macromedia®

FLEXBUILDER™

Using Flex Builder

Acknowledgments

Senior Management: Sheila McGinn

Project Management: Charles Nadeau

Writing: Charles Nadeau

Editing: Lisa Stanziano, Mary Ferguson, Rosana Francescato, Linda Adler

Production Management: Patrice O’Neill

Media Design and Production: John Francis, Adam Barnett, Aaron Begley

Special thanks to Heidi Bauer-Williams, Gabe Chua, Larry McLister, Paul Gubbay, Brad Becker, David Lenoe, Lucian Beebe, Lori Hylan-Cho, Mark Shepherd, Tanya Dimalsky, Matthew Hoffberg, Jon Varese, Dave George, Brandon Purcell, Stephen Gilson and the Flex IMD team, the beta testers, and the entire Flex Builder engineering and QA teams.

First Edition: July 2004

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

CHAPTER 1: Getting Started with Flex Builder 5

Exploring the authoring environment 5
Starting a new application in Flex Builder 9
Defining a site for the Flex Sample Apps application 12
Getting help in Flex Builder 17
What you need to build Flex applications 18
Typographical conventions of this guide 20

CHAPTER 2: Flex Builder Tutorials 21

Tutorial: Setting up a development environment 21
Tutorial: Creating a layout with Flex Builder 26
Tutorial: Building custom components with Flex Builder 35
Tutorial: Binding components to data with Flex Builder 59
Next steps 75

CHAPTER 3: Creating, Coding, and Debugging Flex Files 77

Creating MXML or ActionScript files 77
Working with the code in Flex files 78
Testing and debugging Flex files 86

CHAPTER 4: Building a Flex User Interface Visually 97

Customizing the Document window 98
Laying out your user interface with MXML containers 99
Adding Flex components to the user interface 100
Repositioning Flex components 101
Resizing Flex components 102
Cutting, copying, and pasting Flex components 102
Editing Flex component properties 103
Applying CSS styles to Flex components 104
Applying effects to Flex components 111
Assigning event handlers to components 116

CHAPTER 5: Working with Components 119

Inserting controls and containers in an MXML file 119
Creating MXML component files 123
Synchronizing with custom components on the server 127

CHAPTER 6: Working with Data 129

Binding a control or container. 129
Editing a binding visually 131
Working with Flex data models 131
Working with Flex data services. 138

APPENDIX A: Basic Flex Concepts 147

About the Flex application model 147
Typical application development steps. 148
About Flex files 148
About Flex containers, components, and controls 150
About Flex data bindings. 152

INDEX 157

CHAPTER 1
Getting Started with Flex Builder

This chapter provides an overview of Macromedia Flex Builder and steps to get started developing Flex applications.

Flex Builder is an integrated development environment (IDE) for Flex developers. Because it's tightly integrated with the Macromedia Flex server, the Flex application model, and the Flex programming languages, Flex Builder can improve the productivity of all members of your development team.

This guide focuses on Flex Builder. If you're new to Flex, you may want to review Appendix A, "Basic Flex Concepts," on page 147. For in-depth information about Flex (as opposed to Flex Builder), see Developing Flex Applications Help and the Flex ActionScript and MXML API Reference Help.

The following topics are covered in this chapter:

- "Exploring the authoring environment" on page 5
- "Starting a new application in Flex Builder" on page 9
- "Defining a site for the Flex Sample Apps application" on page 12
- "Getting help in Flex Builder" on page 17
- "What you need to build Flex applications" on page 18
- "Typographical conventions of this guide" on page 20

Exploring the authoring environment

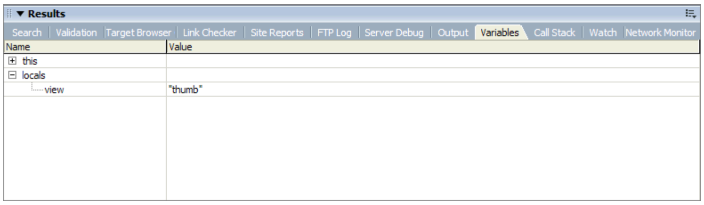
The Flex Builder IDE lets you write, edit, debug, and preview MXML and ActionScript files, and includes the following features:

The Start page provides links to quickly define a site, create and open files, and access useful information about Flex Builder. For example, you can take a quick tour, explore Flex samples, take tutorials, and learn about debugging in Flex Builder.

Design view (View > Design) lets you lay out and edit an MXML user interface visually. Design view has two modes: Expanded and Standard. In Expanded mode, you can easily insert, edit, position, or resize MXML components because Flex Builder adds temporary borders and padding around objects. In Standard mode, Flex Builder removes the borders and padding to give you a more accurate picture of how the MXML file will look in Macromedia Flash Player.

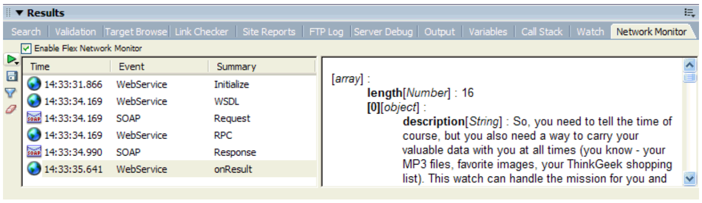
Code view (View > Code) lets you work with code. MXML and ActionScript syntax is color-coded and code hints assist you as you type. Design view tools such as the Insert bar, Tag inspector, and Data panel are also fully functional in Code view.

The ActionScript debugging panels help you debug your code after setting breakpoints and switching to debug mode. The debugging panels include the Variables panel, the Watch panel, the Output panel, and the Call Stack panel. For more information, see "Debugging ActionScript" on page 90.



Name	Value
this	
locals	
view	"thumb"

The Network Monitor panel helps you monitor network traffic and data going back and forth between the client and the Flex server. For more information, see "Debugging applications by monitoring interactions with the server" on page 93.

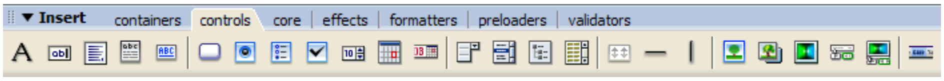


Enable Flex Network Monitor

Time	Event	Summary
14:33:31.866	WebService	Initialize
14:33:34.169	WebService	WSDL
14:33:34.169	SOAP	Request
14:33:34.169	WebService	RPC
14:33:34.990	SOAP	Response
14:33:35.641	WebService	onResult

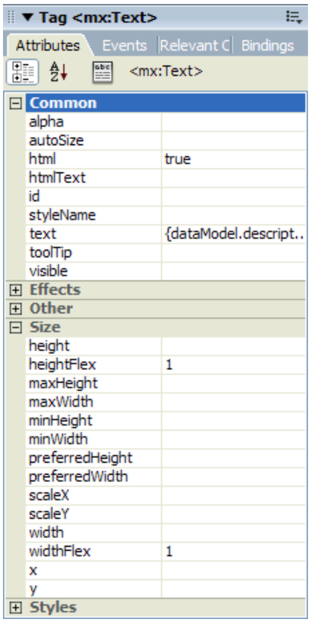
```
[array] :
    length[Number] : 16
    [0][object] :
        description[String] : So, you need to tell the time of
        course, but you also need a way to carry your
        valuable data with you at all times (you know - your
        MP3 files, favorite images, your ThinkGeek shopping
        list). This watch can handle the mission for you and
```

The Insert bar lets you rapidly insert Flex containers and components in Design view by clicking or dragging. The Insert bar also lets you rapidly insert their tags in Code view.



Note: The Favorites tab found in Macromedia Dreamweaver is not supported in Flex Builder.

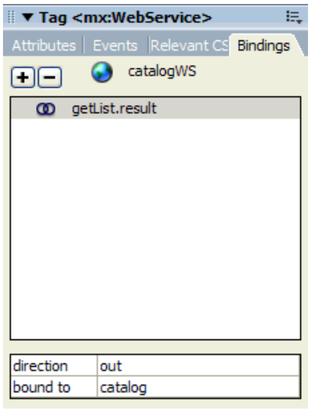
The Tag inspector lets you set the properties of containers and components in the file. You can also use it to apply effects, specify event handlers, modify CSS styles, and create data bindings.



Common	
alpha	
autoSize	
html	true
htmlText	
id	
styleName	
text	{dataModel.descript..
toolTip	
visible	
Effects	
Other	
Size	
height	
heightFlex	1
maxHeight	
maxWidth	
minHeight	
minWidth	
preferredHeight	
preferredWidth	
scaleX	
scaleY	
width	
widthFlex	1
x	
y	
Styles	

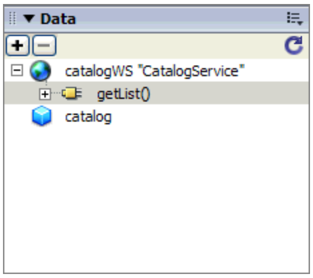
The Tag inspector replaces the traditional Property inspector in Dreamweaver. While you can still use the Property inspector for editing HTML and other document types in Flex Builder, it is disabled when you work on MXML or ActionScript files.

The Bindings panel in the Tag inspector lets you visually bind components, Flex data models, and Flex data services to other components, data models, and data services. After creating the binding, data can pass between the objects at runtime.



direction	out
bound to	catalog

The Data panel lets you inspect the data services (such as web services) and data models in the current MXML file, as well as insert or delete services in the file.



You can work in Design view, Code view, or both.

To switch between Design and Code views:

1. Open any MXML file in Flex Builder by double-clicking the file in the Files panel (Window > Files).
2. If you're in Code view and you want to work in Design view, do one of the following:
   - Click the Design button on the Document toolbar.
   - Select View > Design.
   - Press Control+' (the slanted quotation mark that usually shares the tilde (~) key).

   Note: You can't work on ActionScript files in Design view. Because ActionScript files don't render in Flash Player, they would appear blank in Design view. As a result, Flex Builder always displays ActionScript files in Code view.
3. If you're in Design view and you want to work in Code view, do one of the following:
   - Click the Code button on the Document toolbar.
   - Select View > Code.
   - Press Control+' (the slanted quotation mark that usually shares the tilde (~) key).
4. To split the view to see both Design and Code views, do one of the following:
   - Click the Split button on the Document toolbar.
   - Select View > Code and Design.
5. To place Code view on a separate monitor, open the Code inspector (Window > Code Inspector or F10) and position it on the second monitor.

For more information on the coding environment, see "Working with the code in Flex files" on page 78. For more information on the visual environment, see Chapter 4, "Building a Flex User Interface Visually," on page 97.

Starting a new application in Flex Builder

The first step in creating any new Flex application in Flex Builder is to define a Flex Builder site. A Flex Builder site is analogous to a project in other development environments: it lets you manage files and transfer them to the computer running the Flex server. A Flex Builder site also lets you visually design, preview, and debug MXML and ActionScript files without leaving Flex Builder.

You need access to a Flex server before you can define a site. The server allows you to run and debug MXML and ActionScript files without leaving Flex Builder (see "What you need to build Flex applications" on page 18). You can install the Flex server or consult your system administrator. For information on installing the server, see Installing Flex on the Macromedia website at www.macromedia.com/go/flex_install/. For information on installing the server on the same computer as Flex Builder, see "Install the Flex server on your computer" on page 22.

This section covers the following topics:

- "Defining a Flex Builder site" on page 9
- "Identifying a Flex application root folder" on page 11

Defining a Flex Builder site

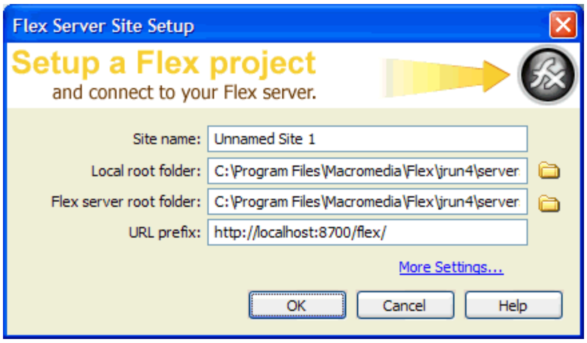
You can define a Flex Builder site quickly or with more advanced options, as follows.

To quickly define a Flex Builder site:

1. In Flex Builder, select Site > Manage Sites, click the New button on the Manage Sites dialog box, and select Flex Site from the context menu.

   Tip: You can also click Flex Site on the Start page.

   The Flex Server Site Setup dialog box appears.

   Site name: Unnamed Site 1
   Local root folder: C:\Program Files\Macromedia\Flex\jrun4\server
   Flex server root folder: C:\Program Files\Macromedia\Flex\jrun4\server
   URL prefix: http://localhost:8700/flex/

2. Enter new values or accept the default values for your site.

   Accept the default values only if you installed the Flex server on the same computer as Flex Builder using the integrated JRun/Flex install option.

   For a list of possible settings for this dialog box, see "Examples of site settings" on page 15.

   For more information, click the Help button.

To define a Flex Builder site with advanced options:

1. In Flex Builder, select Site > Manage Sites, click the New button on the Manage Sites dialog box, and select Site from the context menu.

   The Site Definition dialog box appears. If the Basic tab is selected, click the Advanced tab and select Local Info from the Category list (it should be the default).

   For possible settings for this dialog box, see "Examples of site settings" on page 15.

2. In the Local Info category, specify where you want to store your application files.

   If the Flex server is installed on the same computer as Flex Builder, specify a Flex application root folder as the Local Root Folder. For more information, see "Identifying a Flex application root folder" on page 11.

   For more information on setting the Local Info category options, click the Help button on the dialog box.

3. (Optional) In the Remote Info category, specify where you want to upload or download files from a remote server.

   Complete this category if the Flex server is running on a different computer than Flex Builder. This folder must be an application root folder on the server. For more information, see "Identifying a Flex application root folder" on page 11.

   If you are using FTP access, enter the Flex application root folder as the Host Directory value. If you are using network access, enter the Flex application root folder in the Remote Folder text box.

   For more information on setting the Remote Info category options, click the Help button on the dialog box.

4. In the Testing Server category, select MXML for JSP from the Server Model pop-up menu.

5. In the Testing Server category, specify where Flex Builder can "test" your files—that is, how and where it sends the files to be compiled at design time.

   Flex Builder sends MXML and ActionScript files to the server for compilation at design time so that you can preview Flex files, get debugging information, and connect to back-end systems.

   Specify how and where to send the files as follows:

   - In the Access pop-up menu, select the method Flex Builder should use to send the files to the server. Select Local/Network if you want to transfer the files to a local folder or to a remote folder with network access. Select FTP if you want to transfer the files to a remote folder using FTP.
   - In the Testing Server Folder text box, specify a folder where MXML and ActionScript files can be compiled by the Flex server. This folder must be a Flex application root folder as specified in steps 2 or 3 above. For more information, see "Identifying a Flex application root folder" on page 11.

6. In the URL Prefix text box, specify the URL prefix that Flex Builder should append to Flex file names.

   To get a file compiled at design time, Flex Builder sends the file to the Flex application root folder and then attempts to request it using a URL—in effect Flex Builder becomes a client. Flex Builder deduces the correct URL by combining the URL prefix you specify with the file name.

   For example, if your application's URL is www.macromedia.com/flex/start.mxml, you enter the following URL prefix in the Testing Server category:

   http://www.macromedia.com/flex/

   If Flex Builder runs on the same computer as your Flex server, you can use localhost for your domain name, as follows:

   http://localhost:8700/flex/

   Note: The default port number is 8700. If you changed it during installation, use the appropriate number in the URL prefix.

   For more information on setting the Testing Server category options, click the Help button on the dialog box.

7. Click OK to define the Flex Builder site and dismiss the Site Definition dialog box, and then click Done to dismiss the Manage Sites dialog box.

You can now start developing your application with Flex Builder.

Related topics

- "Tutorial: Setting up a development environment" on page 21
- "Specifying where dynamic pages can be processed" in Using Dreamweaver Help.

Identifying a Flex application root folder

You must specify a valid Flex application root folder when defining a Flex Builder site. Flex Builder puts files such as MXML, ActionScript, CSS, and image files in this folder in order to run and debug files at design time. If the folder is not a valid application root folder, Flex Builder will not work properly.

To identify a folder as a Flex application root folder:

- Look for a WEB-INF/flex/ subfolder with a flex-config.xml file in it.

  A Flex application root folder always has a WEB-INF/flex/ subfolder with a flex-config.xml file in it. For example, if you're using JRun 4 as the J2EE server, the application root folder is as follows:

  jrun_root/servers/*server*/*Flex_app*

  where *jrun_root* is the J2EE server's root folder, *server* is the name of a defined JRun server (such as "default"), and *Flex_app* is the application root folder. The *Flex_app* folder must contain a WEB-INF/flex/ subfolder with a flex-config.xml file in it, as follows:

  Flex_app/WEB-INF/flex/flex-config.xml

For example, if you installed the Flex server on a Windows computer using the integrated JRun4/Flex install option, the application root folder of the default application is as follows:

C:\Program Files\Macromedia\Flex\jrun4\servers\default\flex

In this example, the JRun root (*jrun_root*) is C:\Program Files\Macromedia\Flex\jrun4\, the JRun4 server (*server*) is called default, and the Flex application (*Flex_app*) is called flex. The flex folder has a WEB-INF\flex subfolder with a flex-config.xml file:

...\flex\WEB-INF\flex\flex-config.xml

For more examples, see "Examples of site settings" on page 15.

Defining a site for the Flex Sample Apps application

A Flex application called Flex Sample Apps is created when you install the Flex server using the integrated JRun4/Flex install option. This application provides a number of sample applications you can use to explore Flex and learn to work with Flex Builder. This section describes how to define a Flex Builder site to work with the Flex Sample Apps application.

Note: If you didn't use the JRun4/Flex install option to install Flex, then you must deploy the samples .war file yourself.

For information on installing Flex server, see Installing Flex on the Macromedia website at www.macromedia.com/go/flex_install/. For information on installing Flex server on the same computer as Flex Builder, see "Install the Flex server on your computer" on page 22.

This section consists of the following sections:

- "Defining the site if the Flex server is local" on page 12
- "Defining the site if the Flex server is remote" on page 14
- "Examples of site settings" on page 15

Defining the site if the Flex server is local

This section describes how to define a Flex Builder site to work with the Flex Sample Apps application installed with the Flex server. The section assumes you installed the server on the same computer as Flex Builder using the integrated JRun4/Flex install option (see "Install the Flex server on your computer" on page 22). For settings for other configurations, see "Examples of site settings" on page 15.

For information on setting up Flex Builder when the Flex server is on another computer, "Defining the site if the Flex server is remote" on page 14.

To set up Flex Builder for the samples application if Flex is running locally:

1. In Flex Builder, select Site > Manage Sites, click the New button in the Manage Sites dialog box, and select Flex Site from the context menu.

