the Flex compiler. You cannot use, for example, the Flex 3 compiler with the Flex 2.0.1 framework SWC files.

Using a different SDK can be useful if you are given a project that was developed using Flex Builder 2.0.1 (which uses the Flex 2.0.1 SDK), but you are running Flash Builder 4 (which uses the Flex 4 SDK by default). By selecting an older SDK to build with, you can maintain projects that have not been updated to be compatible with the latest version of the SDK. In addition, if you are currently working on a project for the Flex 2.0.1 SDK, but want to use the Flash Builder 4 features such as code refactoring, you can upgrade your edition of Flash Builder, but then select the older SDK as the default SDK.

If you develop a project and then change the SDK, Flash Builder performs a full rebuild, not an incremental build. As a result, Flash Builder flags any differences that would throw compiler errors as if the project had been developed under the original SDK. For example, if you use an AdvancedDataGrid component in your project (which was first introduced in the Flex 3 SDK), but then change the project to use the 2.0.1 SDK, Flash Builder will notify you that the AdvancedDataGrid class is unknown.

Flash Builder also regenerates all supporting files for the projects. These include the history management and deep linking files used by the HTML wrapper. For Flex 2.0.1 SDK projects, Flash Builder creates the Flex 2.0.1. SDK–compatible history.swf, history.html, and history.js history management files in the html-templates directory. For Flex 3 SDK projects, Flash Builder creates the Flex 3 SDK–compatible deep-linking history.htm, history.js. and historyFrame.html files in the html-templates/history directory.

In addition, the availability of Flash Builder options change depending on the selected SDK. For example, if you add a module to your project with a project that uses the Flex 2.0.1 SDK, Flash Builder does not let you select whether you want to optimize that module or not. You must do this manually.

For more information about the differences between the Flex 3 SDK and the Flex 2.0.1 SDK, see Backward compatibility in *Using Adobe Flex 4*.

When you create a new Flex project, Flash Builder uses the default SDK. The default is the latest SDK that shipped with Flash Builder, but you can change it to any SDK that is visible in the list of available SDKs in Flash Builder.

When you create a new Flex library project or ActionScript project, you can select which SDK you want that project to use in the New Flex Library Project and New ActionScript Project dialog boxes.

Add a new Flex SDK to the list of available SDKs

1. Open the Preferences dialog and select Flash Builder > Installed Flex SDKs.

   The currently installed SDKs are listed. The default SDK has a check mark next to its name.

2. Click Add.
3. Enter the location of the SDK in the Flex SDK Location field.
4. Enter a name for the SDK in the Flex SDK Name field. You should not use the name of an existing SDK for this field.
5. Click OK to save your changes.
6. Click OK again to add the new SDK to the list of available SDKs. This list is maintained in the Flash Builder workspace, across Flex projects. The next time you create a new project, the list of available SDKs includes this new SDK.

Change the SDK version for the current project

1. Select Project > Properties.
2. Select Flex Compiler.
3. Click Use a Specific SDK.

4 Select the SDK you want to use from the drop-down list. If the SDK you want to use is not in the drop-down list, click the Configure Flex SDKs link.

5 Click OK.

Flash Builder applies the new SDK to the current project. This can cause errors and warnings to appear if the project uses code that is not compatible with the new SDK.

Select a new default SDK

1 Open the Preferences dialog, select Flash Builder > Installed Flex SDKs.

The default SDK has a check mark next to its name.

2 Select the check box next to an SDK. This SDK becomes the default SDK. It is also applied to any project that has the Use Default SDK option selected in the Flex Compiler dialog box, including the current project. If the current project is set to use a specific SDK, then it will continue to use that specific SDK, even if you change the workspace's default SDK to a different one.

3 Click OK to save your changes.

4 Click OK again.

Publishing source code

When your applications are ready to be released, Flash Builder lets you choose whether users can view source code and assets in the application. As with HTML, users can access and view the source in a web browser by selecting View Source from the context menu. The source viewer formats and colors the code so that it is easy to read. It is also a convenient way to share code with other Flex and ActionScript 3.0 developers.

Enable the view source option

1 With the completed application project open in the editor, select Project > Export Release Build.

2 Select Enable View Source or Include Source for ActionScript projects.

3 Click Choose Source Files.

4 In the Publish Application Source dialog box, select the application file or files to include in the View Source menu. By default, the main application file is selected.

Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

5 (Optional) Change the source output folder. By default, a source view folder is added to the project output folder.