   The Flex Server Site Setup dialog box appears.

   Tip: You can also open this dialog box by clicking the Flex Site link on the Start page.

2. In the Site Name text box, enter **Flex Sample Apps** or something similar.

   The name identifies your Flex Builder site.

3. In the Local Root Folder text box, click the folder icon and select the following folder:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\

   This step tells Flex Builder where the application files are located on your hard drive.

4. In the Flex Server Root Folder text box, make sure the following path is specified:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\

   This folder must always be a Flex application root folder on the server. To determine whether a folder is an application root folder, check to see if it has a WEB-INF/flex folder containing a flex-config.xml file. An application root folder always has this folder and file. For more information, see "Identifying a Flex application root folder" on page 11.

   This step tells Flex Builder where it can "test" files—that is, where it can send MXML and ActionScript files to be compiled by the Flex server at design time. Flex Builder needs to get these files compiled so you can preview MXML files, get build and debugging information, and connect to back-end systems.

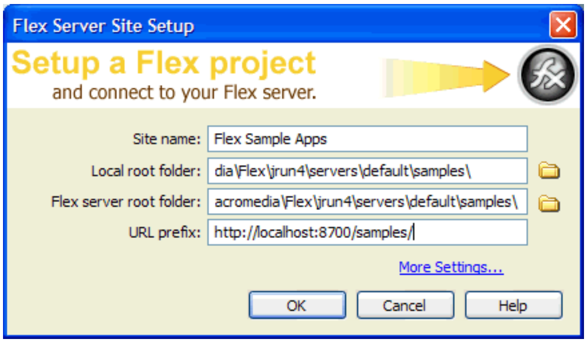
5. In the URL Prefix text box, enter the following URL:

   http://localhost:8700/samples/

   Note: The default port number is 8700. If you changed it during installation, use the appropriate number in the URL prefix.

   To get the Flex server to compile files while you work, Flex Builder uploads the file to the application root folder you specified and attempts to request it using the URL prefix.

   The Flex Server Site Setup dialog box should look as follows:

   Site name: Flex Sample Apps
   Local root folder: dia\Flex\jrun4\servers\default\samples\
   Flex server root folder: acromedia\Flex\jrun4\servers\default\samples\
   URL prefix: http://localhost:8700/samples/

6. Click OK to define the site and close the Flex Server Site Setup dialog box, and then click Done to close the Manage Sites dialog box.

The Flex Sample Apps site is defined and you can start exploring, editing, previewing, and debugging the sample apps with Flex Builder.

Caution: You should make a backup of the samples folder so you can restore files if necessary.

Defining the site if the Flex server is remote

This section describes how to define a Flex Builder site to work with the Flex Sample Apps application if the Flex server is installed and running on a different computer than Flex Builder. For more information, see Installing Flex on the Macromedia website at www.macromedia.com/go/flex_install/.

Before you start, you can review example settings for common J2EE servers. See "Examples of site settings" on page 15.

To set up Flex Builder for the samples application if Flex is running remotely:

1. In Flex Builder, select Site > Manage Sites, click the New button on the Manage Sites dialog box, and select Site from the context menu.

   The Site Definition dialog box appears. If the Basic tab is selected, click the Advanced tab and select Local Info from the Category list (it should be the default). You use the Local Info category to tell Flex Builder where the Flex application files are located on your hard drive.

2. In the Site Name text box, enter **Flex Sample Apps** or something similar.

   The name identifies your Flex Builder site.

3. In the Local Root Folder text box, click the folder icon and specify where you want to store your files on your hard drive while you work on them with Flex Builder.

4. In the Remote Info category, specify where you want to put files to the remote server.

   For this Flex Builder site, you must specify the folder called samples on the server. For example, if you installed Flex with JRun 4, the application root folder is as follows:

   jrun_root/servers/default/samples

   where *jrun_root* is the J2EE server's root folder, default is the name of a defined JRun server, and samples is a Flex application root folder.

   The remote folder for a Flex site must always be an application root folder on the server. To determine whether a folder is an application root folder, check to see if it has a WEB-INF/flex folder containing a flex-config.xml file. An application root folder always has this folder and file. The samples folder contains a WEB-INF/flex/ subfolder with a flex-config.xml file in it, as follows:

   samples/WEB-INF/flex/flex-config.xml

   For more information, see "Identifying a Flex application root folder" on page 11.

   If using FTP access, select FTP from the Access pop-up menu and specify the samples folder as the Host Directory value, as well as any other FTP settings required to access that folder.

   If using network access, select Local/Network from the Access pop-up menu and specify the network path to the samples folder in the Remote Folder text box.

   For more information on setting the Remote Info category options, click the Help button on the dialog box.

5. Click Testing Server in the Category list to display the Testing Server settings.

   You use the Testing Server options to tell Flex Builder where it can "test" files—that is, where it can send files to be compiled at design time. Flex Builder needs to get these files compiled so you can preview MXML files, get build and debugging information, and connect to back-end systems.

6. Select MXML for JSP from the Server Model pop-up menu.

   This tells Flex Builder what kind of web application you're developing.

7. In the Access pop-up menu, select the same access method you selected in the Remote Info category (FTP or Local/Network).

   After you make your selection, Flex Builder automatically populates the rest of the dialog box based on the information you entered in the Remote Info category. This is only a best guest. Make sure the following values are used:

   - **Testing Server Folder** (if using Local/Network access) is identical to the remote folder in the Remote Info category (.../samples)
   - **FTP** settings (if using FTP access) are identical to the settings in the Remote Info category
   - **URL Prefix** is an HTTP address Flex Builder can use to request a file in the testing server folder, such as http://<hostname>/samples/.

     To get a file compiled at design time, Flex Builder uploads the file to the testing server folder you specified and attempts to request it using the URL prefix.

8. Click OK to define the site and dismiss the Site Definition dialog box, and then click Done to dismiss the Manage Sites dialog box.

The Flex Sample Apps site is defined and you can start exploring, editing, previewing, and debugging the sample apps with Flex Builder.

Caution: You should make a backup of the samples folder so you can restore files if necessary.

Examples of site settings

This section lists Flex Builder site settings for the Flex Sample Apps application running on common J2EE servers. This application provides a number of sample applications you can use to explore Flex and learn to work with Flex Builder.

The Flex Sample Apps application is created when you install the Flex server using the JRun4/Flex install option. If you don't use this install option, you must deploy the samples .war file yourself.

Use the following settings in the Flex Server Site Setup dialog box. For more information, see "Defining a Flex Builder site" on page 9. You can also use these settings in the advanced Site Definition dialog box (if you access the server through a network). Enter the Flex Server Root Folder value in both the Remote Folder text box and the Testing Server Folder text box in the Site Definition dialog box.

Replace items in angle brackets with the information specific to your server configuration.

Caution: These site settings assume default installations of both the application server and the Flex server. The settings may not work for custom installations.

Integrated JRun4/Flex - Local Install	
Local Root Folder:	C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples
Flex Server Root Folder:	C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples
URL Prefix	http://localhost:8700/samples/

Non-integrated JRun4 - Local Install	
Local Root Folder:	C:\JRun4\servers\‹servername›\samples\
Flex Server Root Folder:	C:\JRun4\servers\‹servername›\samples\
URL Prefix	http://localhost:‹port›/samples/

Tomcat 5.0 - Remote Install	
Local Root Folder:	Any folder on the computer running Flex Builder
Flex Server Root Folder:	...‹installfolder›\Tomcat 5.0\webapps\samples
URL Prefix	http://‹hostname›:8080/samples/

WebSphere 5.1 - Remote Install	
Local Root Folder:	Any folder on the computer running Flex Builder
Flex Server Root Folder:	...‹installfolder›\WebSphere\AppServer\installedApps\‹nodename›\‹appname›.ear\samples.war (where ‹nodename› is the machine name—but could also be an IP or a fqdn—and ‹appname› is configurable at deploy time)
URL Prefix	http://‹hostname›:9080/samples/

WebLogic 8.1 - Remote Install	
Local Root Folder:	Any folder on the computer running Flex Builder
Flex Server Root Folder:	...‹installfolder›\bea\user_projects\‹domain›\applications\samples (assumes you exploded the war file into a folder named samples to get a samples context root)
URL Prefix	http://‹hostname›:7001/samples/

Getting help in Flex Builder

Flex Builder has a variety of resources to help you develop Flex applications. You can get context-sensitive help about an MXML tag, find out how to use a Flex Builder feature, or look up a term in the Flex ActionScript Language Reference.

This section covers the following topics:

- "Context-sensitive help" on page 17
- "Flex Builder Help" on page 17
- "Macromedia website resources" on page 18

Context-sensitive help

While working in Code view, you can get context-sensitive help about MXML components.

To get help about an MXML component:

1. In Code view, click anywhere inside the MXML component's tag.
2. Press F1.

   Flex ActionScript and MXML API Reference Help opens and displays information about the tag.

You can also access this help from the Help menu.

Flex Builder Help

The Flex Builder Help menu gives you access to thousands of pages of information while you work. The following menu items can be useful for Flex developers:

Using Flex Builder describes how to use Flex Builder to develop Flex applications.

Developing Flex Applications is a code-centric developer guide that describes how to develop Flex applications with MXML and ActionScript. The guide includes chapters on working with various controls, containers, and their properties.

Flex ActionScript Language Reference is a revised version of the ActionScript Reference Guide included in Flash MX 2004. The reference also acts as a developer guide providing a good overview of ActionScript syntax, information on how to use ActionScript when working with different types of objects, and details on the syntax and usage of the language elements.

Flex ActionScript and MXML API Reference is an API reference for Flex developers.

Using Dreamweaver provides comprehensive, task-oriented information about Dreamweaver MX 2004, the professional web authoring tool built into Flex Builder.

Macromedia website resources

The Macromedia website contains the following resources to support Flex developers:

Flex Builder support pages on the Macromedia website at www.macromedia.com/go/fb_support/ helps you get the most out of Flex Builder. The website is updated regularly with the latest information on Flex Builder, plus advice from expert users, examples, tips, updates, and information on advanced topics.

The Flex Support Center on the Macromedia website at www.macromedia.com/go/flex_support/ provides information on Flex technology.

The Flex Documentation Center on the Macromedia website at www.macromedia.com/go/flex_docs/ provides product manuals, tutorials, and articles about Flex.

Macromedia Developer Center on the Macromedia website at www.macromedia.com/go/devnet/ provides tools, tutorials, and more for all Macromedia products.

The Flex online forum on the Macromedia website at www.macromedia.com/go/flex_newsgroup.

What you need to build Flex applications

To develop and test Flex applications with Flex Builder, you need a computer running Microsoft Windows 2000 or Windows XP. For more information, see "System requirements for Flex Builder" on page 19.

You also need the Flex server (see "About the Flex server" on page 20), which requires the following software:

- An HTTP web server (see "About web servers and Flex" on page 19)
- A J2EE application server (see "About J2EE application servers and Flex" on page 19)

You can quickly set up a Flex server by using the HTTP web server and J2EE server installed with the Flex server. For added convenience, you can install and run this software on the same computer as Flex Builder. For more information, see "Install the Flex server on your computer" on page 22.

You can also configure the Flex server to work with third-party web servers and J2EE servers. For more information, see the Flex installation guide on the Macromedia website at www.macromedia.com/go/flex_install/, or consult your system administrator. Generally, it's best to match your development environment to your production environment as closely as possible. For example, if you use Tomcat in your production environment, you may want to configure your development environment around Tomcat.

A Flex best practice is to place the code implementing the functionality or business rules of the application in a business layer. Flex is designed for building the presentation layer (or user interface) of multi-tier enterprise applications. There is no inherent database support. To access databases, you must connect your presentation layer to a business layer. For more information, see "Flex data services" on page 154. The development of the business and data layers is beyond the scope of this guide.

This section covers the following topics:

- "System requirements for Flex Builder" on page 19
- "About web servers and Flex" on page 19
- "About J2EE application servers and Flex" on page 19
- "About the Flex server" on page 20

System requirements for Flex Builder

The following hardware and software is required to run Flex Builder.

- Windows 2000 or Windows XP
- An Intel Pentium III processor or equivalent, 600 MHz or faster
- At least 256 MB of available RAM (512 MB recommended)
- At least 275 MB available disk space
- A 16-bit (thousands of colors) monitor capable of 1024 x 768 pixel resolution or better (millions of colors recommended)

About web servers and Flex

To host websites containing Flex files, you need a web server, which is sometimes called an HTTP server. A web server is software that serves files in response to requests from web browsers using the HTTP protocol. Common web servers include Apache HTTP Server and Microsoft Internet Information Server (IIS).

If you select the integrated JRun4/Flex option when installing the Flex server, you don't need a separate web server. JRun4 has a web server that's installed and configured for you. For more information, see "Install the Flex server on your computer" on page 22.

If you plan on using Tomcat as your J2EE application server, you don't need to install a web server either. Apache HTTP Server is built into Tomcat. For more information, see "About J2EE application servers and Flex" on page 19.

For information on installing and configuring a web server, see the server vendor's documentation or your system administrator.

About J2EE application servers and Flex

A J2EE application server helps the Flex server process MXML and ActionScript files.

If you select the integrated JRun4/Flex option when installing the Flex server, you don't need a separate J2EE server. A J2EE server (JRun 4) is installed and configured for you. For more information, see "Install the Flex server on your computer" on page 22.

The following are the application servers supported by Flex:

- Macromedia JRun 4 with Updater 2. You can install JRun 4 with the Flex server by selecting the integrated JRun4/Flex install option. You can also download a trial version of JRun 4 from the Macromedia website at www.macromedia.com/go/jun_trial/.
- Jakarta Tomcat 4.1.29 or 5.0.18. You can download a copy of Tomcat from the Jakarta Project website at http://jakarta.apache.org/tomcat/.
- IBM WebSphere Application Server 5. You can download a trial version from the IBM website at www7b.software.ibm.com/wsdd/downloads/WASsupport.html#download.
- BEA WebLogic Server 7 or 8.1. You can download a trial version from the BEA website at http://commerce.bea.com/index.jsp.

For information on installing and configuring an application server, see the server vendor's documentation or your system administrator.

About the Flex server

To compile and run MXML and ActionScript files, you need the Flex server, which is available for Windows, Linux, and Solaris operating systems.

You can order the Developer edition of the Flex server for a nominal fee from the Macromedia website at www.macromedia.com/go/flex_trial/. The Developer edition is for noncommercial use for developing and testing Flex applications. It is not licensed for deployment. After 60 days, it will support requests from only five IP addresses (plus localhost), but you can still use it for development and testing as long as you want. The software does not expire.

For information on installing the Developer edition on the same computer as Flex Builder, see "Install the Flex server on your computer" on page 22. For information on other possible configurations, see the Flex installation guide on the Macromedia website at www.macromedia.com/go/flex_install/.

Related topics

- Appendix A, "Basic Flex Concepts," on page 147

Typographical conventions of this guide

The following typographical conventions are used in this guide:

- Menu items are shown in this format: menu name > menu item name. Items in submenus are shown in this format: menu name > submenu name > menu item name.
- `Code font` indicates HTML tag and attribute names as well as literal text used in examples.
- *`Italic code font`* indicates replaceable items (sometimes called metasymbols) in code.
- **Bold roman text** indicates text for you to enter verbatim.

CHAPTER 2
Flex Builder Tutorials

This series of four interconnected tutorials explains how you can build a simple Macromedia Flex application using Macromedia Flex Builder. The application is part of a website for the fictitious Flex Store company. The application gives visitors the ability to view a product catalog, find out more about each product, and add products to a shopping cart. The application is a simplified version of the Flex Store application installed with the Flex server.

The tutorials focus on using the development tools in Flex Builder, not on developing Flex applications. The goal is to show you how to use Flex Builder to develop a Flex application more rapidly and with less coding.

You can complete each tutorial as a stand-alone unit or as a part of a larger four-part tutorial. You don't have to complete all the tutorials or complete them in order, except for the first tutorial, which you must complete before you can start any of the others.

The first tutorial shows you how to set up a Flex development environment. The second shows you how to lay out a Flex user interface with Flex Builder. The third shows you how to use Flex Builder to create custom components, the building blocks of Flex applications. The fourth shows you how to visually bind components to data with Flex Builder.

This chapter includes the following tutorials:

- "Tutorial: Setting up a development environment" on page 21
- "Tutorial: Creating a layout with Flex Builder" on page 26
- "Tutorial: Building custom components with Flex Builder" on page 35
- "Tutorial: Binding components to data with Flex Builder" on page 59

Tutorial: Setting up a development environment

This tutorial shows you how to install the Flex server on your computer, copy sample files to a folder, and define a Flex Builder site. You must complete these tasks to complete the other tutorials in this chapter.

In this tutorial, you'll accomplish the following tasks:

- "Install the Flex server on your computer" on page 22
- "Copy the tutorial folders to the Flex samples folder" on page 24
- "Define a Flex Builder site" on page 25

Install the Flex server on your computer

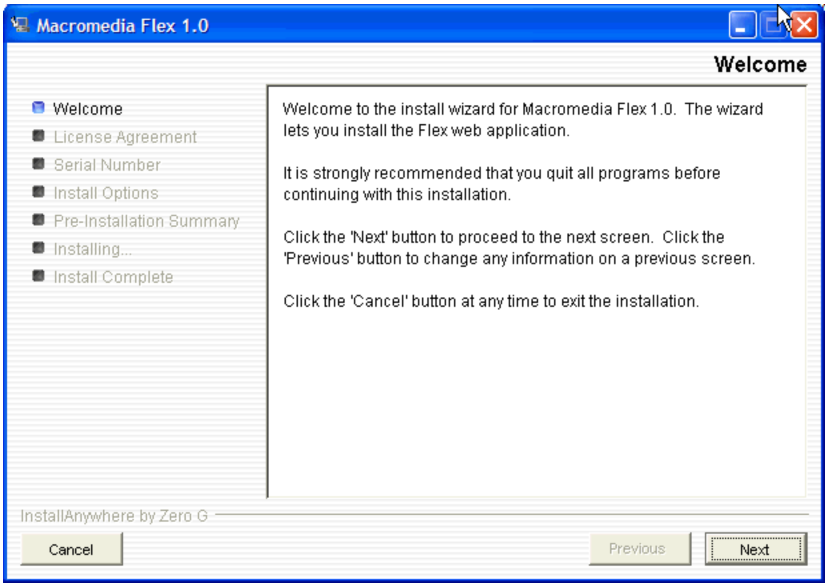
This part of the tutorial describes how to set up the Developer edition of Flex on the same computer as Flex Builder. To compile and run MXML and ActionScript files, you need access to a Flex server.

The setup described in this part of the tutorial is only one of many possible configurations. You can also install Flex on a remote server running a different J2EE server and access it through a network or using FTP. If you don't select the integrated JRun4/Flex install option, make sure you deploy the samples .war file. For more information, see the Flex installation guide on the Macromedia website at www.macromedia.com/go/flex_install/.

The Developer edition is for noncommercial use for developing and testing Flex applications. It is not licensed for deployment. After 60 days, it will support requests from only five IP addresses (plus localhost), but you can still use it for development and testing as long as you want. The software does not expire.

1. Copy the Windows version of the Flex installer from the Macromedia Flex CD.
2. In Windows, double-click the Flex installer file, flex-10-win.exe.

   The install wizard appears:



Welcome to the install wizard for Macromedia Flex 1.0. The wizard lets you install the Flex web application.

It is strongly recommended that you quit all programs before continuing with this installation.

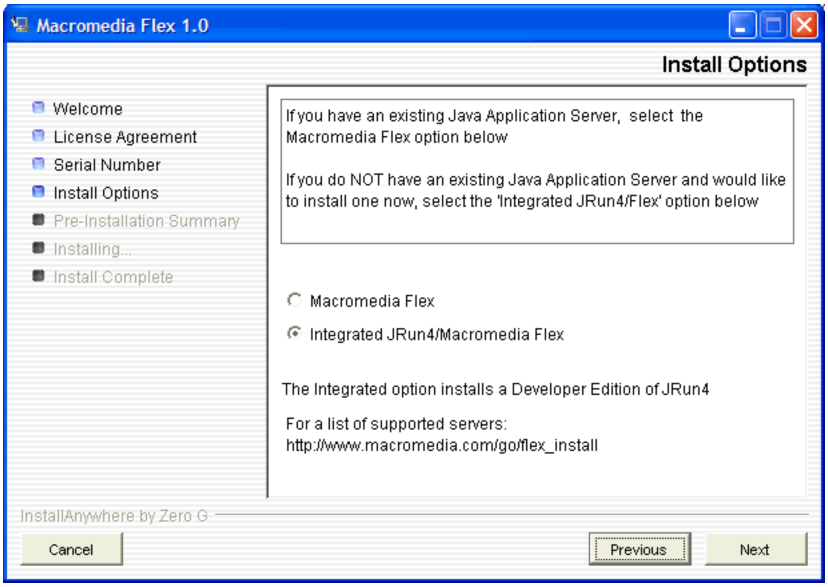
Click the 'Next' button to proceed to the next screen. Click the 'Previous' button to change any information on a previous screen.

Click the 'Cancel' button at any time to exit the installation.

3. Follow the onscreen instructions.
4. When prompted for a serial number, leave the text box blank.

   You don't need to enter a serial number for the Developer edition. The Flex server works for 60 days as a trial edition; after that, it becomes a Developer edition.

5. On the Install Options dialog box, select the Integrated JRun4/Macromedia Flex option.



If you have an existing Java Application Server, select the Macromedia Flex option below

If you do NOT have an existing Java Application Server and would like to install one now, select the 'Integrated JRun4/Flex' option below

( ) Macromedia Flex
(•) Integrated JRun4/Macromedia Flex

The Integrated option installs a Developer Edition of JRun4

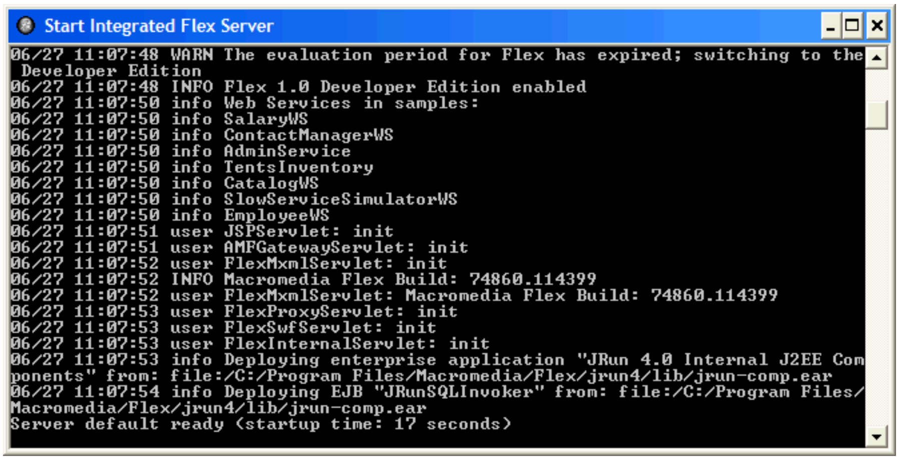
For a list of supported servers:
http://www.macromedia.com/go/flex_install

The dialog box gives you other installation options. This tutorial assumes that you selected the Integrated JRun4/Macromedia Flex option.

6. After you finish installing Flex 1.0, install any Flex updaters available on the Macromedia website at www.macromedia.com/go/flex_updaters.

7. Start the server by selecting Start > All Programs > Macromedia > Macromedia Flex > Start Integrated Flex Server.

   The server displays startup information.



```
06/27 11:07:48 WARN The evaluation period for Flex has expired; switching to the
 Developer Edition
06/27 11:07:48 INFO Flex 1.0 Developer Edition enabled
06/27 11:07:50 info Web Services in samples:
06/27 11:07:50 info SalaryWS
06/27 11:07:50 info ContactManagerWS
06/27 11:07:50 info AdminService
06/27 11:07:50 info TentsInventory
06/27 11:07:50 info CatalogWS
06/27 11:07:50 info SlowServiceSimulatorWS
06/27 11:07:50 info EmployeeWS
06/27 11:07:51 user JSPServlet: init
06/27 11:07:51 user AMFGatewayServlet: init
06/27 11:07:52 user FlexMxmlServlet: init
06/27 11:07:52 INFO Macromedia Flex Build: 74860.114399
06/27 11:07:52 user FlexMxmlServlet: Macromedia Flex Build: 74860.114399
06/27 11:07:53 user FlexProxyServlet: init
06/27 11:07:53 user FlexSwfServlet: init
06/27 11:07:53 user FlexInternalServlet: init
06/27 11:07:53 info Deploying enterprise application "JRun 4.0 Internal J2EE Com
ponents" from: file:/C:/Program Files/Macromedia/Flex/jrun4/lib/jrun-comp.ear
06/27 11:07:54 info Deploying EJB "JRunSQLInvoker" from: file:/C:/Program Files/
Macromedia/Flex/jrun4/lib/jrun-comp.ear
Server default ready (startup time: 17 seconds)
```

Caution: Do not close the Start Integrated Flex Server dialog box once the server is running. Closing it shuts down the Flex server.

For more information, see the Flex installation documentation on the Macromedia website at www.macromedia.com/go/flex_install/.

You can verify that the Flex server is running normally as follows:

1. Open the following URL in your browser:

   http://localhost:8700/samples

   The Flex Sample Apps page appears.
2. Select a sample application to run it.

If the application doesn't run, try the following troubleshooting suggestions:

Problem	Possible solution
Port conflicts	If you created a new server for the Flex application, make sure the ports used by that server don't conflict with other servers running on your web application server.
Application server	Make sure the integrated JRun4/Flex server is running. In Windows, select Start > All Programs > Macromedia > Macromedia Flex > Start Integrated Flex Server.
Flash Player	Make sure you're running the latest Macromedia Flash Player. Flash Player is installed when you install Flex Builder.

Copy the tutorial folders to the Flex samples folder

In this part of the tutorial, you copy the Flex Builder tutorial folders to the folder called samples on the Flex server. The samples folder is the Flex application root folder of the Flex Sample Apps application.

Caution: You must copy the tutorial folders to the folder called samples or the tutorials will not work properly.

1. Locate the Tutorial folder in the following folder on your hard disk.

   C:\Program Files\Macromedia\Flex Builder\Tutorial\

   If you didn't install Flex Builder to the default location, locate the Tutorial folder on your hard disk.
2. Select the three folders (fbBindings, fbComponents, and fbLayout) in the Tutorial folder and copy them to the following Flex application root folder:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\

   Note: If you installed Flex on a different computer, you must copy the files to the samples folder on the remote computer. For more information, see "Examples of site settings" on page 15.
3. In the Flex application root folder, locate the folder named products in the \samples\flexstore\assets\ folder and copy it to the \samples\fbBindings\assets\ folder.

   The products folder contains the product images used in the tutorial. If you're working with the integrated JRun4/Flex server locally, the path of the original products folder is as follows:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\flexstore\assets\products

   After copying the folder, it should also appear at the following location:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\fbBindings\assets\products

Define a Flex Builder site

In this part of the tutorial, you define a Flex Builder site for the tutorials. A Flex Builder site lets you visually design, preview, and debug MXML and ActionScript files without leaving Flex Builder.

After you define the Flex Builder site, you can start working on the tutorials.

This section assumes you installed the Flex server on the same computer as Flex Builder using the integrated JRun/Flex install option. If you installed the Flex server on a different computer, you must use a procedure that's different from the one below to define the site. For instructions, see "Defining the site if the Flex server is remote" on page 14.

1. In Flex Builder, select Site > Manage Sites, click the New button in the Manage Sites dialog box, and select Flex Site from the context menu.

   The Flex Server Site Setup dialog box appears.

   Tip: You can also open this dialog box by clicking the Flex Site link on the Start page.

2. In the Site Name text box, enter **Flex Builder Samples**.

   The name identifies your site. The samples folder contains not only the Flex Builder tutorials, but other sample applications as well.

3. In the Local Root Folder text box, click the folder icon and select the following folder:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\

   It should contain the Flex Builder tutorial folders (see "Copy the tutorial folders to the Flex samples folder" on page 24), among other folders.

   This step tells Flex Builder where your files are located on your hard disk.

4. In the Flex Server Root Folder text box, make sure the following path is specified:

   C:\Program Files\Macromedia\Flex\jrun4\servers\default\samples\

   This folder must always be a Flex application root folder on the server. You can tell if a folder is an application root folder by checking to see if it has a WEB-INF/flex folder containing a flex-config.xml file. An application root folder always has this folder and file. For more information, see "Identifying a Flex application root folder" on page 11.

   This step tells Flex Builder where it can "test" files—that is, where it can send MXML and ActionScript files to be compiled by the Flex server at design time. Flex Builder needs to get these files compiled so you can preview MXML files, get build and debugging information, and connect to back-end systems.

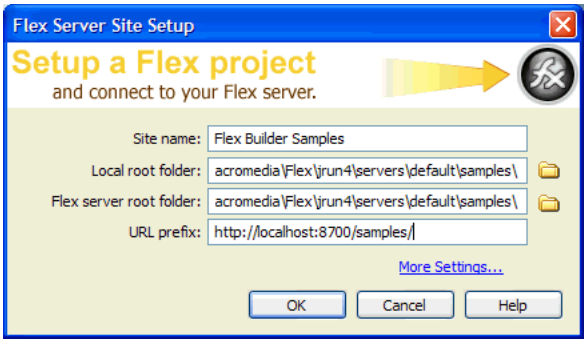
5. In the URL Prefix text box, enter the following URL:

   http://localhost:8700/samples/

   Caution: Flex Builder may auto-populate this text box with http://localhost:8700/flex/. Make sure you change the value to http://localhost:8700/samples/.

   To get the Flex server to compile files while you work, Flex Builder uploads the file to the application root folder you specified and attempts to request it using the URL prefix.

   The Flex Server Site Setup dialog box should look like this:



Site name: Flex Builder Samples
Local root folder: acromedia\Flex\jrun4\servers\default\samples\
Flex server root folder: acromedia\Flex\jrun4\servers\default\samples\
URL prefix: http://localhost:8700/samples/

6. Click OK to define the site and close the dialog box, and then click Done to close the Manage Sites dialog box.

The Flex Builder site is defined and the tutorials are set up. To start working on a tutorial, see any of the following:

- "Tutorial: Creating a layout with Flex Builder" on page 26
- "Tutorial: Building custom components with Flex Builder" on page 35
- "Tutorial: Binding components to data with Flex Builder" on page 59

Tutorial: Creating a layout with Flex Builder

This tutorial shows you how to use Flex Builder to quickly lay out a Flex user interface.

You can complete this tutorial as a stand-alone unit or as the first part of a multipart tutorial. In either case, you must complete the setup tutorial before you begin. For instructions, see "Tutorial: Setting up a development environment" on page 21.

In this tutorial, you'll accomplish the following tasks:

- "Review the approved user interface mock-ups" on page 27
- "Create an MXML file" on page 28
- "Import your CSS styles" on page 29
- "Position the page title" on page 29
- "Position the catalog component" on page 31
- "Position the product detail and shopping cart components" on page 32
- "Add view buttons to the product catalog" on page 33

Review the approved user interface mock-ups

During the planning and approval stages of the project, members of your team produced final mock-ups of the user interface for the Flex Store. You will use these mock-ups to guide you when you create the MXML layout.

The final mock-ups for the Flex Store application are stored in your fbLayout folder:

/fbLayout/mockups

You can open a mock-up file in your favorite image editor from the Files panel in Flex Builder. This feature is useful if you want to quickly change or touch up a mock-up.

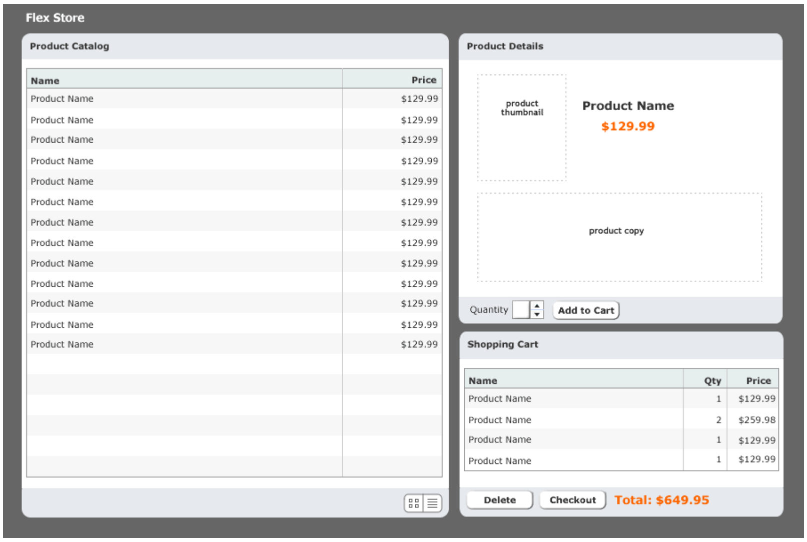
1. In Flex Builder, make sure the Flex Builder Samples site is selected in the Files panel.

   If the Files panel is closed, select Window > Files to open it.

2. In the Files panel, locate the uiGrid.png image and double-click it.

   Flex Builder opens the image in your default image editor. For more information on setting a default image editor, see “Setting external image editor preferences” in Using Dreamweaver Help.

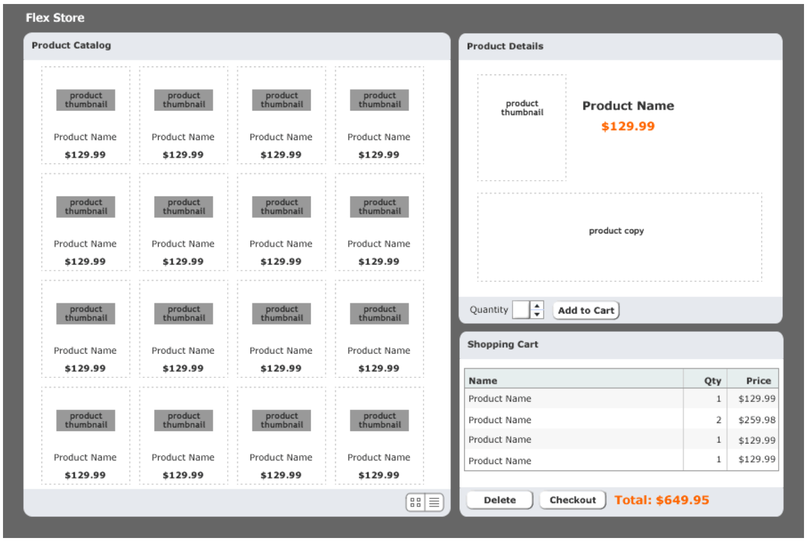
The first mock-up file, uiGrid.png, shows the general layout of the Flex Store user interface:



Name	Price
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99

Name	Qty	Price
Product Name	1	$129.99
Product Name	2	$259.98
Product Name	1	$129.99
Product Name	1	$129.99

The user will also have the option of viewing thumbnails in the product catalog on the left side of the layout. The second mock-up file, uiThumbs.png, shows this thumbnail view:



Name	Qty	Price
Product Name	1	$129.99
Product Name	2	$259.98
Product Name	1	$129.99
Product Name	1	$129.99

The layout contains the following custom components:

- Two interchangeable product catalog components on the left side, which give the user two views of the catalog
- A product detail component on the upper right side
- A shopping cart component on the lower right side

Another tutorial shows you how to build these components (see "Tutorial: Building custom components with Flex Builder" on page 35). In this tutorial you create the basic layout of the application.

Create an MXML file

In this part of the tutorial, you take the first step in laying out the application by creating an MXML file. MXML is a XML-based language for building Flex applications. For more information, see "About MXML files" on page 148.

1. In Flex Builder, make sure the Flex Builder Samples site is selected in the Files panel.
2. Select File > New.

   The New Document dialog box appears.
3. Select Flex Development in the left pane and MXML Application in the right pane, and then click Create.

   The dialog box closes and a blank MXML file appears.

4. Set the following property in the Attributes panel of the Tag inspector (Window > Tag Inspector):

   - `Styles > verticalGap:` **0**

   Note: The angle bracket means the `verticalGap` property is located in the Styles category of the Attributes tab. This convention is used throughout the tutorials.

5. Save the file in the fbLayout folder by selecting File > Save, double-clicking the fbLayout folder, and naming the file as follows:

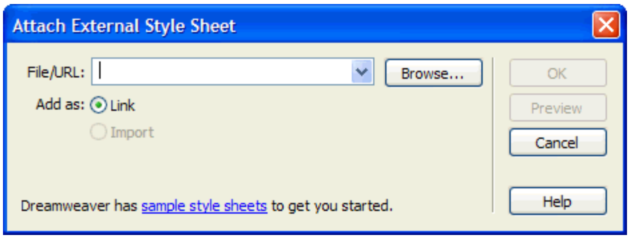
   flexstore.mxml

Import your CSS styles

In this part of the tutorial, you import CSS styles into the flexstore.mxml file. You want to import styles defined in an external CSS file to make sure your site has a consistent look and feel. CSS styles also give you more flexibility when you want to change the design of your site.