6 Click OK.

When users run your application, they can access the source code by selecting View Source from the context menu. The source code appears in the default web browser as a source tree that reflects the structure of the resources (packages, folders, and files) contained in your application (the ones that you decided to publish). Selecting a source element displays the code in the browser. Users can also download the entire set of source files by selecting the Download.zip file link.

Note: *Because of Internet Explorer security restrictions, you may not be able to view the source on your local development computer. If this is the case, you will need to deploy the application to a web server to view the source.*

Adding the view source menu to ActionScript projects

In Flex projects you add the View Source menu option to your application with the Export Release Build wizard. In ActionScript applications, you must add this option manually.

The Flex framework contains the following function that you can use in an ActionScript application's constructor to enable the view source menu:

```
com.adobe.viewsource.ViewSource.addMenuItem(obj:InteractiveObject, url:String,
hideBuiltins:Boolean = true)
```

You can use this in your ActionScript applications as shown here:

```
package {
    import flash.display.MovieClip;
    import com.adobe.viewsource.ViewSource;

    public class MyASApp extends MovieClip
    {
        public function MyASApp()
        {
            ViewSource.addMenuItem(this, "srcview/index.html");

            // ... additional application code here
        }
    }

}
```

This example demonstrates adding the view source menu using the default location of the source folder (srcview). If you change the location of the source folder, your code should use the correct location.

Flash Builder command line build

Flash Builder provides the Ant task <fb.exportReleaseBuild>. Use this task to implement command line builds that synchronize a developer's individual build settings with the nightly build. Alternatively, you can use the <mxmlc> task in custom scripts for nightly builds.

<fb.exportReleaseBuild> task

Using the <fb.exportReleaseBuild> task ensures that the nightly build's settings exactly match the settings used by developers during their daily work.

For example, if a developer changes the library path of a Flex project, the new library path is written to the Flash Builder project. When the nightly build machine runs <fb.exportReleaseBuild>, that task loads the Flash Builder project and all of its settings.

Another advantage of using <fb.exportReleaseBuild> is that it automatically takes care of additional "housekeeping" tasks normally included in a Flash Builder build, such as:

- Automatically compile associated library projects
- Copy assets such as JPEGs, and so on, into the output directory
- Copy the HTML template, including macro substitution based on the compilation results (such as width and height)

Note: *The <fb.exportReleaseBuild> task requires you to install Flash Builder on the nightly build machine.*

<mxmlc> task

If you write a custom script that uses the <mxmlc> task (for example, an Ant script), then you do not need to install Flash Builder on the build machine. However, the build machine is required to have the Flex SDK available. Thus, the build machine can be on a Linux platform.

However, the disadvantage of this approach is that you have two sets of build settings to synchronize: One in Flash Builder, used by developers during their daily work, and another on your nightly build machine.

<fb.exportReleaseBuild> usage

1 Install Flash Builder on a build machine.

2 Write `build.xml` with fb.exportReleaseBuild as a target. For example:

```
<?xml version="1.0"?>
<project default="main">
    <target name="main">
        <fb.exportReleaseBuild project="MyProject" />
    </target>
</project>
```

`build.xml` specifies to run a command line build of your Flex project, using the settings saved in your project files. See "Parameters for fb.exportReleaseBuild task" on page 83for details on available parameters.

3 Create a nightly build script that tells Eclipse to look for a build file and execute its target.

The following examples specify `build.xml` as a build file, which executes MyTarget.

If your nightly build script is on a Macintosh platform, you could run the following script:

```
WORKSPACE="$HOME/Documents/Adobe Flash Builder"

# works with either FlashBuilder.app or Eclipse.app
"/Applications/Adobe Flash Builder/FlashBuilder.app/Contents/MacOS/FlashBuilder" \
    --launcher.suppressErrors \
    -noSplash \
    -application org.eclipse.ant.core.antRunner \
    -data "$WORKSPACE" \
    -file "$(pwd)/build.xml" MyTarget
```

If your nightly build is on a Windows platform, you could run the following batch file:

```
set WORKSPACE=%HOMEPATH%\Adobe Flash Builder

REM works with either FlashBuilderC.exe or eclipsec.exe
"C:\Program Files\Adobe\Adobe Flash Builder\FlashBuilderC.exe" ^
    --launcher.suppressErrors ^
    -noSplash ^
    -application org.eclipse.ant.core.antRunner ^
    -data "%WORKSPACE%" ^
    -file "%cd%\build.xml" MyTarget
```

fb.running Ant property

The fb.running Ant property has a value of true when Flash Builder is running. You can use this property when running scripts inside Flash Builder. For example:

```
<target name="myFlashBuilderTasks" if="fb.running">
    <fb.exportReleaseBuild ... />
</target>
```

Eclipse Ant Tasks

Eclipse provides several Ant tasks you can incorporate as targets in your build script. For example:

- eclipse.incrementalBuild
- elicpse.refreshLocal
- eclipse.convertpath

Consult the Eclipse documentation for more information on these scripts.