1. Make sure the flexstore.mxml file is open in Flex Builder.
2. Open the CSS Styles panel (Window > CSS Styles) and click the Attach Style Sheet button at the lower edge of the panel.

   The Attach External Style Sheet dialog box appears.



Attach External Style Sheet

File/URL:

Add as: (● Link) ○ Import

Dreamweaver has sample style sheets to get you started.

3. In the Attach External Style Sheet dialog box, click Browse to select the external CSS style sheet called flexstore.css located in the fbLayout folder.

   After you click OK twice to close the dialog boxes, the name of the external CSS style sheet appears in the CSS Styles panel. You can click the Plus (+) icon to display a tree view of all the styles in the style sheet and their properties.
4. Save your work.

Position the page title

After importing the CSS styles, the next task is to create a basic layout for the user interface. In this part of the tutorial, you position the page title (Flex Store) in the upper left corner of the layout, as indicated in the mock-up of the user interface (see "Review the approved user interface mock-ups" on page 27).

You decide to use a VBox (for vertical box) container to position the page title at the top of the file. The rest of the user interface will be positioned below the page title.

1. In Design view, click the Expanded button on the Document toolbar and then click anywhere inside the Application container.

   Expanded mode adds borders and padding to controls and containers in Design view to help you lay out your application. (The borders and padding appear only at design time.) You can click the Standard button at any time to get a better representation of how your project will look after it's compiled.

2. In the Containers category of the Insert bar, click the VBox button.

   Flex Builder inserts a VBox container in the file.

3. With the insertion point still blinking in the VBox container, set the following property in the Attributes panel:

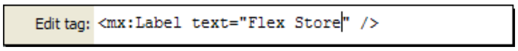
   - `Size > widthFlex:` **0**

   Tip: To verify that the component is selected, check that <mx:VBox> and its associated icon appear at the top of the Attributes panel.

4. Insert the page title by clicking anywhere inside the VBox container and clicking the Label button in the Controls category of the Insert bar.

   Flex Builder inserts a Label control in the file and selects it.

5. Modify the Label text by double-clicking the Label control to open the Quick Tag Editor, and then changing the `text` property value to `"Flex Store"` as follows and pressing Enter.

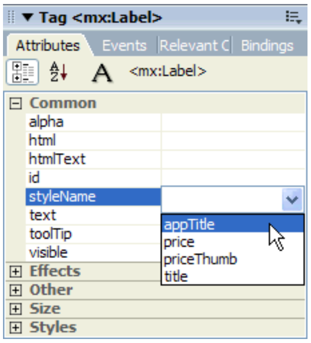


Edit tag: `<mx:Label text="Flex Store" />`

6. With the Label control still selected, set the following property in the Attributes panel:

   - `Common > styleName:` **appTitle**

   You can select the style from the pop-up menu that appears when you click the field for the `styleName` property.



   You can see the effect of applying the style in Design view.

7. Save your work.

Position the catalog component

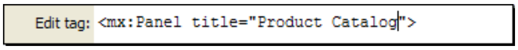
In this part of the tutorial, you position the product catalog on the left side of the user interface as shown in the mock-ups (see "Review the approved user interface mock-ups" on page 27). To meet this design requirement, you decide to use an HBox (for horizontal box) container to position the two halves of your user interface side by side.

1. In Code view (View > Code), place the insertion point at the lower edge of the VBox container by clicking immediately before the closing `</mx:VBox>` tag.
2. In the Containers category of the Insert bar, click the HBox button.

   Flex Builder inserts an HBox container in the file.
3. Click anywhere in the `<mx:HBox>` tag, click the Refresh button on the Attributes panel, and set the following properties in the panel:
   - `Styles > horizontalGap:` **4**
   - `Size > height:` **548**
   - `Size > width:` **860**

   Note: When you set the first size property, a message box appears warning you that the container clips any content that extends beyond the specified size. Select the Don't Show Me This Message Again option, and click OK.
4. Switch to Design view (View > Design), click anywhere in the HBox container, and click the Panel button in the Containers category of the Insert bar.

   Flex Builder inserts a Panel container in the file. You want to use a Panel container for the product catalog.
5. Specify the panel's title by double-clicking the Panel container to open the Quick Tag Editor, and then entering the property `title="Product Catalog"` as follows.


Edit tag: `<mx:Panel title="Product Catalog">`

6. With the Quick Tag Editor still open, enter the following properties and then Press Enter:

   `id="main" height="544" width="478"`

   Tip: Use code hints to work more rapidly.
7. Insert a ViewStack container inside the Panel container by clicking anywhere inside the Panel container, clicking the ViewStack button on the Insert bar, and entering **bodyStack** in the ID text box in the dialog box that appears.

   After you click OK, Flex Builder inserts a ViewStack container in the Panel container. You want to use this container to place two views of the product catalog—a grid view and a thumbnail view—in the same space.
8. With the ViewStack container still selected, set the following property in the Attributes panel:
   - `Effects > changeEffect:` **Fade**

   You can select the Fade value from the pop-up menu. The Fade effect will make the ViewStack container change from transparent to opaque in 500 milliseconds (the default effect duration).
9. Save your work.

Position the product detail and shopping cart components

In this part of the tutorial, you position the product detail and shopping cart components on the right side of the user interface as shown in the mock-ups (see "Review the approved user interface mock-ups" on page 27). You decide to use the existing HBox container to meet this layout requirement.

The mock-up also shows the product detail and shopping cart components stacked on top of each other. To meet this layout requirement, you decide to use a VBox container.

1. In Design view, click inside the HBox container to the right of the Panel container.

   Be careful not to click inside the Panel container; the vertical insertion bar should appear beside the Panel container.

2. In the Containers category of the Insert bar, click the VBox button.

   Flex Builder inserts a VBox container in the file. You want to use the VBox container to stack the product detail and shopping cart components on top of each other.

3. With the insertion point still blinking in the VBox container, set the following property in the Attributes panel:

   - `Size > widthFlex`: **1**

   You want the VBox container to scale based on its content.

4. Click anywhere in the new VBox container, click the Canvas button in the Containers category of the Insert bar, and accept the default width and height values in the dialog box that appears.

   Flex Builder inserts a Canvas container in the VBox. You want to insert the product detail component in this container.

5. With the Canvas container still selected, set the following properties in the Attributes panel:

   - `Common > id`: **topCanvas**
   - `Size > height`: **326**
   - `Size > width`: **364**
   - `Size > widthFlex`: **1**
   - `Other > vScrollPolicy`: **off**

   Tip: If you prefer, you can list all the properties alphabetically by clicking the Show List View button on the Attributes panel.

6. Insert another Canvas container by switching to Code view, clicking immediately after the closing `</mx:Canvas>` tag, and clicking the Canvas button on the Insert bar.

   After you accept the default width and height values in the dialog box that appears, Flex Builder inserts another Canvas container in the VBox. You want to insert the shopping cart component in this container.

7. Click anywhere in the new `<mx:Canvas>` tag, click the Refresh button on the Attributes tab, and set the following properties in the Attributes tab:
   - `Common > id`: **bottomCanvas**
   - `Size > height`: **208**
   - `Size > width`: **364**
   - `Size > widthFlex`: **1**
   - `Other > vScrollPolicy`: **off**
8. Switch back to Design view to inspect your layout.

   Tip: Press F4 to hide the workspace and maximize the Document window. Press F4 again to restore the workspace.
9. Save your work.

Add view buttons to the product catalog

The product catalog area of the layout gives the user two views of the products—a grid view and a thumbnail view. According to the user interface mock-ups, the user should be able to switch views by clicking view buttons at the lower edge of the catalog. For information about the user interface mock-ups, see "Review the approved user interface mock-ups" on page 27.

After studying the mock-up, you decide to use the following containers to lay out the buttons:

- A ControlBar container in the left Panel container to create a footer
- An HBox container in the ControlBar container to position the buttons horizontally

You use the following steps to lay out and insert the buttons with Flex Builder.

1. Make sure the flexstore.mxml file is open in Flex Builder.
2. Insert a ControlBar container by clicking inside the lower edge of the Panel container (without clicking inside the ViewStack container), and then clicking the ControlBar button in the Containers category of the Insert bar.

   Flex Builder inserts an empty ControlBar container and automatically positions it at the lower edge of the Panel container.
3. With the insertion point still blinking in the ControlBar container, specify the values for the following properties in the Attributes panel:
   - `Size > height`: **45**
   - `Styles > horizontalAlign`: **right**
4. Insert an HBox container by clicking anywhere inside the ControlBar container, and clicking the HBox button on the Insert bar.
5. With the insertion point still blinking in the HBox container, set the value for the following property in the Attributes panel:
   - `Styles > horizontalGap`: **0**

6. Insert the thumbnail view button by clicking inside the HBox container, clicking the Image button in the Controls category of the Insert bar, and selecting the following image file:

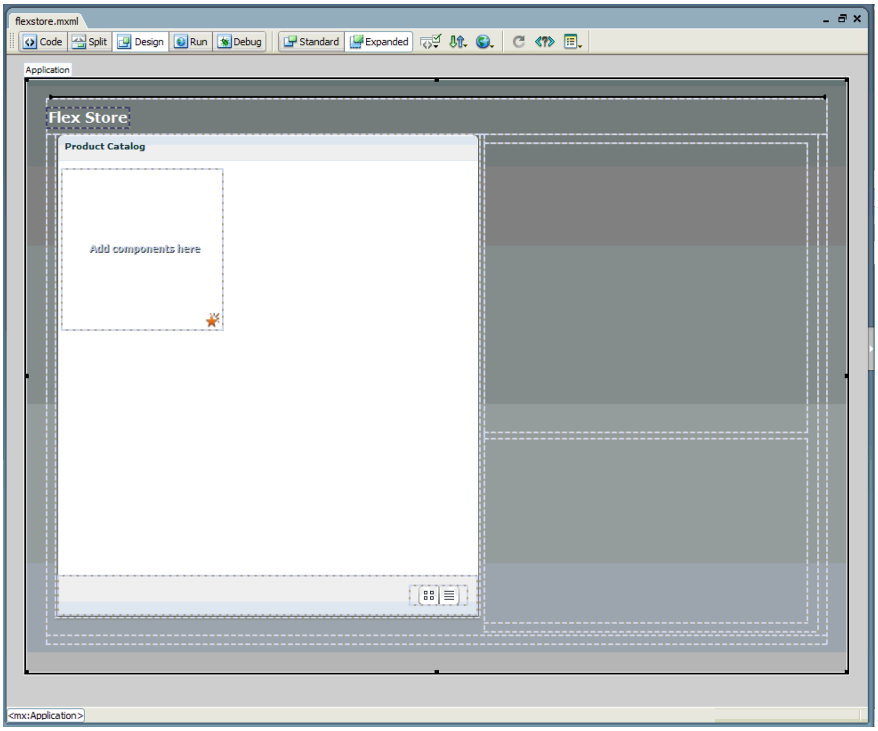
   /fbLayout/assets/images/thumb_off.png

7. Insert the grid view button by clicking inside the HBox container on the right side of the thumbnail view button (without selecting the button), clicking the Image button on the Insert bar, and selecting the following image file:

   /fbLayout/assets/images/list_off.png

8. Save your work.

In Design view, the completed layout should look similar to the following figure if you select the Application container:



This completes the layout tutorial. If you like, you can continue building the Flex Store by completing the components tutorial, which shows you how to build custom components and insert them in your layout.

Tutorial: Building custom components with Flex Builder

In this tutorial, you learn how to use Flex Builder to create custom components, the building blocks of Flex applications. The Flex Store application requires the following custom components:

- Two catalog components that give the user different views of the product catalog
- A product detail component that gives the user more detail on a product the user clicks in the catalog
- A shopping cart component that lists the products the user wants to purchase

You can complete this tutorial as a stand-alone unit or as the second part of a multipart tutorial. In either case, you must complete the setup tutorial before you begin. For instructions, see "Tutorial: Setting up a development environment" on page 21.

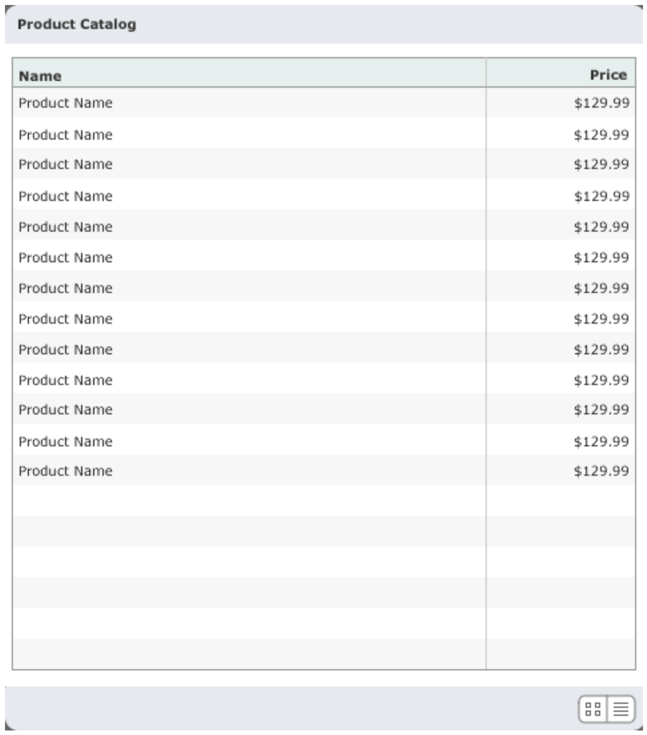
The tutorial includes a pre-built set of files so you can complete the tutorial without completing the layout tutorial first. If you completed the layout tutorial, you can overwrite the files in the fbComponents folder with your files in the fbLayout folder.

In this tutorial, you will accomplish the following tasks:

- "Build the grid view component" on page 36
- "Build the thumbnail view component" on page 39
- "Build the product detail component" on page 41
- "Build the shopping cart component" on page 48
- "Insert the view components in the Flex Store layout" on page 52
- "Insert the detail and cart components in the layout" on page 54
- "Activate the catalog view buttons" on page 55

Build the grid view component

According to the following mock-up, the grid view component should display the Flex Store product catalog as a two-column grid of names and prices. The user can select the grid view by clicking a button at the lower edge of the catalog.



Product Catalog

Name	Price
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99
Product Name	$129.99

The grid view component does not contain any product data in this tutorial. Another tutorial describes how to add data (see "Tutorial: Binding components to data with Flex Builder" on page 59).

1. In Flex Builder, make sure the Flex Builder Samples site is selected in the Files panel.
2. Select File > New.

   The New Document dialog box appears.
3. Select Flex Development in the left pane and MXML Component:Vertical in the right pane, and then click Create.

   The dialog box closes and a component file with a VBox container appears. By default, Flex Builder assigns the value of 400 to the `height` and `width` properties. You don't want to use these values.

4. With the insertion point still blinking in the VBox container, specify the values for the following properties in the Attributes panel:
   - `Size > height`: Clear the value
   - `Size > width`: Clear the value
   - `Styles > verticalGap`: **0**

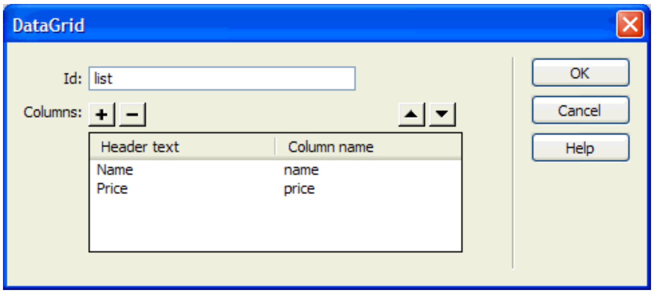
   Note: The "Size › height" expression means the height property is located in the Size category of the Attributes tab. This convention is used throughout the tutorials.

5. Insert a DataGrid component to display the product catalog by clicking anywhere in the VBox container, and clicking the DataGrid button in the Controls category of the Insert bar.

   The DataGrid dialog box appears.

6. Set the dialog box options as follows:
   - `ID`: **list**
   - `Header Text (first row)`: **Name**
   - `Header Text (second row)`: **Price**
   - `Column Name (first row)`: **name**
   - `Column Name (second row)`: **price**

   The DataGrid dialog box should look similar to the following figure:



Id: list

Columns:

Header text	Column name
Name	name
Price	price

   Caution: Make sure the column name values exactly match "name" and "price." Column names must exactly match the name of the data fields that will be assigned to the DataGrid in the bindings tutorial.

7. Click OK.

   Flex Builder inserts a DataGrid component in your component file.

8. Click the DataGrid component and set the following properties in the Attributes tab:
   - `Size > heightFlex`: **1**
   - `Size > widthFlex`: **1**

9. Switch to Code view (View > Code) and set column properties of the DataGrid by locating the two `<mx:DataGridColumn>` tags, and adding the following properties (shown in bold type):

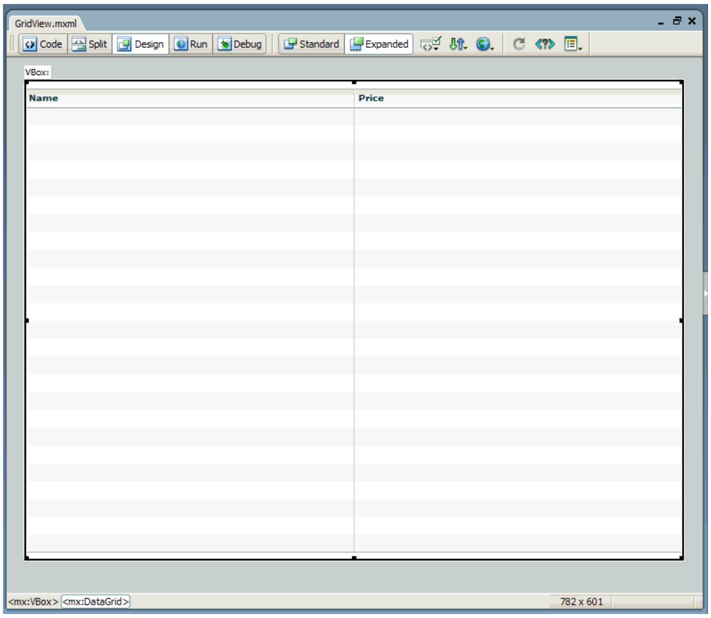
```
<mx:DataGridColumn headerText="Name" columnName="name" width="300"/>
<mx:DataGridColumn headerText="Price" columnName="price" textAlign="right"
  marginRight="4" />
```

   You can use Code hints to quickly set these properties. For example, to set the `width` property, click in the tag before the closing angle bracket, press the Spacebar to display the code hints, type the letter **w** to quickly select the property, and press Enter. Type **300** to set the value.