Parameters for fb.exportReleaseBuild task

Attribute	Description	Required?	Default Value
project	The project to build. Specify the name of a project in your Flash Builder workspace, without a path. For example, "MyFlexProject."	Yes	n/a
application	The name of the application to compile. You can specify just the application name with no path or extension (for example: app1). To avoid ambiguity in naming, you can specify full path, relative to the project root (for example: src/app1.mxml).To compile all applications, specify '*', or omit this attribute. When running against an AIR project, you can only specify a single application. The '*' value is not allowed.	No	The default application of the project.
publishsource	Whether to publish the source of the application, allowing the user to view source files using the context menu View Source.	No	false
locale	Specifies the locale, for example, en-US. This value is passed to the compiler using the compiler's -locale flag. If specified, this overrides any locale that has been specified in Flash Builder's Additional Compiler Arguments field.	No	n/a
destdir	The output folder. The folder can be a relative path or an absolute path. If you specify a relative path, it is relative to the root of the project. If compiling an AIR project, the folder is a temporary directory that is deleted after the .air file has been created.	No	bin-release
failonerror	Indicates whether compilation errors cause the build to fail.	No	true
verbose	The <fb.exportReleaseBuild> task outputs additional information. For example, it lists the files that were packaged into the AIR file and how long each step of the process took.	No	false
package	For AIR projects only: Indicates whether to package the result into a .air or .airi file. If true, a .air or .airi file is created, and the temporary output directory (bin-release by default, set by the destdir attribute) is deleted. If false, a .air or .airi file is not created, and the intermediate directory remains intact after compilation.	No	true

Attribute	Description	Required?	Default Value
destfile	For AIR projects only: The filename of the .air or .airi file to create. You can specify a relative path or absolute path. If you specify a relative path, the path is relative to the root of the project.	No	appname.air or appname.airi (in the project root)
certificate	For AIR projects only: The path to the certificate used to sign the AIR file.	No	If omitted, an unsigned .airi file is generated, which can be signed later.
password	For AIR projects only: The password for the certificate that is used to sign the AIR file. If this argument is omitted, an error message displays. **Caution**: Specifying a literal value for a password can compromise security.	No	n/a
timestamp	For AIR projects only: Indicates whether the generated AIR file includes a timestamp.	No	false

Export Release Build wizard

When you run the Export Release Build wizard (Project > Export Release Build), the settings you make in the wizard are saved in the .actionScriptProperties file. A command line build that uses fb.exportReleaseBuild task picks up the settings from the wizard. The Export Release Build wizard saves the following settings:

- View Source

  The source files you specify for View Source are saved. If you specify the publishsource parameter to fb.exportReleaseBuild, then the wizard includes these files as viewable source files.

  Important: *For server projects, you can select the services folder when exporting source files. Exporting files that implement services has security implications. These files can expose access to your database, including user names and passwords. See Exporting source files with release version of an application.*

- For AIR projects, any additional output files that you specify in the wizard to include with the AIR or AIRI file.

Running command line builds on Linux and other platforms

The <fb.exportReleaseBuild> task is only supported on Windows and Mac platforms.

However, if you are writing a build script for another platform, use the `-dump-config` option to the mxmlc or compc compiler to write compiler configuration settings to a file. You can then use the -load-config option to read the configuration options.

Modify the configuration settings in the file as necessary. For example, change <debug>true</debug> to <debug>false</debug> if your nightly build is supposed to do a release build.

Run a command line build using Flash Builder compiler settings

1. In Flash Builder, select Project > Properties > Flex Compiler
2. In Additional Compiler Arguments, specify the following argument:

   -dump-config *pathname*, where *pathname* specifies the absolute path to a file on your system.

3. Apply the changes in the Project window.

The compiler settings are written to the specified file. Remove the -dump-config argument after you have verified that the file has been written.