10. Save the component file in the fbComponents folder by selecting File > Save, double-clicking the fbComponents folder, and naming the component file as follows:

    GridView.mxml

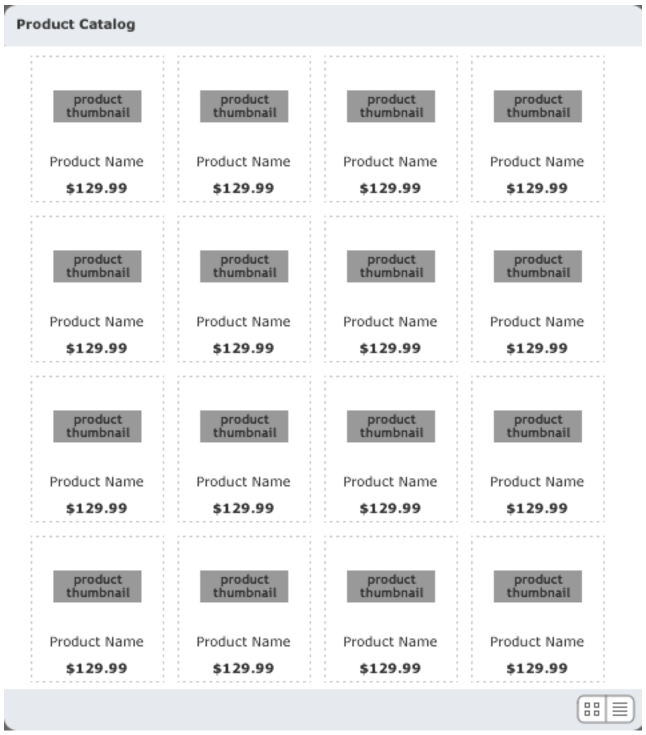
In Design View, the completed component should look similar to the following figure if you select the VBox container:



Note: The component will scale to fit in the space allotted to it in the flexstore.mxml file.

Build the thumbnail view component

According to the following mock-up, the thumbnail view component should display the Flex Store product catalog as a set of thumbnails with names and prices. The user can select the thumbnail view by clicking a button at the lower edge of the catalog.



The thumbnail view component will not contain any product data in this tutorial. Another tutorial describes how to add data (see "Tutorial: Binding components to data with Flex Builder" on page 59).

1. In Flex Builder, select File > New.

   The New Document dialog box appears.

2. Select Flex Development in the left pane and MXML Component:Vertical in the right pane, and then click Create.

   The dialog box closes and a component file with a VBox container appears.

3. With the insertion point still blinking in the VBox container, specify the values for the following properties in the Attributes panel:

   - `Size > height`: Clear the value.
   - `Size > width`: Clear the value.
   - `Styles > verticalGap`: **0**

4. Insert a Tile container to lay out the products by clicking anywhere in the VBox container, and clicking the Tile button in the Containers category of the Insert bar.

   Flex Builder inserts a Tile container in the component file.

5. With the insertion point still blinking in the Tile container, specify the values for the following properties in the Attributes panel:

   - `Common > id`: **tileView**
   - `Size > y`: **12**

6. Save the component file in the fbComponents folder by selecting File > Save, and naming the component file as follows:

   ThumbnailView.mxml

7. Insert placeholder images for the products by clicking anywhere inside the Tile container, clicking the Image button in the Controls category of the Insert bar, and selecting the following image in the mockups folder:

   fbComponents/mockups/tnProduct.png

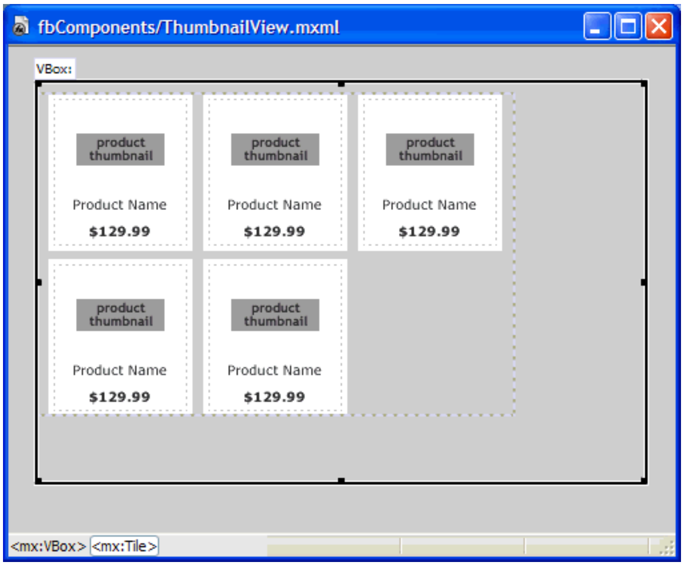
   You want to insert five or more placeholder images to represent the products. In the tutorial on data bindings (see "Tutorial: Binding components to data with Flex Builder" on page 59), you replace these placeholder images with a Repeater component and a custom component that retrieves product data and displays each product's thumbnail image, name, and price.

8. Click inside the right edge of the Tile container (without selecting the thumbnail image), and repeat step 7 four more times to insert more thumbnail images in the Tile container.

   If you're unable to click inside the edge, click the Expanded button on the Document toolbar to add extra space around the container.

9. Save your work.

In Design view, the completed component should look similar to the following figure if you select the VBox container:

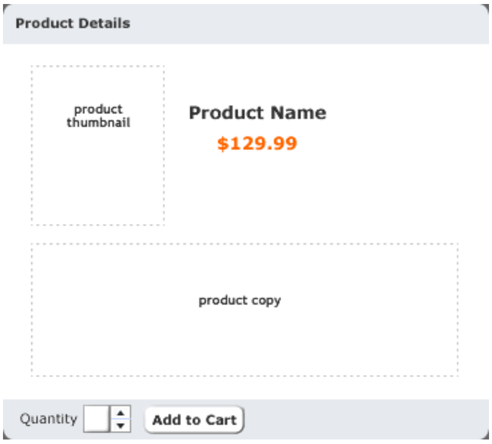


Note: The component will scale to fit in the space allotted to it in the flexstore.mxml file.

Build the product detail component

According to the component's functional specification, the product detail component should display product details when a user clicks one of the products in the product catalog. The user can add the product to the shopping cart by specifying a quantity and clicking the Add to Cart button at the lower edge of the panel.

The following is a mock-up of the component:



The product detail component will not contain any data in this tutorial. Another tutorial describes how to add data (see "Tutorial: Binding components to data with Flex Builder" on page 59).

In this part of the tutorial, you complete the following tasks:

- "Lay out the product detail component" on page 42
- "Add the product details" on page 44
- "Finish the footer of the product detail component" on page 46

Lay out the product detail component

After studying the component mock-up, you decide to use the following Flex containers to lay out the component:

- A Panel container to create the header and position the component's child containers vertically
- An HBox container within the Panel container to position the product thumbnail and the product information side by side
- A VBox container within the HBox to position the product name on top of the price
- A ControlBar container to create the footer

You start by creating a new component file based on a Panel container.

1. In Flex Builder, press Control+N.

   The New Document dialog box appears.

2. Select Flex Development in the left pane and double-click MXML Component:Panel in the right pane.

   The dialog box closes and a component file with a Panel container appears.

3. With the insertion point still blinking in the Panel container, specify the values for the following properties in the Attributes panel:

   - `Common > title`: **Product Details**
   - `Styles > marginBottom`: **8**
   - `Styles > marginLeft`: **8**
   - `Styles > marginRight`: **8**
   - `Styles > marginTop`: **8**

4. Insert an HBox container inside the Panel container by clicking anywhere inside the Panel container and clicking the HBox button in the Containers category of the Insert bar.

   Flex Builder inserts an empty HBox container in the component.

5. With the insertion point still blinking in the HBox container, specify the value for the following property in the Attributes panel:

   - `Styles > verticalAlign`: **middle**

6. Insert a Canvas container to position the product thumbnail by clicking anywhere inside the HBox container, clicking the Canvas button on the Insert bar, clearing the default values in the dialog box that appears, and clicking OK.

   Flex Builder inserts an empty Canvas container in the component.

7. With the Canvas container still selected, specify the values for the following properties in the Attributes panel:

   - `Size > height`: **140**
   - `Size > width`: **150**
   - `Common > clipContent`: **false**

8. Insert a VBox container for the product name and price by clicking inside the right edge of the HBox container (without clicking inside the Canvas container), and clicking the VBox button on the Insert bar.

   Flex Builder inserts an empty VBox container in the component.

9. Insert a ControlBar container by clicking inside the Panel container outside and below the HBox container, and clicking the ControlBar button on the Insert bar.

   Flex Builder inserts an empty ControlBar container and automatically positions it at the lower edge of the Panel container.

10. With the insertion point still blinking in the ControlBar container, specify the value for the following property in the Attributes panel:

    - `Common > id`: **ctrl**

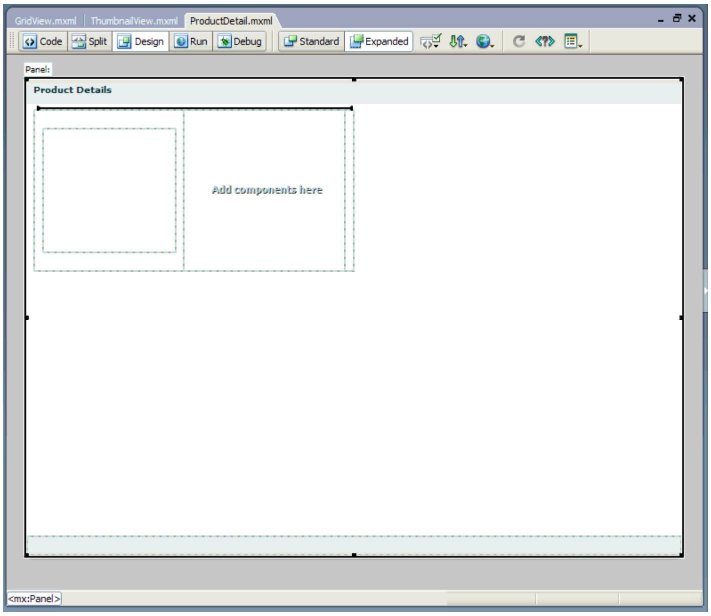
11. To make sure the component fits correctly in the Flex Store user interface, select the Panel container and *clear* the following properties in the Attributes panel:

    - `Size > height`:
    - `Size > width`:

12. Save the component file in the fbComponents folder by pressing Control+S, double-clicking the fbComponents folder, and naming the component file as follows:

    ProductDetail.mxml

In Design view, the component's layout should look similar to the following figure if you select the Panel container:



Note: The component will scale to fit in the space allotted to it in the flexstore.mxml file.

Add the product details

According to the mock-up, the component should display the following information: a thumbnail image, the product name and price, and a brief product description.

1. Make sure the component file, ProductDetail.mxml, is open in Flex Builder.
2. Insert a placeholder image for the product thumbnail by clicking in the upper left corner of the Canvas container, clicking the Image button in the Controls category of the Insert bar, and selecting the following image:

   fbComponents/mockups/tnImage.png

   In the tutorial on data bindings (see "Tutorial: Binding components to data with Flex Builder" on page 59), you modify the `source` property of the Image so that its value is set dynamically when the user clicks a product in the catalog.
3. With the Image still selected in the file, change the following properties in the Attributes panel:
   - `Size > height`: **140**
   - `Size > width`: **150**
   - `Size > x`: **0**
   - `Size > y`: **0**

4. Insert a Label component for the product name by clicking anywhere inside the VBox container, clicking the Label button on the Insert bar, and specifying the following properties in the Attributes panel:

   - `Common > text`: **Product Name**
   - `Common > styleName`: **title**

   In the tutorial on data bindings, you modify the `text` property so that its value is set dynamically when the user clicks a product in the catalog.

   The title style is defined in the flexstore.css file attached to the flexstore.mxml file (see "Import your CSS styles" on page 29). Your custom component will have access to this style sheet when the component is used in the flexstore.mxml file.

5. Insert another Label component for the product price by switching to Code view (View > Code), clicking immediately after the existing `<mx:Label>` tag and clicking the Label button on the Insert bar.

   Flex Builder inserts a `<mx:Label>` tag.

6. Click anywhere in the new `<mx:Label>` tag, click the Refresh button on the Attributes panel to display the tag properties, and specify the following properties:

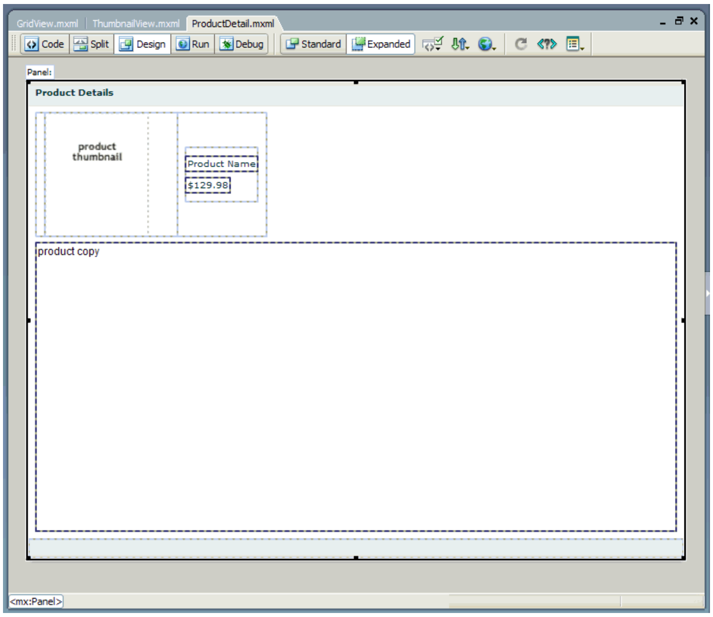
   - `Common > text`: **$129.98**
   - `Common > styleName`: **price**

7. Switch back to Design view and insert a Text component to display the product description by clicking inside the Panel container outside and below the HBox container, clicking the Text button on the Insert bar (not to be confused with the Label button), and specifying the following properties in the Attributes panel:

   - `Common > html`: **true**
   - `Common > text`: **product copy**
   - `Size > heightFlex`: **1**
   - `Size > widthFlex`: **1**

8. Save your work.

In Design view, the component should look similar to the following figure if you select the Panel container:



Finish the footer of the product detail component

According to the component mock-up, the footer has a control that lets users specify the quantity of the selected item they want to add to the shopping cart. The footer also has a button that lets users add the selected item to the shopping cart.

1. Make sure the component file, ProductDetail.mxml, is open in Flex Builder.
2. In Design view, click inside the ControlBar container at the lower edge of the Panel container.

   Make sure the blinking insertion bar appears inside the ControlBar container.
3. Insert a Label component by clicking anywhere inside the ControlBar container and clicking the Label button in the Controls category of the Insert bar.
4. Modify the label text by double-clicking the Label component in Design view to open the Quick Tag Editor, and then changing the value of the `text` property as follows (shown in bold type):

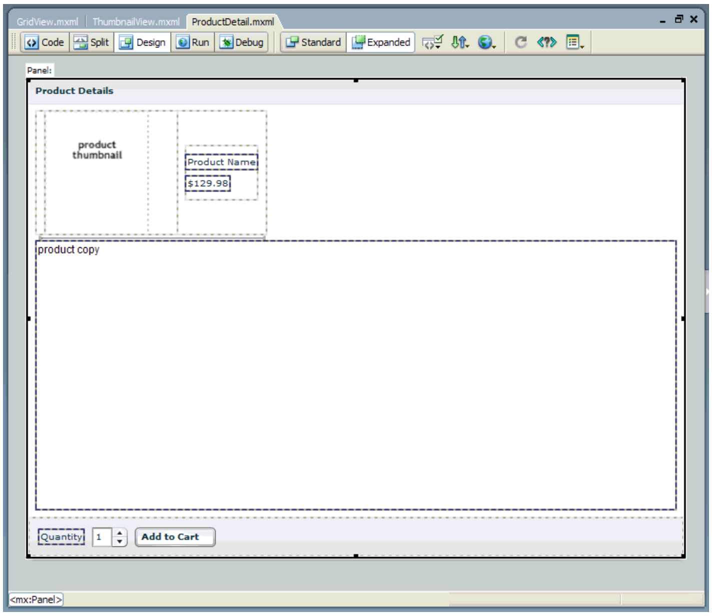
   ```
   <mx:Label text="Quantity" />
   ```
5. Press Enter to accept your change and close the Quick Tag Editor.

6. Insert a NumericStepper control by clicking to the right of the Quantity label in the ControlBar container, clicking the NumericStepper button in the Controls category of the Insert bar, and specifying the following properties in the Attributes panel:
   - `Common > id`: **qty**
   - `Common > maximum`: **100**
   - `Common > minimum`: **1**
   - `Common > stepSize`: **1**
   - `Common > value`: **1**
   - `Size > width`: **40**
7. Insert a button by clicking to the right of the NumericStepper control in the ControlBar container, and clicking the Button control on the Insert bar.
8. Modify the button text by double-clicking the button to open the Quick Tag Editor, and then changing the value of the `label` property as follows (shown in bold type):

```
<mx:Button label="Add to Cart" />
```

9. Press Enter to accept your change and close the Quick Tag Editor.
10. Save your work.

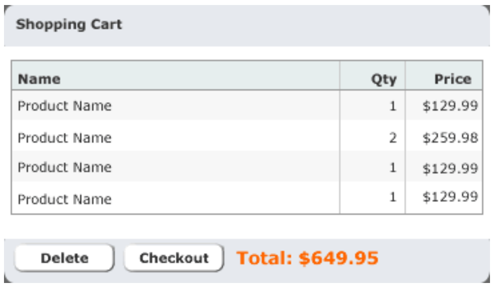
In Design view, the completed product detail component should look similar to the following figure if you select the Panel container:



Build the shopping cart component

According to the component's functional specification, the shopping cart component should display the products that the user wants to bring to the checkout area for purchase. The user can delete items from the cart by selecting an item and clicking the Delete button. The user can also proceed to the checkout area by clicking the Checkout button.

The following is a mock-up of the component:


Shopping Cart

Name	Qty	Price
Product Name	1	$129.99
Product Name	2	$259.98
Product Name	1	$129.99
Product Name	1	$129.99

The shopping cart component does not contain any data in this tutorial. Another tutorial describes how to add data (see "Tutorial: Binding components to data with Flex Builder" on page 59).

The Checkout button is disabled in these tutorials.