4 Modify the configuration settings as necessary.

5 In your build script, run the compiler so it includes the saved compiler settings:

mxmlc -load-config *pathname*

Limitations to command line builds

There are a few limitations to running command line builds using the <fb.exportReleaseBuild> task.

Running command line builds on Mac platforms using Eclipse 3.4

On Eclipse 3.4 on Mac platforms, headless build fails if the Eclipse installation path contains spaces.

Running command line builds on 64-bit platforms

Flash Builder runs on platforms that implement 32-bit Java. To run a command line build on platforms that support 64-bit Java (for example, Mac OS X Snow Leopard), add `-d32` to the command-line options that are passed to Java. For example:

```
java -d32 ...
```

Running applications

When you test your applications in Adobe® Flash® Builder™, you run the application SWF files in a web browser or the stand-alone Flash Player. If you encounter errors in your applications, you use the debugging tools to set and manage breakpoints in your code; control application execution by suspending, resuming, and terminating the application; step into and over the code statements, select critical variables to watch, and evaluate watch expressions while the application runs.

About running and debugging applications

After your projects are built into applications (see "Building projects" on page 69), you can run and debug them in Flash Builder.

Use debugger version of Flash Player

To use the Flash Builder debugging feature, you must run the debugger version of Flash Player. This version is available as a browser plug-in or ActiveX control, or as a stand-alone version. This is the version that is installed with Flash Builder, but it is also available as a download from the Adobe web site.

The installers for the debugger version of Flash Player are located in the *flex_builder_install*/Player directory.

You can programmatically determine which version of Flash Player you are running by using the `Capabilities.isDebugger()` method. For more information, see Determining Flash Player version in Flex.

Use launch configurations to run and debug applications

A launch configuration defines the project name, main application file, and the path to the run and debug versions of the application. Flash 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 88.

Note: *In the plug-in configuration of Flash 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 87.*

Customize launch configurations

You can customize the launch configurations that Flash Builder creates automatically for you. For example, you can switch the main application file or modify the path to point to the SWF file rather than the HTML wrapper file so that you can run and debug your applications directly in the stand-alone Flash Player instead of a web browser. You can also create separate launch configurations for each of the application files in your project. For more information, see "Creating or editing a launch configuration" on page 88.

Run and debug applications in a browser or the stand-alone Flash Player

By default, the launch configuration specifies that the run and debug paths point to the HTML wrapper files in the output folder of your project; therefore, your applications are run and debugged in Flash Player running in a web browser. Instead you can run and debug your applications in the stand-alone Flash Player (see "Running the application SWF file in the stand-alone Flash Player" on page 90). 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 90).

Note: *The default security settings for Internet Explorer 6 (or later) block applications launched from Flash Builder. You can change your Internet Explorer security settings by selecting Tools > Internet Options > Advanced > Security > Allow active content to run in files on My Computer. Or you can explicitly allow each Internet Explorer page to run after it is blocked.*

Run debugging tools from the code editor

The Flash Debug perspective provides all the debugging tools you expect from a robust, full-featured development tool. The debugging tools allow you to:

- Set and manage breakpoints
- Determine how to suspend, resume, and terminate the application
- Step into and over code
- Watch variables
- Evaluate expressions

For more information about the Flash Builder debugging tools, see "Debugging your applications" on page 132.

For more information about code editing, see "About code editing in Flash Builder" on page 96.

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 Flash Debug perspective is then activated and you can inspect the state of and manage the application by using the debugging tools. For more information, see "Starting a debugging session" on page 132.

Compare debug and non-debug versions of your Flex application

By default, Flash Builder generates debug versions of your Flex application's SWF file and stores them in your project's bin-debug directory. This application is larger than the non-debug version because it includes additional code and metadata that the debugger uses.

To generate a non-debug version of your Flex application, you can do one of the following:

- Select Project > Export Release Build. This creates a non-debug SWF file or AIR file in the bin-release directory.
- Add `-debug=false` to the Additional Compiler Arguments field. This generates a non-debug SWF file no matter where you export it to.

Running your applications

Your applications are run (and debugged) based on a launch configuration. When you create new Flex and ActionScript applications, a launch configuration specifies the location of the built applications files and the main application file. You can modify the launch configuration or create custom launch configurations. For more information, see "Managing launch configurations" on page 88.

Running your projects opens the main application SWF file in your default web browser or directly in the stand-alone Flash Player. For information about changing the default web browser or running and debugging with the stand-alone Flash Player, see "Changing the default web browser" on page 90 and "Running the application SWF file in the stand-alone Flash Player" on page 90.