In this part of the tutorial, you complete the following tasks:

- "Lay out the shopping cart component" on page 48
- "Add the product list to the shopping cart" on page 49
- "Finish the footer of the shopping cart component" on page 51

Lay out the shopping cart component

After studying the component mock-up, you decide to use the following Flex containers to lay out the component:

- A Panel container to create the header and position the component's child containers vertically
- A ControlBar container to create the footer

You start by creating a component file based on a Panel container.

1. In Flex Builder, press Control+N.

   The New Document dialog box appears.

2. Select Flex Development in the left pane and double-click MXML Component:Panel in the right pane.

   The dialog box closes and a component file with a Panel container appears.

3. Double-click the Panel container and enter the following property value (shown in bold type) in the Quick Tag Editor:

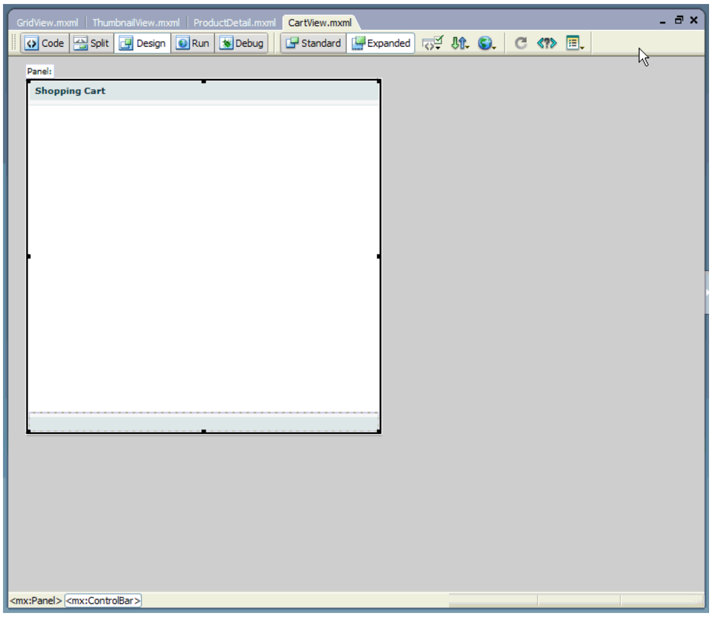
   ```
   <mx:Panel... title="Shopping Cart" />
   ```

4. Press Enter to accept your change and close the Quick Tag Editor.
5. Insert a ControlBar container by clicking inside the Panel container and clicking the ControlBar button in the Containers category of the Insert bar.

   Flex Builder inserts an empty ControlBar container and automatically positions it at the lower edge of the Panel container.
6. With the insertion point still blinking in the ControlBar container, specify the value for the following property in the Attributes panel:
   - `Common > id`: **ctrl**
7. Save the component file in the fbComponents folder by pressing Control+S and naming the component file as follows:

   CartView.mxml

In Design view, the component's layout should look similar to the following figure if you select the Panel container:



Add the product list to the shopping cart

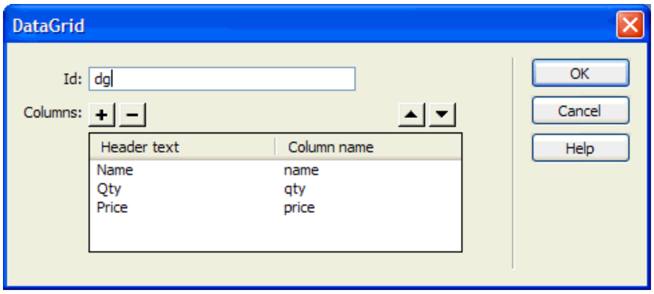
According to the mock-up, the component should display a list of products the user wants to bring to the checkout area for purchase. The list should include the quantity and price of each product. You decide to use a DataGrid component for this task.

1. Make sure the component file, CartView.mxml, is open in Flex Builder.
2. Insert a DataGrid by clicking anywhere inside the Panel container and clicking the DataGrid button in the Controls category of the Insert bar.

   The DataGrid dialog box appears.

3. Click the Plus (+) button to add a third column, and then set the following options in the DataGrid dialog box:
   - `ID`: **dg**
   - `Header Text (first row)`: **Name**
   - `Header Text (second row)`: **Qty**
   - `Header Text (third row)`: **Price**
   - `Column Name (first row)`: **name**
   - `Column Name (second row)`: **qty**
   - `Column Name (third row)`: **price**

   The DataGrid dialog box should look similar to the following figure:



Id: dg

Header text	Column name
Name	name
Qty	qty
Price	price

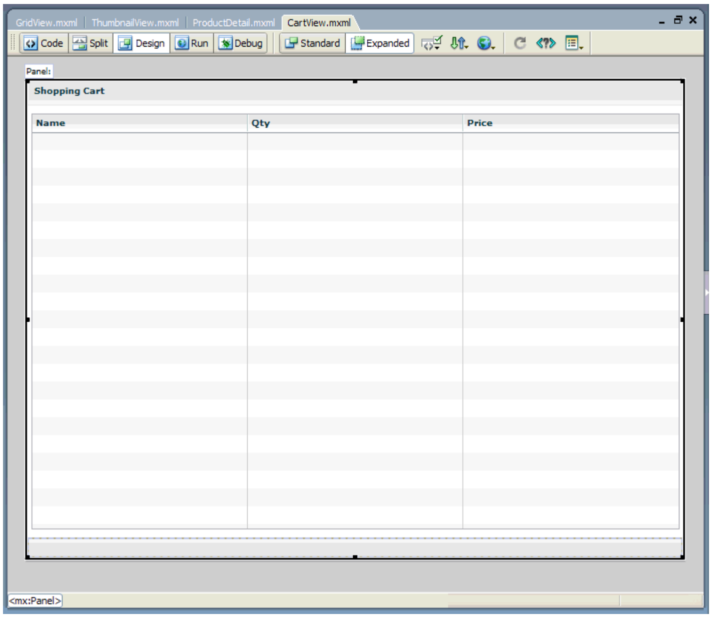
4. Click OK.

   Flex Builder inserts a DataGrid component in your component file.
5. Click the DataGrid component to select it and set the following properties in the Attributes panel:
   - `Size > heightFlex`: **1**
   - `Size > widthFlex`: **1**
6. Switch to Code view (View > Code) and set the column properties of the DataGrid component by locating the three `<mx:DataGridColumn>` tags and adding the following properties (shown in bold type):

```
<mx:DataGridColumn headerText="Name" columnName="name" width="180" />
<mx:DataGridColumn headerText="Qty" columnName="qty" width="50"
  textAlign="right" marginRight="4" />
<mx:DataGridColumn headerText="Price" columnName="price" width="60"
  textAlign="right" marginRight="4" />
```

   Tip: You can use Code hints to quickly set these properties.
7. To make sure the component fits correctly in the Flex Store user interface, locate and delete the `height` and `width` properties in the `<mx:Panel>` tag.
8. Save your work.

In Design view, the component should look similar to the following figure if you select the Panel container:



Note: The component will scale to fit in the space allotted to it in the flexstore.mxml file.

Finish the footer of the shopping cart component

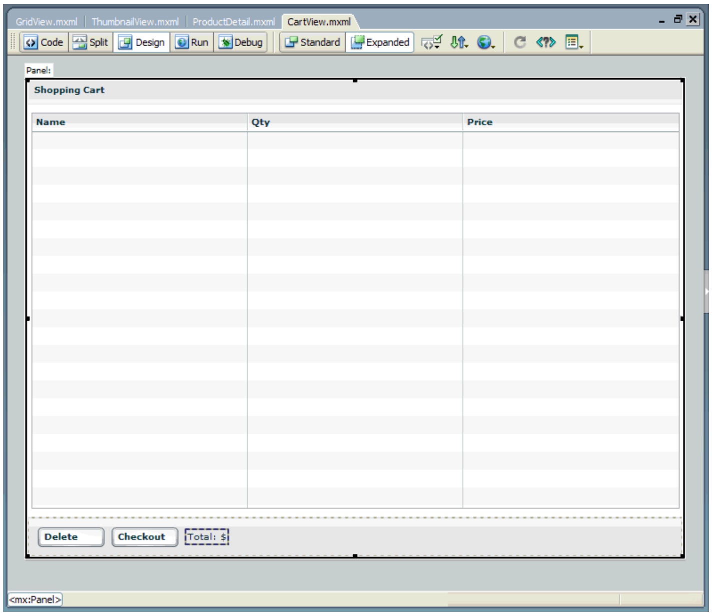
According to the component mock-up, the footer has one button to let the user delete a selected item in the shopping cart, and another button to let the user proceed to the checkout area. The footer should also display the total price of all the products in the cart.

1. Make sure the component file, CartView.mxml, is open in Flex Builder.
2. In Design view, click anywhere inside the ControlBar container at the lower edge of the Panel container.

   Make sure the blinking insertion bar appears inside the ControlBar container.
3. Insert a button by clicking the Button control in the Controls category of the Insert bar, and specifying the values for the following properties in the Attributes panel:
   - `Common > label`: **Delete**
   - `Size > width`: 75
4. Insert a second button by clicking to the right of the Delete button in the ControlBar container, clicking the Button control on the Insert bar, and specifying the values for the following properties in the Attributes panel:
   - `Common > label`: **Checkout**
   - `Size > width`: 75

5. Insert a Label component by clicking to the right of the Checkout button in the ControlBar container, clicking the Label button on the Insert bar, and specifying the values for the following properties in the Attributes panel:
   - `Common > text`: **Total: $**
   - `Common > styleName`: **price**
6. Save your work.

In Design view, the CartView component should look similar to the following figure if you select the Panel container:



Insert the view components in the Flex Store layout

After completing the custom components, you're ready to insert them in the Flex Store layout. The layout is already created in this tutorial. To create it yourself, see "Tutorial: Creating a layout with Flex Builder" on page 26).

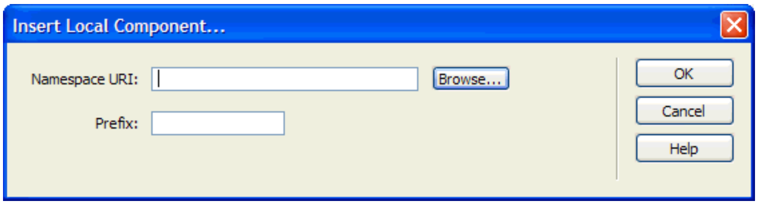
The product catalog is positioned in the Flex Store layout with a ViewStack container so that two views of the catalog—a grid view and a thumbnail view—can occupy the same space. For more information, see "Position the catalog component" on page 31. In this part of the tutorial, you insert the two components you created, GridView.mxml and ThumbnailView.mxml, into the ViewStack container.

1. Open the fbComponents/flexstore.mxml file in Flex Builder.

   Make sure you open the file located in the fbComponents folder.

2. In Design view, click anywhere inside the ViewStack container in the Product Catalog panel, and select Insert > Local Component.

   The Insert Local Component dialog box appears:



3. Click the Browse button and select the ThumbnailView component, ThumbnailView.mxml, in the Open dialog box that appears.

   After you click Open and then OK to close the dialog boxes, Flex Builder inserts the custom component in the ViewStack container.

4. Insert the GridView component by switching to Code view, clicking immediately after the `<local:ThumbnailView>` tag, selecting Insert > Local Component from the menu, and selecting your GridView component.

   You insert the GridView component in the same ViewStack container as the ThumbnailView component.

5. Click anywhere in the opening `<mx:Panel>` tag, click the Refresh button on the Attributes panel to display the tag's properties, and modify the following values:

   - `Size > height`: Clear the value.
   - `Size > width`: **484**

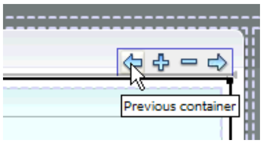
6. Locate the `<local:ThumbnailView>` tag, and add the following property (shown in bold type):

```
<local:ThumbnailView xmlns:local="*" id="thumbView" />
```

7. Locate the `<local:GridView>` tag, and add the following property (shown in bold type):

```
<local:CartView xmlns:local="*" id="gridView" />
```

8. In Design view, you can switch between the two catalog views by selecting the ViewStack container and then clicking the left or right arrows at the top of the container.



9. Save your work.

Insert the detail and cart components in the layout

Two Canvas containers were inserted in the Flex Store layout to position the product detail and shopping cart components. In this part of the tutorial, you insert the two components that you created, ProductDetail.mxml and CartView.mxml, into the Canvas containers.

1. Make sure the flexstore.mxml file is open in Flex Builder.

   The file is located in the fbComponents folder.

2. In Design view, insert the product detail component by clicking in the first Canvas container (ID is topCanvas) on the right side of the layout, and selecting Insert > Local Component.

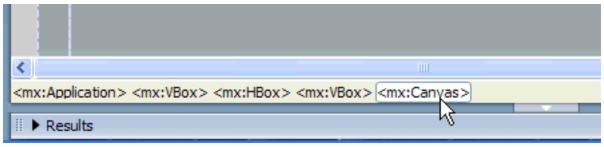
   The Insert Local Component dialog box appears.

3. Click the Browse button in the dialog box that appears, and select the ProductDetail component, ProductDetail.mxml.

4. Click Open, and then click OK to close the dialog boxes.

   Flex Builder inserts the custom component in the Canvas.

5. Click the `<mx:Canvas>` tag on the tag selector on the lower edge of the Document window to select the container, and then clear the `width` and `height` properties in the Tag inspector.

   These values were for prototyping purposes only.

6. Insert the CartView component by clicking in the second Canvas container (ID is bottomCanvas) and inserting CartView.mxml using the technique described in steps 2 through 4.

7. Select the second Canvas container and clear the `width` and `height` properties in the Tag inspector.

8. Switch to Code view (View > Code), locate the `<local:ProductDetail>` tag, and add the following properties (shown in bold type):

```
<local:ProductDetail xmlns:local="*" id="productDetail" height="330"
  width="370" vScrollPolicy="off" />
```

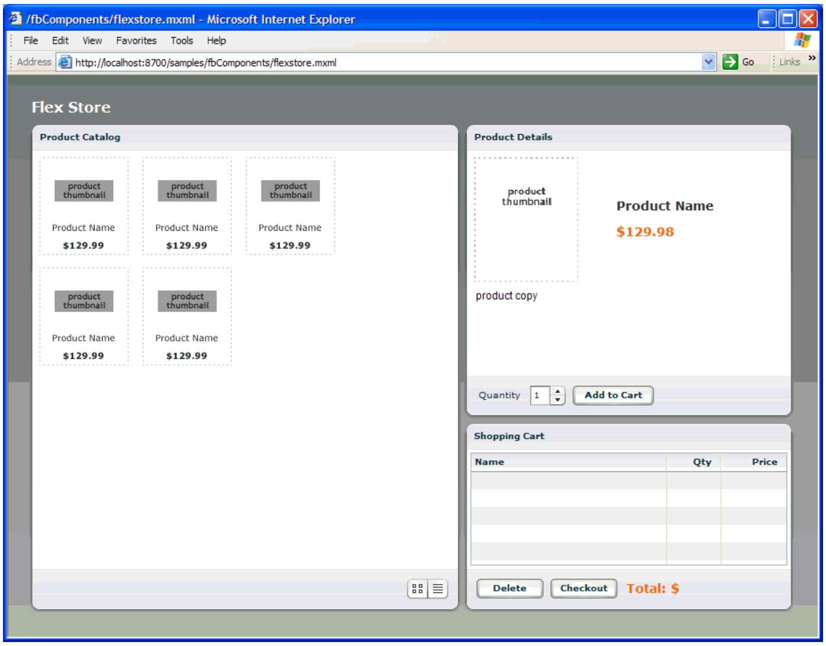
9. Locate the `<local:CartView>` tag, and add the following properties (shown in bold type):

```
<local:CartView xmlns:local="*" id="cartView" height="212" width="370" />
```

10. Save your work.

11. Make sure the Flex server is started, and then press F12 to run the flexstore.mxml file in a browser.

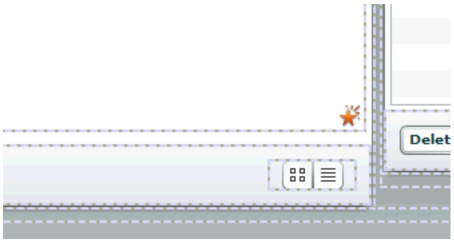
    The file should look similar to the following figure:



    If the Results panel displays validation errors, carefully retrace the preceding steps. If you get a server error, make sure the testing server is correctly defined in Flex Builder. For more information, see "Define a Flex Builder site" on page 25.

Activate the catalog view buttons

According to the functional specifications, the Flex Store application should give the user the ability to switch between the grid view and the thumbnail view of the product catalog. The user interface contains two buttons for that purpose at the lower edge of the product catalog, as in the following figure:



In this part of the tutorial, you configure the view buttons so that the user can switch views.

You decide to write an ActionScript function called `changeView()` that sets the child component of the ViewStack container, which displays when the user clicks one of the buttons. The ViewStack container has two child components: the GridView and the ThumbnailView components that you inserted in it (see "Insert the view components in the Flex Store layout" on page 52).

To make your code more readable, you decide to place the ActionScript function in a separate file called flexstore_script.as, which you then import in the flexstore.mxml file.

1. In Flex Builder, press Control+N.

   The New Document dialog box appears.

2. Select Flex Development in the left pane and double-click the ActionScript item in the right pane.

   The dialog box closes and a blank ActionScript file appears.

3. Copy and paste, or enter, the following code into the file:

   ```
   var currentView="thumb";
   // Handle view button events
   function changeView(view) {
     currentView=view;
     if (view=="thumb") {
       bodyStack.selectedChild=thumbView;
     } else if (view=="grid") {
       bodyStack.selectedChild=gridView;
     }
   }
   ```

   Note: Make sure you include the variable declaration at the top of the code sample.

   If the user clicks the thumbnail button, the ViewStack container displays the ThumbnailView component. If the user selects the list button, the ViewStack container displays the GridView component.

   Note: You assigned the IDs bodyStack, gridView, and thumbView to the ViewStack, GridView, and ThumbnailView components, respectively.

4. Save the ActionScript file in the fbComponents folder by pressing Control+S and naming the file as follows:

   flexstore_script.as

5. Switch to your flexstore.mxml file, make sure you're in Code view, and enter the following tag immediately after the opening `<mx:Application>` tag:

   ```
   <mx:Script source="flexstore_script.as" />
   ```

   This line imports the ActionScript file into the flexstore.mxml file.

   Tip: To quickly open an ActionScript file in Flex Builder, click anywhere on the filename in the <mx:Script> tag (in this case, flexstore_script.as) and press Control+D.

6. Assign the event handler to the thumbnail view button by clicking anywhere in the first `<mx:Image>` tag in the flexstore.mxml file and setting the following property in the Events panel of the Tag inspector:

   - `mouseDown`: **changeView('thumb')**

   When the user clicks the image, the `changeView()` function is called.