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

Note: *The Run button has two elements: the main action button, and a pop-up menu that shows the application files in the project that can be run or debugged. When you click the main action button, the default application file is run. Alternatively you can click the pop-up menu and select any of the application files in the project and create or edit a launch configuration in the Create, Manage, and Run Configurations dialog box.*

Run project with default Flex application launch configuration

1. In the Flex Package Explorer, select the project to run.
2. On the main workbench toolbar, click the Run button.

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

   Your application appears in your default web browser or the stand-alone Flash Player.

You can run and debug any project files that have been set as application files. For more information see "Managing project application files" on page 50.

Create custom launch configuration

1. In the Flex Package Explorer, select the project you want to run and open the main application file in the editor.
2. From the main menu, select Run > Run > Other.
3. In the Create, Manage, and Run Configurations dialog box, select either Web Application or Desktop Application.
4. Click the New button on the dialog box toolbar.
5. Enter the launch configuration name.
6. (Optional) Modify the configuration properties as needed.
7. (Optional) Click Run to run the application.

Run other application files in your project

1. In the Flex Package Explorer, select the project to run.
2. From the main menu, select Run > Run > Other.

3 Select the application you want to run.

Run a custom launch configuration

❖ From the main menu, select Run > Run > Other.

In the Create, Manage, and Run Configurations dialog box, select an existing custom launch configuration or create a new one. See "Managing launch configurations" on page 88.

The Run command works a bit differently in the plug-in configuration of Flash Builder. Instead of running the currently selected project, it runs the most recently launched configuration. You can also select from a list of recently launched configurations.

Run the last launched configuration

❖ Click the Run button on the main toolbar.

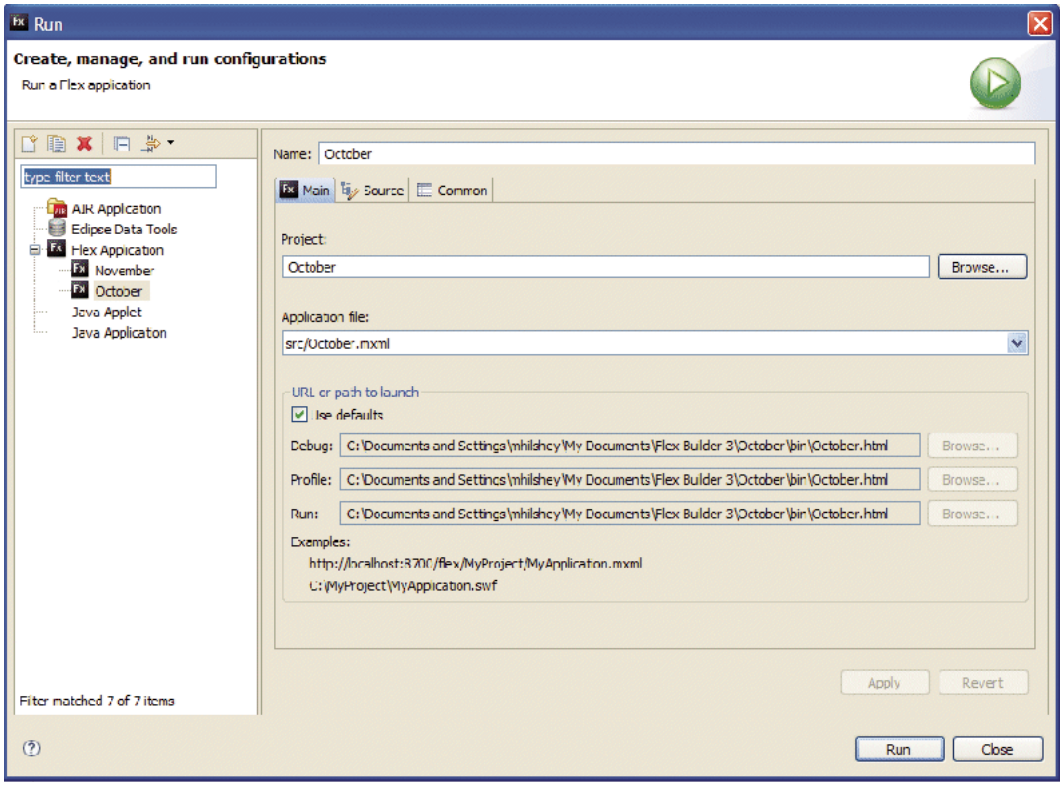
Managing launch configurations

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

Creating or editing a launch configuration

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

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



Access and edit a launch configuration

1 In the Flex Package Explorer, select a project.

2 With a project file open in the code editor, open the Create, Manage, and Run Configurations dialog box. The way to do this depends on your Flash Builder installation.