7. Assign the event handler to the grid view button by clicking anywhere in the second `<mx:Image>` tag and setting the following property in the Events panel:

   - `mouseDown`: **changeView('thumb')**

8. Save the flexstore.mxml file.
9. Click the Run button on the Document toolbar to test the buttons in the embedded browser.

   You discover a bug. When you click the grid view button at the lower edge of the product catalog, the application doesn't switch to the grid view. The view should change when you click the button.

   You decide to run the ActionScript debugger to locate and fix the problem.

Debug the view buttons

One of the catalog view buttons is broken: it won't display the catalog's grid view. You believe you can find clues to the problem in the ActionScript file that contains the buttons' event handler. You decide to run the ActionScript debugger to pinpoint the problem.

1. Make sure both the flexstore.mxml and flexstore_script.as files are open in Flex Builder.
2. Switch to the flexstore_script.as file and review the event handler for the view buttons:

```
function changeView(view) {
  currentView=view;
  if (view=="thumb") {
    bodyStack.selectedChild=thumbView;
  } else if (view=="grid") {
    bodyStack.selectedChild=gridView;
  }
}
```

   The changeView function will select `gridView` only if the string "grid" is passed to the function's `view` argument. You decide to set a breakpoint to check the value passed to the function.

3. Set a breakpoint on the following line by Control-clicking the line's gutter:

```
currentView=view;
```

   A breakpoint marker appears in the gutter. You set a breakpoint on this line because it immediately follows the line that assigns a value to the `view` variable. When the debugger stops here, the `view` variable will have a value.

4. Switch to the flexstore.mxml file and click the Debug button on the Document toolbar.

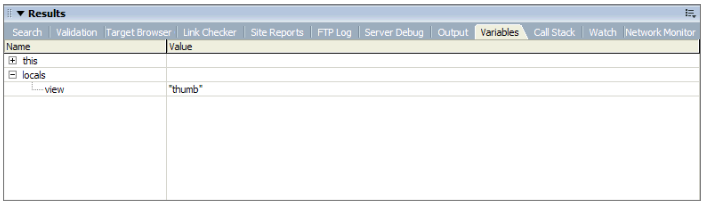
   Flex Builder compiles the application and runs it in its embedded browser.

5. Scroll down the Flex Store page and click the grid view button.

   The click event calls the changeView function in the ActionScript file. Flex Builder does the following:

   - Stops the application's execution at the breakpoint you set.
   - Switches to Code view and centers on the line with the breakpoint.
   - Displays the Call Stack panel in the Results panel group.

6. Click the Variables tab next to the Call Stack tab and then expand the Locals category.



Name	Value
⊞ this	
⊟ locals	
view	"thumb"

   The Variables panel tells you that the `view` variable contains the value "thumb". The value should be "grid". The function is receiving an incorrect value from the grid view button.

7. Switch to the flexstore.mxml file and click the Code button on the Document toolbar.

8. Locate the `<mx:Image>` tag associated with the grid view button (it uses the list_off.png image) and check the value of the changeView function's parameter:

```
<mx:Image source="@Embed('assets/images/list_off.png')"...
  mouseDown="changeView('thumb')" />
```

   The value of the parameter is wrong: it should be `'grid'`.

9. Click the Stop button on the Debug toolbar and change the parameter from `'thumb'` to `'grid'` in the `<mx:Image>` tag.

   Note: You cannot edit files when the debugger is running.

10. Save the file.

11. Click Debug to recompile the application and check the grid view button again.

    After you click the application's grid view button in the embedded browser, the Variables panel should confirm that the value of the `view` variable is now "grid".

12. Click Continue on the Debug toolbar.

    Since you didn't set any other breakpoints, the application finishes executing. It should now display the catalog's grid view.

For more information on using the ActionScript debugger, see "Debugging ActionScript" on page 90.

This completes the components tutorial. If you like, you can continue building the Flex Store by completing the bindings tutorial.

Tutorial: Binding components to data with Flex Builder

This tutorial shows you how to use Flex Builder to visually bind Flex components to data. It also explains how to use the Network Monitor to monitor the data passed between your application and the Flex server.

In a real-world situation, the Flex Store application would connect to a backend system to retrieve and display information about the products. The backend system in this tutorial is simulated with a sample web service installed with the Flex server.

You can complete this tutorial as a stand-alone unit or as the third part of a multipart tutorial. In either case, you must complete the setup tutorial before you begin. For instructions, see "Tutorial: Setting up a development environment" on page 21.

The tutorial includes a pre-built set of files so you can complete the tutorial without completing the layout or components tutorials first. If you completed the components tutorial, you can overwrite the files in the fbBindings folder with your files in the fbComponents folder.

Tip: This tutorial requires that you type a lot of code. If you're working with a printed copy of the tutorial, you may want to paste the code from an electronic version of the tutorial in Using Flex Builder Help.

In this tutorial, you will accomplish the following tasks:

- "Bind the view components" on page 59
- "Bind the product detail component" on page 66
- "Bind the shopping cart component" on page 71

Bind the view components

The product catalog in the Flex Store layout consists of two custom components, GridView.mxml and ThumbnailView.mxml, which provide different views of the catalog. Your job is to bind the two custom components to product data.

In this part of the tutorial, you complete the following tasks:

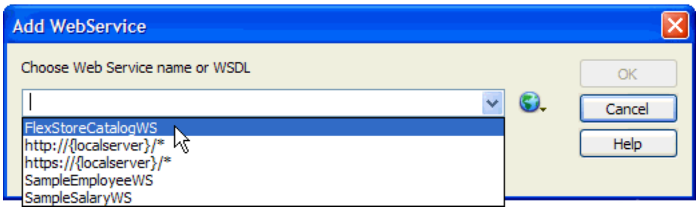
- "Specify the web service to use" on page 60
- "Bind the web service to a data model" on page 61
- "Send the web service request" on page 62
- "Bind the GridView component to the data" on page 64
- "Bind the ThumbnailView component to the data" on page 65

Specify the web service to use

The source of the Flex Store product data is a web service that returns an array containing the data.

1. In Flex Builder, open the flexstore.mxml file located in the fbBindings folder (not the fbComponents folder).
2. In the Data panel (Window > Data), click the Plus (+) button, and select Web Service from the pop-up menu.

   The Add Web Service dialog box appears.
3. Select FlexStoreCatalogWS from the pop-up menu.



   A web service with this name is mapped in the whitelist for the Flex server samples (/samples/WEB-INF/flex/flex-config.xml).

   Note: If you get an error message that says the WSDL fetch failed, make sure the Flex server is running. Also make sure you entered the correct values for the local and the Flex server root folders. The values must be exactly the same as those described in the site setup tutorial. See "Define a Flex Builder site" on page 25.

4. Click OK.

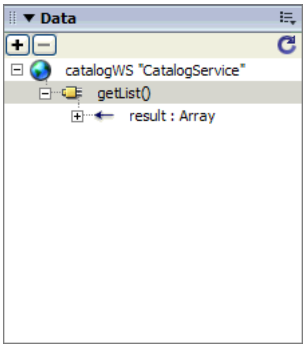
   The web service appears in the Data panel.
5. Click the web service in the Data panel to select it and specify the value of the following property in the Attributes panel:

   ■ `Common > id`: **catalogWS**

   Note: The angle bracket means the ID property is located in the Common category of the Attributes tab. This convention is used throughout the tutorials.

6. Expand the tree control in the Data panel to inspect the web service.

   The catalogWS web service has one method called getList that returns an array.



Bind the web service to a data model

A Flex data model provides a convenient way to refer to and manipulate data in Flex applications. For the Flex Store, you decide to create a data model and bind it to the product data returned by the web service.

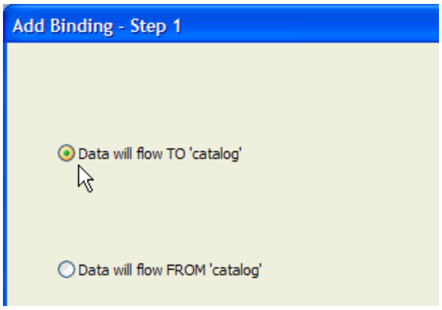
1. Make sure your flexstore.mxml file is open in Flex Builder.
2. Switch to Code view, and insert a data model by entering the following lines after the closing `</mx:WebService>` tag:

```
<mx:Model id="catalog">
</mx:Model>
```

3. Switch to Design view.

   The new catalog model appears in the Data panel.
4. Select the catalog model in the Data panel, and then, in the Bindings panel in the Tag inspector, click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears. Make sure the first option is selected:



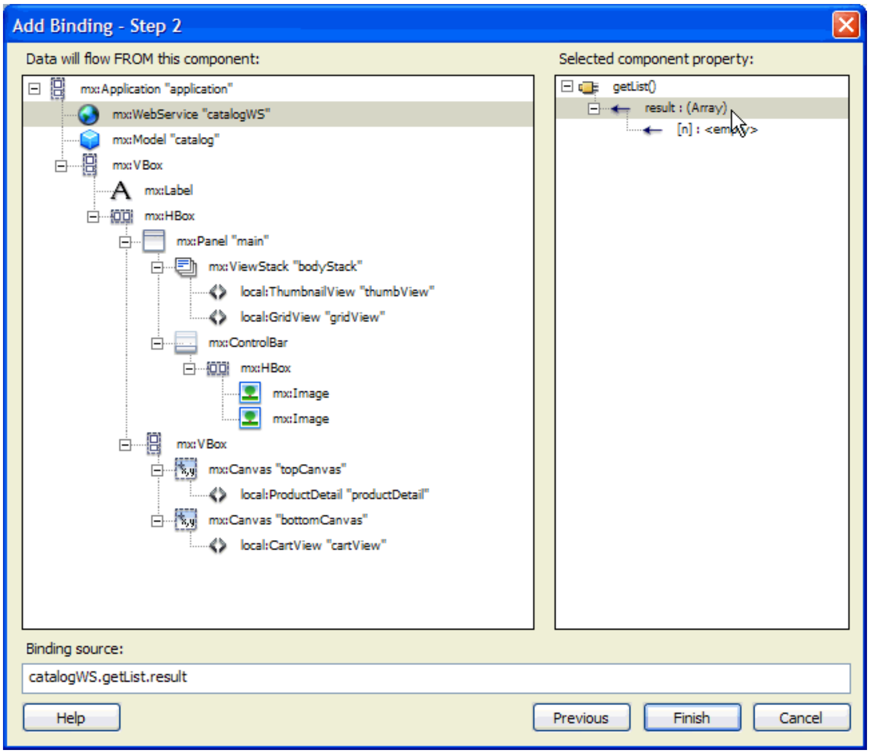
[✓ Data will flow TO 'catalog']
Data will flow FROM 'catalog'

   This option specifies the destination of the data. In this case, the destination is the data model called catalog.
5. Click Next to go to Step 2.

6. In the left pane in the Add Binding - Step 2 dialog box, select the catalogWS web service.

   The Flex component containing the data you want to pass to the data model is the web service.

7. In the right pane, expand the tree control, and select result.



8. Click Finish to create the binding.

   The new binding appears in the Bindings panel.

9. Save your work.

Send the web service request

You must specify an event that sends a request to the web service for the product data. The Flex Store functional specification approved by the client says that the product catalog must be displayed every time the flexstore.mxml file is opened in a browser. Therefore, you must send the web service request when the application is initialized. You decide to write an event handler that sends the request, and then you assign the handler to the initialization event of the application.

1. Open the flexstore_script.as file located in the fbBindings folder.

2. Enter the following event handler function in the ActionScript file:

```
function initApp() {
  catalogWS.getList.send();
}
```

3. Save the ActionScript file.
4. Switch to the flexstore.mxml file.
5. In Code view, assign the event handler to the application's initialize event by locating the `<mx:Application>` tag, and adding the following property (shown in bold type):

   ```
   <mx:Application
     xmlns:mx="http://www.macromedia.com/2003/mxml"
     initialize="initApp()"
     ...
   ```

   When the application is initialized, the `initApp()` function is called.
6. Save the flexstore.mxml file.

Verify that the web service works

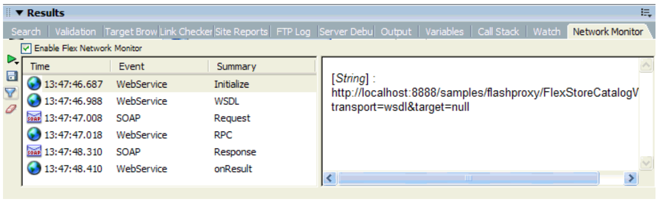
Before you continue, you decide to make sure the web service is called when the application is initialized and data is successfully passed to the application. You want to catch and fix any data problems early. Since the data is not yet bound to any visual controls that you could use to check, you use the Network Monitor in Flex Builder to make sure the web service is passing catalog data to your application.

When enabled, the Network Monitor intercepts all data transactions with the server and displays the information in a panel.

1. Make sure your flexstore.mxml file is open in Flex Builder.
2. In the Network Monitor panel (Window > Network Monitor), click the Enable Flex Network Monitor checkbox on the top edge of the panel.
3. Click the Filter Settings button on the panel's sidebar and deselect the following event types: trace, HTTP, XML, AMF, and RemoteObject.

   You don't want to track these events in this situation, but you do want to track SOAP and WebService events.
4. Click the Run button on the Document toolbar.

   While the Flex server compiles the flexstore.mxml file and Flex Builder displays it in the embedded browser, the Network Monitor monitors and lists all the captured events of the type you specified. The left side of the panel displays the event types while the right side displays specific information for the selected event.

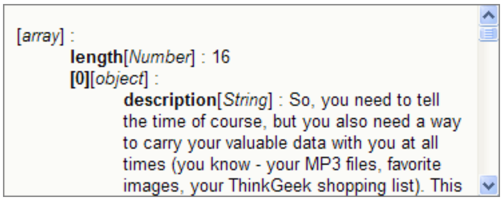


Time	Event	Summary
13:47:46.687	WebService	Initialize
13:47:46.988	WebService	WSDL
13:47:47.008	SOAP	Request
13:47:47.018	WebService	RPC
13:47:48.310	SOAP	Response
13:47:48.410	WebService	onResult

```
[String] :
http://localhost:8888/samples/flashproxy/FlexStoreCatalogV
transport=wsdl&target=null
```

5. Click the final WebServer event (onResult) and inspect the information in the right pane.

   If the application successfully called the web service when it was initialized, then the data for the product catalog should appear in the right pane. Note how the Network Monitor converts the raw data passed between the server and the application into ActionScript. Scroll down to view the catalog data passed to the application.

   ```
   [array] :
           length[Number] : 16
           [0][object] :
                   description[String] : So, you need to tell
                   the time of course, but you also need a way
                   to carry your valuable data with you at all
                   times (you know - your MP3 files, favorite
                   images, your ThinkGeek shopping list). This
   ```

   This confirms that the web service is called when the application is initialized and the catalog data is passed to the application.

6. Disable the Network Monitor for now by clicking the Enable Flex Network Monitor checkbox again.

For more information on using the Network Monitor, see "Debugging applications by monitoring interactions with the server" on page 93.

Bind the GridView component to the data

After specifying where and when to retrieve the data, and then verifying that the data is passed to the application, you can use the data to populate the GridView component.

1. Open the GridView.mxml file located in the fbBindings folder.

2. In Code view, enter the following script block after the opening `<mx:VBox>` tag to declare a variable called dataObject:

   ```
   <mx:Script>
     var dataObject;
   </mx:Script>
   ```

   You want to use this variable as a property of the GridView tag that you inserted in the flexstore.mxml file to pass the catalog model in the flexstore.mxml file to your custom component.

   Note: If you use Code hints, Flex Builder automatically inserts a CDATA block for you. You can declare the variable in this block. For more information on CDATA blocks, see "About ActionScript files in Flex applications" on page 149.

3. Locate the `<mx:DataGrid>` tag in the file and add the following property (shown in bold type):

   ```
   <mx:DataGrid id="list"
     dataProvider="{dataObject}"
     ...
   ```

   The product data passed by the flexstore.mxml file is stored in the dataObject variable.

4. Save the GridView.mxml file.

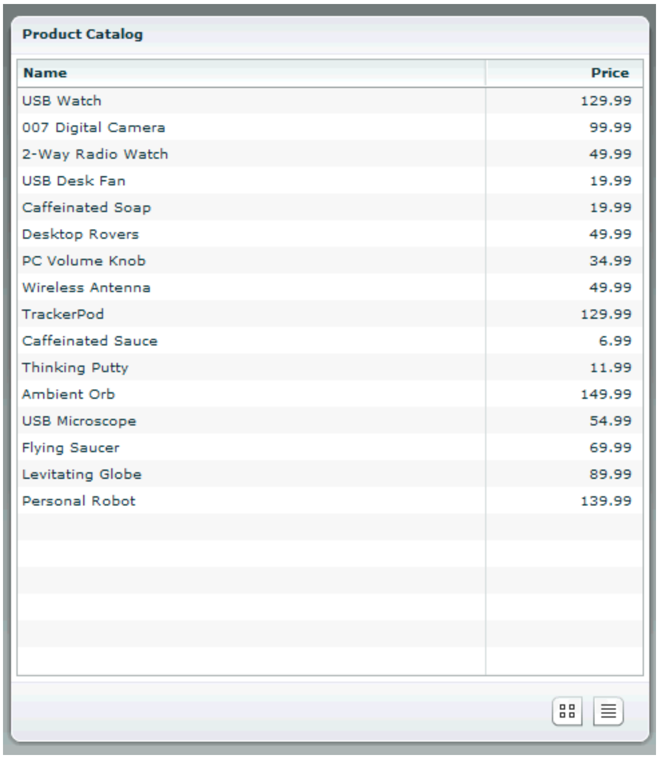
5. Switch to the flexstore.mxml file.

6. In Code view, locate the `<local:GridView>` tag and add the following property (shown in bold type):

```
<local:GridView xmlns:local="*" id="gridView" dataObject="{catalog}" />
```

7. Save the flexstore.mxml file.
8. Make sure the Flex server is running and then press F12 to test the flexstore.mxml file in a browser.

   If the thumbnail view is showing, switch views by clicking the grid view button at the lower edge of the catalog. The catalog should now display the actual product data, as follows:



Product Catalog

Name	Price
USB Watch	129.99
007 Digital Camera	99.99
2-Way Radio Watch	49.99
USB Desk Fan	19.99
Caffeinated Soap	19.99
Desktop Rovers	49.99
PC Volume Knob	34.99
Wireless Antenna	49.99
TrackerPod	129.99
Caffeinated Sauce	6.99
Thinking Putty	11.99
Ambient Orb	149.99
USB Microscope	54.99
Flying Saucer	69.99
Levitating Globe	89.99
Personal Robot	139.99

Bind the ThumbnailView component to the data

After binding the GridView component to the data, you bind the ThumbnailView component.