- Stand-alone Flash Builder: Select the project and then select Run > Run > Other or Run > Debug > Other.
- Flash Builder plug-in with Eclipse 3.2: Right-click (Control-click on Macintosh) to display the context menu and select Run As > Run or Debug As > Debug.
- Flash Builder plug-in with Eclipse 3.3: Right-click (Control-click on Macintosh) to display the context menu and select Run As > Open Run Dialog or Debug As > Open Debug Dialog.

3 Select the launch configuration to edit.

A number of configurations may be listed if you have other projects in the workspace, if you have set other project files as application files, or if other Eclipse plug-ins are installed.

By default, the first time you run a project, Flash Builder creates a project-specific launch configuration, which is based on the default Flex application launch configuration.

You can edit application-specific configurations. You can also create new a launch configuration or base a new configuration on an existing configuration.

4 Modify the configuration preferences as needed, and click Run or Debug.

Running the application SWF file in the stand-alone Flash Player

Your applications are run or debugged in the default web browser. You can change the default web browser to use run and debug applications (see "Changing the default web browser" on page 90). You can also choose to run and debug your applications in the stand-alone Flash Player by making a simple change to the launch configuration.

Run and debug applications in the stand-alone Flash Player

1. From the Create, Manage, and Run Configurations dialog box, select the launch configuration you want to modify.
2. In the Main tab, deselect Use Defaults.
3. In either or both of the Run and Debug paths, click Browse.

   The file selection dialog box appears and lists the contents of the build output folder.
4. Select the application SWF file in the bin-debug directory. Do not select the SWF file in the bin-release directory, if there is one. This SWF file does not contain debug information.
5. Click Open to select the file and return to the configuration dialog box.
6. Apply your changes and use the modified configuration to run or debug the application.

Changing the default web browser

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

Change the default web browser

1. Open the Preferences dialog and select General > Web Browser.
2. Select a web browser from the list of web browsers installed on your system.

   Note: *The Use Internal Web Browser option does not apply to running and debugging applications. Applications are always run and debugged in an external web browser.*

   You can also add, edit, and remove browsers from the list.
3. Click OK to apply your changes.

Creating Modules

You can create, add, optimize, and debug modules in Adobe® Flash® Builder™. For information on writing module code, see Creating modular applications.

Creating modules in Flash Builder

The following steps describe how to create a new module in Flash Builder. After you create a new module, you can compile it.

Create modules in Flash Builder

1. In Flash Builder, select File > New > MXML Module. The New MXML Module dialog box appears.
2. Select a parent directory for the module. You typically store modules in the same directory as the main application so that relative paths to shared resources are the same.

Because modules are runnable, they must be in the project's source folder.

3 Enter a filename for the module; for example, MyModule.

4 Enter the Width, Height, and Layout properties for the module.

5 Specify whether to optimize the module.

If you optimize a module for an application, classes used by the application are excluded from the module. This can result in smaller download sizes for your SWF files. For more information, see "Optimizing modules in Flash Builder" on page 94.

- Optimize for Application

  If you select this option, specify an application to optimize the module against.

- Do Not Optimize

  If you select this option, all classes are included in the module, whether or not they are defined in the main application. This can improve the performance of the incremental compilation. In addition, you can load the module into any application because it has all of its dependencies compiled into it.

6 Click Finish.

Flash Builder adds a new MXML module file in your project.

Compiling modules in Flash Builder

In Flash Builder, you can either run the module as if it were an application or you can build the module's project. If the modules are in the same project as your application, then when you run your application, Flash Builder compiles SWF files for all modules in the project. The SWF files are then loaded into the application at run time.

You cannot run the module-based SWF file as a stand-alone Flash application or load it into a browser window. It must be loaded by an application as a module. If you run the module in Flash Builder to compile it, close Adobe Flash Player or the browser window and ignore any errors. Modules should not be requested by the Player or through a browser directly.

The module SWF files and main application SWF file are typically in the same directory, although Flash Builder compiles the modules at the same time as your application, regardless of their location. Modules can be in the same directory as the application or in subdirectories.

You can also create a separate Flex or ActionScript project for each module or for groups of modules. This gives you greater control over how modules are compiled because each project can have different compiler options than the application or other modules. It also lets you compile the module's project or projects without compiling the application. However, this approach requires that you manually compile each module before compiling the application. One way to do this is to compile all open projects in Flash Builder at one time.

If you compile modules separately from the main application, be sure to include or exclude debugging information, based on whether you want to debug your application and modules. For more information, see "Debugging modules in Flash Builder" on page 95.

The Flash Builder workflow is designed around associating modules with a single application. If you want to use modules across multiple applications, consider encapsulating the code in a library component or class and including that in a simple module for each application. Modules are not intended to be used for cross-application code reuse; that is for libraries.

Using multiple projects for modules

When you set up your project's architecture, you can decide to include modules in your application's project, create a separate project for each module, or create a separate project for all modules.

Using one project for each module

Using one project for each module has the following benefits:

- Module projects can be located anywhere in the workspace.
- Module projects can have their own compiler settings, such as a custom library path.

  Module projects can use the `load-externs` compiler option to remove overlapping dependencies.

Using one project for each module has the following drawbacks:

- Many projects uses more memory.
- Many projects in a single workspace can make the workspace crowded.
- By default, when you compile the application, not all module projects are compiled even if they have changed.
- To optimize your module's file size, manually apply the `load-externs` and `link-report` compiler options.

Using one project for all modules

A related approach is to use a single project for all modules, while keeping the application in its own separate project. This has some of the drawbacks of using a single project for both the application and the modules, but it has many of the same benefits as using a separate project for each module.

Using one project for all modules has the following benefits:

- The module project can be located anywhere in the workspace.
- You can compile just the modules or just the application, without having to recompile both at the same time.
- The module project can use the `load-externs` compiler option to remove overlapping dependencies.

Using one module project for all modules has the following drawbacks:

- All of the modules in the module project must use the same compiler settings, such as the library path.
- By default, when you compile the application, the module project is not compiled even if the module project has changed.
- To optimize your module's file size, you must manually apply the `load-externs` and `link-report` compiler options.

Creating projects for modules

When creating a separate project for modules, you change the module project's output folder to a directory that is used by the application. You also suppress the generation of wrapper files.

Create a separate project for modules in Flash Builder

1. Create a main project.
2. Create a new project for your module or modules.
3. From the context menu for the module's project select Properties. The Properties dialog box appears.
4. Select the Flex Build Path option.
5. Change the Output Folder to point to the MainProject modules directory. For example, change it to the following:

   ```
   ${DOCUMENTS}\MainProject\assets
   ```

   This redirects the output of your module's compilation to your application project's (MainProject) assets directory. In your main application, you can point the ModuleLoader `url` property to the SWF files in the assets directory. The value of this property is relative to the output folder.

6 Click OK to save your changes.

7 Open the project properties again and select the Flex Compiler option.

8 Deselect the Generate HTML Wrapper File option. This prevents the module's project from generating the HTML wrapper files. You typically use these files only for the application. For modules, they are not necessary.

9 Click OK to apply the changes.

Compiling projects for modules

Compiling multiple projects in Flash Builder is a common operation. First you choose the order in which you want the projects to be compiled and then you compile all projects at the same time.

Compile all projects at the same time in Flash Builder

❖ From the main menu, select Project > Build All.

Flex builds all projects in the workspace. The application files are added to each project's output folder. If you haven't already chosen to save files automatically before a build begins, you are prompted to save the files.

If you want to change the build order, use the Build Order dialog box. Changing the build order is not always necessary. Projects that use modules need to be compiled only by the time the main project application runs, not as it is compiled. In most cases, the default build order is adequate.

However, if you want to eliminate overlapping dependencies, you might need to change the build order so that the main application is compiled first. At that time, you use the `link-report` compiler option to generate the linker report. When you compile the modules, you use the `load-externs` compiler option to use the linker report that was just generated by the shell application. For more information on reducing module size, see "Optimizing modules in Flash Builder" on page 94.

Change the build order of the projects

1 Open the Preferences dialog and select General > Workspace > Build Order.

The Build Order dialog box appears.

2 Deselect the Use Default Build Order checkbox.

3 Use the Up and Down buttons to reorder the projects in the Project Build Order list. You can also use the Remove Project button to remove projects that are not part of your main application or that are not modules used by the application. The removed project is built, but only after all the projects in the build order list are built.

4 Click OK.

5 Modify the build order as needed and then click OK.

If you create dependencies between separate projects in the workspace, the compiler automatically determines the order in which the projects are built, so these dependencies are resolved properly.

When you use a separate project for each module, you can compile a single module at a time. This can save time over compiling all projects at once, or over compiling a single project that contains all module and application files.

Compile a single module's project

1 Right-click the module's MXML file in the module's project.

2 Select Run Application. The Player or browser window tries to run the module after it is compiled. You can close the Player or browser window and ignore any error messages that appear at run time. Modules are not meant to be run in the Player or in a browser directly.

Adding modules to your project

In some cases, you use modules that are not in your main application's project. You might have the module in a separate project so that you can use custom configuration options, or you might want to share the module across multiple applications. Add the module's source code to your application's source path and then add the module to the application's module list before you can use it in your project.

Add an already-compiled module to your project

1. Select the main application in the Flex Package Explorer.
2. Select Project > Properties > Flex Build Path to add the module's source to the application project's source path.
3. Click the Add Folder button and browse to the module's source path. Click OK to select the module. You must do this for each external module that you add to your application's project.
4. Click OK again to save your changes.
5. Click Project > Properties > Flex Modules to add the module to the application's module list. The Flex Modules dialog box lists all modules that have been added to the current project or that are in the current project. When you first create a project, this dialog box is empty.
6. Click the Add button. The Add Module dialog box appears.
7. Use the Browse button or enter the location of the module's MXML file in the Source text box. All modules that are in the project's source path are available to add by using this dialog box.
8. Select one of the radio buttons under Module SWF Size to enable or disable module optimization. If you select Optimize for Application, Flash Builder compiles the module against the selected application and excludes all classes that are defined in the main application. These can include framework classes or custom classes. When you select this option, you cannot use the same module in another application, because the list of excluded classes might be different. For more information, see "Optimizing modules in Flash Builder" on page 94.
9. Click OK to save your changes. Flash Builder adds the module to the list of available modules in your application's project.

Optimizing modules in Flash Builder

In Flash Builder, you typically select a single application to optimize the module against when you first create the module or add it to a project. If you later decide to change the application that you optimize the module against, or if do not want to optimize the module, you can edit the module's properties within the project. For more information, see Reducing module size.

Optimize modules with Flash Builder

This procedure assumes the module and application are in the same Flash Builder project. If the modules are in a separate project, manually add the `load-externs` and `link-report` compiler options.

1. Right-click the application's project in the Flex Package Explorer and select Properties. The project's Properties dialog box appears.
2. In the left pane, select Flex Modules.
3. From the list of modules, select the module and then click the Edit button. The Edit Module dialog box appears.
4. To remove optimization, select the Do Not Optimize radio button under Module SWF Size.
5. To optimize the module for a different application, select the new application from the Optimize for Application pop-up menu.
6. Click OK.

To further optimize a module's file size, you can remove debugging information. If you build a module in Flash Builder, debugging information is included in the module by default. By removing debugging information, you can further reduce the size of the module. For instructions on how to remove debugging information from modules, see "Debugging modules in Flash Builder" on page 95.

Debugging modules in Flash Builder

To debug modules and applications, you must include debug information in the SWF files when they are compiled. To do this in Flash Builder, just run the application, because debug information is included by default. On the command line, you set the `debug` compiler option to `true`. The default is `true`, but if you disabled it in a configuration file, you must be sure to override it.

By default, Flash Builder builds a single SWF file that includes debug symbols, so both Run and Debug work when you execute an application that uses modules in Flash Builder. However, including debug symbols in a module's SWF file makes the SWF file larger. To exclude debug symbols prior to deployment, you must disable debugging for the application's modules. To do this, you export the release version of the modules by selecting Project > Export Release Build.

To exclude debugging information from SWF files in Flash Builder, you can either set the `debug` option to `false` in the Additional Compiler Arguments text box, or you can output the SWF files by using the Export Release Build feature, which generates non-debug SWF files. This includes the modules, if those modules are in the current project.

If you create a separate project for your modules, you can enable or disable debugging for the entire project, without changing the settings of your main application.

When you want to debug a module, you must also debug the application that loads the module. The Flex debugger will not connect to an application that does not contain debug information, even if the modules that the application loads contain that information. In other words, you cannot exclude debug information from the application if you want to debug the module that the application loads.

When you're using modules in an AIR application, the module SWF must be located in the same directory as the main application SWF or one of its subdirectories.