1. Open the ThumbnailView.mxml file located in the fbBindings folder.
2. In Code view, enter the following script block after the opening `<mx:VBox>` tag to declare a variable called dataObject:

```
<mx:Script>
  var dataObject;
</mx:Script>
```

   You want to use this variable as a property of the ThumbnailView tag, which you inserted in the flexstore.mxml file, to pass the catalog model in the flexstore.mxml file to your custom component.

3. Select and delete all the `<mx:Image>` tags in the Tile container.

   These placeholder images are no longer required.

4. With the insertion point still between the opening and closing `<mx:Tile>` tags, enter the following script block:

   ```
   <mx:Repeater id="list" dataProvider="{dataObject}">
     <ProductThumbnail id="product" xmlns="*" dataObject="{list.currentItem}"
     />
   </mx:Repeater>
   ```

   The product data passed by the flexstore.mxml file is stored in the dataObject variable. You use the custom component called ProductThumbnail to display the product's thumbnail image, name, and price. The custom component file, ProductThumbnail.mxml, is supplied with this tutorial.

5. Save the ThumbnailView.mxml file.

6. Switch to the flexstore.mxml file.

7. In Code view, locate the `<local:ThumbnailView>` tag, and add the following property (shown in bold type):

   ```
   <local:ThumbnailView xmlns:local="*" id="thumbView" dataObject="{catalog}" /
     >
   ```

8. Save the flexstore.mxml file.

9. Press F12 to test the file in a browser.

   If necessary, click the thumbnail view button at the lower edge of the product catalog to switch views.

Bind the product detail component

When a user clicks a product in the catalog in the grid view or the thumbnail view, the application should display details about the product in the ProductDetail component on the right.

In this part of the tutorial, you complete the following tasks:

- "Detect the product in the GridView component" on page 67
- "Detect the product in the ThumbnailView component" on page 67
- "Retrieve the product selection" on page 68
- "Display the product detail" on page 69

Detect the product in the GridView component

The first task is to modify the GridView component so it can detect when a user clicks a product.

1. Open the GridView.mxml file located in the fbBindings folder and enter the following tag after the opening `<mx:VBox>` tag:

   ```
   <mx:Metadata>
     [Event("change")]
   </mx:Metadata>
   ```

2. Locate the `<mx:Script>` tag and add the following variable declaration (shown in bold type):

   ```
   <mx:Script>
     var dataObject;
     var selectedItem;
   </mx:Script>
   ```

   You want to use this local variable to store the product the user selected.

3. Locate the `<mx:DataGrid>` tag and add the following property (shown in bold type):

   ```
   <mx:DataGrid id="list"
     dataProvider="{dataObject}"
     change="selectedItem=event.target.selectedItem;
     dispatchEvent({type:'change'})"
     ...
   ```

4. Save the GridView.mxml file.

Detect the product in the ThumbnailView component

You also modify the ThumbnailView component so it can detect when the user clicks a product and what the product is.

1. Open the ThumbnailView.mxml file located in the fbBindings folder and enter the following tag after the opening `<mx:VBox>` tag:

   ```
   <mx:Metadata>
     [Event("change")]
   </mx:Metadata>
   ```

2. Locate the `<mx:Script>` tag and add the following variable declaration (shown in bold type):

   ```
   <mx:Script>
     var dataObject;
     var selectedItem;
   </mx:Script>
   ```

3. Locate the `<ProductThumbnail>` tag and add the following property (shown in bold type):

   ```
   <ProductThumbnail id="product"
     xmlns="*"
     dataObject="{list.currentItem}"
     mouseDown="selectedItem=event.target.dataObject;
     dispatchEvent({type:'change'})" />
   ```

4. Save the ThumbnailView.mxml file.

Retrieve the product selection

After detecting the product selected by the user, you must pass the information from the GridView and ThumbnailView components to the main application. If the user hasn't selected a product yet, you must specify a default selection.

1. Open the flexstore_script.as file located in the fbBindings folder.
2. Enter the following variable declaration at the top of the page:

   ```
   var selectedItem;
   ```

   You want to use this variable to store the product selected by the user. The variable is automatically incorporated in the flexstore.mxml file because the ActionScript file is included in the file.
3. Save the ActionScript file.
4. Switch to the flexstore.mxml file.
5. In Code view, assign the selected product to the `selectedItem` variable by locating the `<local:GridView>` tag and adding the following property (shown in bold type):

   ```
   <local:GridView xmlns:local="*"
     id="gridView"
     dataObject="{catalog}"
     change="selectedItem=event.target.selectedItem" />
   ```

   When a change is detected in the GridView component—for example, when the user clicks a product in the catalog—the selected product is assigned to the `selectedItem` variable.
6. Locate the `<local:ThumbnailView>` tag and add the following property (shown in bold type):

   ```
   <local:ThumbnailView xmlns:local="*"
     id="thumbView"
     dataObject="{catalog}"
     change="selectedItem=event.target.selectedItem" />
   ```
7. Assign the default product to the `selectedItem` variable by locating the `<mx:WebService>` tag and adding the following property (shown in bold type) in the `<mx:operation>` child tag:

   ```
   <mx:WebService id="catalogWS" serviceName="FlexStoreCatalogWS">
     <mx:operation name="getList"
     result="selectedItem=catalogWS.getList.result[0]" >
     </mx:operation>
   </mx:WebService>
   ```

   The first product in the catalog (`getList.result[0]`) is assigned to the `selectedItem` variable so it can be displayed in the ProductDetail component when the user hasn't selected a product in the catalog yet.
8. Save the flexstore.mxml file.

Display the product detail

After detecting and retrieving the product selection, you can retrieve the product's data and use it in the ProductDetail component.

1. Open the ProductDetail.mxml file located in the fbBindings folder.
2. In Code view, enter the following code after the opening `<mx:Panel>` tag to declare an object variable called dataObject:

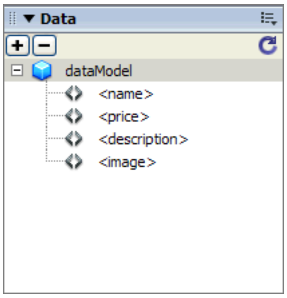
   ```
   <mx:Script>
     var dataObject:Object;
   </mx:Script>
   ```

   The variable declaration creates a property of the ProductDetail component. You want to use this property in the `<local:ProductDetail>` tag in the flexstore.mxml file to pass the product data to your custom component.
3. In Code view, enter the following code after the `<mx:Script>` block you just entered:

   ```
   <mx:Model id="dataModel">
     <name>{dataObject.name}</name>
     <price>{dataObject.price}</price>
     <description>{dataObject.description}</description>
     <image>{dataObject.image}</image>
   </mx:Model>
   ```

   You want to store the data passed to the ProductDetail component in this data model.
4. Switch to the Design view and click the Refresh button on the Data panel.

   The new model appears in the panel.

   The next step is to bind the elements of the data model to the display controls.
5. Select the Label control for the product name and clear the value of the `text` property in the Attributes panel.

   The literal string "Product Name" served as a placeholder until now.

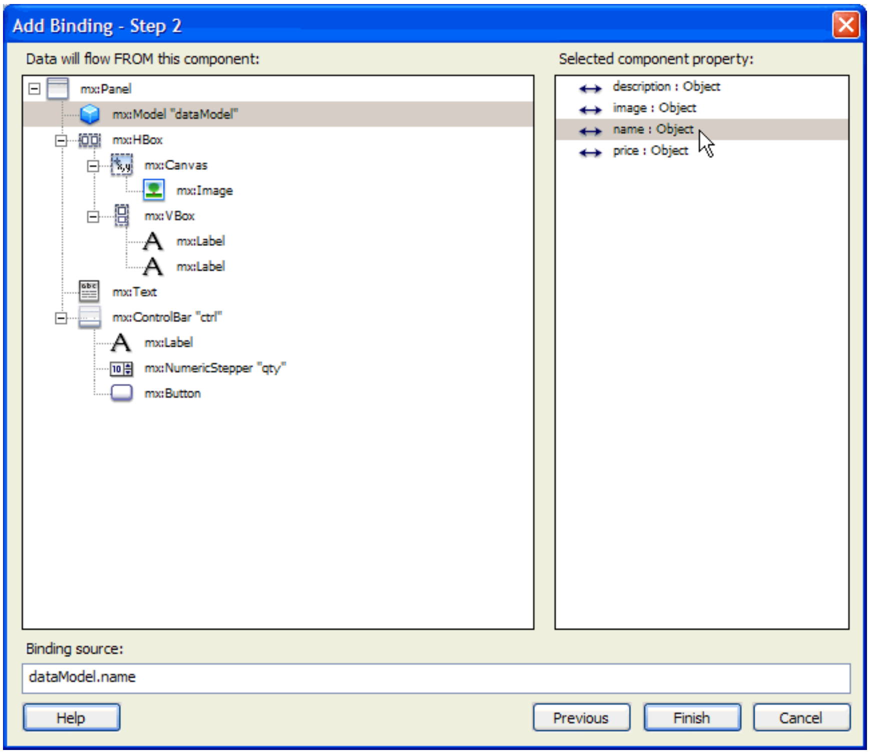
   Note: If you're working in Standard mode, the Label control disappears after you clear the value. Click the Expanded button on the Document toolbar to display the control again.
6. Bind the Label control to the data model by switching to the Bindings panel and clicking the Plus (+) button to start a new binding.

   The Add Binding dialog box appears.

7. With the To option selected, scroll down and select the `text` property (it should be selected by default).

   You want the `text` property to receive the product name data.

8. Specify the source of the data by clicking the Next button, and then selecting the name element of the dataModel data model, as follows:



Binding source:
dataModel.name

9. Click Finish to create the binding.
10. Bind the second Label control to the price data by repeating steps 5 through 9—select the control, clear the placeholder value for the `text` property, start a new binding, select `text` as the destination property, and select the price element of the data model as the source property.
11. Repeat the procedure to bind the Text control to the description data.
12. Bind the Image control by selecting the Image control, clearing the placeholder value for the `source` property, starting a new binding, selecting `source` as the destination property (it should be selected by default), and selecting the image element in the data model as the source property.
13. Save the ProductDetail.mxml file.

14. Switch to the flexstore.mxml file and in Code view, locate the `<local:ProductDetail>` tag and add the following property (shown in bold type):

```
<local:ProductDetail xmlns:local="*"
   id="productDetail"
   dataObject="{selectedItem}"
   ...
```

You bind the `dataObject` property to the `selectedItem` variable to pass the product stored in the variable to the ProductDetail component.

15. Save the flexstore.mxml file.
16. Make sure the Flex server is running and then press F12 to test the file in a browser.

Click any product in either the list view or thumbnail view. The product details should appear in the product detail view (the ProductDetail component) on the right side of the catalog.

Bind the shopping cart component

The shopping cart view (the CartView component) should display the contents of the shopping cart as the user adds products to it. The spec says that it should display the following information: the name of the items added to the shopping cart, the quantity of each item, and the price of each item. The total price of all the products in the cart must be displayed, as well as a button to delete items in the cart and a button to let the user proceed to the checkout pages.

Note: The Checkout button is not activated in this tutorial.

In this part of the tutorial, you complete the following tasks:

- "Add the shopping cart logic" on page 71
- "Add products to the shopping cart" on page 72
- "Display the products in the shopping cart" on page 73
- "Activate the Delete button" on page 74

Add the shopping cart logic

You need a way to keep track of the items in the shopping cart. A member of your team writes an ActionScript class to perform the task. The class is defined in a file called ShoppingCart.as, which is provided in this tutorial.

1. Switch to the flexstore.mxml file.
2. In Code view, add the ShoppingCart class to your application by entering the following tag after the opening `<mx:Application>` tag:

```
<ShoppingCart id="cart" xmlns="*" />
```

You can open the ShoppingCart.as file in Flex Builder to view the class members. ShoppingCart is a model class created in ActionScript that defines the functions you need to track the items that are in the cart, how many there are, and the total price for the items.

3. Save the flexstore.mxml file.

Add products to the shopping cart

The product detail view (ProductDetail component) must not only display details about a product, it must let the user do the following tasks:

- Specify the quantity of the displayed product the user wants to add to the shopping cart
- Add the product and quantity to the shopping cart.

You modify the ProductDetail component so that it adds the displayed product and quantity to the shopping cart when the user clicks the Add to Cart button.

1. Switch to the ProductDetail.mxml file.
2. In Code view, locate the `<mx:Script>` tag and enter the following variable declaration (shown in bold type):

   ```
   <mx:Script>
     var dataObject:Object;
     var shoppingCart;
   </mx:Script>
   ```

   The variable declaration creates a property of the ProductDetail component. You want to use this property in the `<local:ProductDetail>` tag in the flexstore.mxml file to pass a shopping cart object to the component. This object keeps track of items in the shopping cart.
3. Locate the `<mx:Button>` tag in the file, and add the following property (shown in bold type):

   ```
   <mx:Button
     label="Add to Cart"
     click="shoppingCart.addItem(dataObject, qty.value); qty.value=1;" />
   ```

   When the user clicks the Add to Cart button, the `addItem` method of the shoppingCart object adds the product (`dataObject`) and the quantity (`qty.value`) to the shoppingCart object.

   Note: The `qty` identifier is the name of the NumericStepper component that you use to specify a quantity.
4. Save the ProductDetail.mxml file.
5. Switch to your flexstore.mxml file.
6. In Code view, locate the `<local:ProductDetail>` tag, and add the following property (shown in bold type):

   ```
   <local:ProductDetail xmlns:local="*"
     id="productDetail"
     dataObject="{selectedItem}"
     shoppingCart="{cart}"
     ...
   ```

   You bind the `shoppingCart` property to the shopping cart object (cart) to pass the object to the ProductDetail component.
7. Save the flexstore.mxml file.

Display the products in the shopping cart

After using the shoppingCart object to add products to the shopping cart, you can retrieve the data and use it in the CartView component.

1. Open the CartView.mxml file located in the fbBindings folder.
2. In Code view, enter the following code after the opening `<mx:Panel>` tag:

   ```
   <mx:Script>
     var dataObject:ShoppingCart;
   </mx:Script>
   ```

   The variable declaration creates a property of the CartView component. You want to use this property in the `<CartView>` tag in the flexstore.mxml file to pass the shoppingCart object to the component.
3. Locate the `<mx:DataGrid>` tag, and add the following property (shown in bold type):

   ```
   <mx:DataGrid id="dg"
     dataProvider="{dataObject.items}"
     ...
   ```
4. Locate the `<mx:Label>` tag near the end of the file and modify the value of the `text` property as follows (shown in bold type):

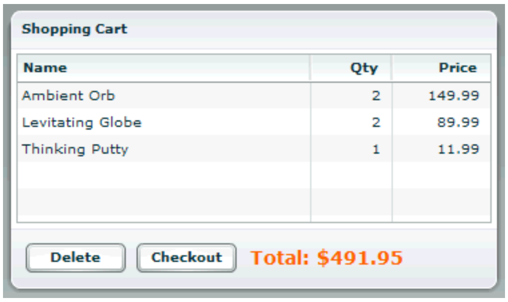
   ```
   <mx:Label styleName="price"
     text="Total: ${dataObject.total}" />
   ```
5. Save the CartView.mxml file.
6. Switch to your flexstore.mxml file.
7. In Code view, locate the `<local:CartView>` tag, and add the following property (shown in bold type):

   ```
   <local:CartView xmlns:local="*"
     id="cartView"
     dataObject="{cart}"
     ...
   ```

   You bind the `dataObject` property to the shopping cart object (cart) to pass the object to the CartView component.
8. Save the flexstore.mxml file.

9. Press F12 to test the file in a browser.

   Click any product in the list view or the thumbnail view. The product details should appear in the product detail area. Click the Add to Cart button to add the product to the shopping cart. Choose another product, change the quantity, and click the Add to Cart button again.



Shopping Cart

Name	Qty	Price
Ambient Orb	2	149.99
Levitating Globe	2	89.99
Thinking Putty	1	11.99

Activate the Delete button

The final step in this tutorial is to active the Delete button in the shopping cart area so the user can remove items from the cart.

1. Switch to the CartView.mxml file.
2. In Code view, enter the following function in the `<mx:Script>` tag (shown in bold type):

```
<mx:Script>
  var dataObject:ShoppingCart;

  function removeItem() {
    if (dg.selectedIndex!=undefined)
      dataObject.removeItemAt(dg.selectedIndex);
  }

</mx:Script>
```

   The function calls the `remoteItemAt` method of the cart object.

3. Locate the `<mx:Button>` tag for the Delete button near the end of the file and add the following two properties (shown in bold type):

```
<mx:Button label="Delete"
  width="75"
  enabled="{dg.selectedItem!=null}"
  click="removeItem()" />
```

   The Delete button is enabled only when a user selects an item in the shopping cart. When the user clicks the button when an item is selected, the `removeItem()` function is called.

4. Save the CartView.mxml file.
5. Switch to the flexstore.mxml file and click Run on the Document toolbar to test the Delete button in the browser embedded in Flex Builder.

   Add items to the shopping cart, select an item in the cart, and click the Delete button. The item should be removed from the cart.

This completes the tutorial on Flex data bindings. If you want, you can continue experimenting with Flex Builder and Flex development by adding more features to your Flex Store application. For example, you could add drag-and-drop support to let the user drag a product from the catalog view to the product detail view.

Next steps

For more samples and tutorials, see the following websites:

- Flex Builder pages on the Macromedia website at www.macromedia.com/go/fb_devcenter/
- Flex Developer Center on the Macromedia website at www.macromedia.com/go/flex_devcenter/

For more information on the Flex Builder features discussed in these tutorials, see the following topics:

- Chapter 3, “Creating, Coding, and Debugging Flex Files,” on page 77
- Chapter 4, “Building a Flex User Interface Visually,” on page 97
- Chapter 5, “Working with Components,” on page 119
- Chapter 6, “Working with Data,” on page 129
- Developing Flex Applications Help
- Flex ActionScript Language Reference Help
- Flex ActionScript and MXML API Reference Help

CHAPTER 3
Creating, Coding, and Debugging Flex Files

Macromedia Flex applications are made up of plain text files containing MXML and ActionScript code, plus assets such as Flash components (SWC files) and images. The Flex server compiles the MXML and ActionScript files into Macromedia Flash SWF files that can run in Macromedia Flash Player or in a web browser with Flash Player installed.

Macromedia Flex Builder provides you with an integrated development environment for writing, editing, debugging, previewing, and deploying MXML and ActionScript files.

This chapter contains the following topics:

- "Creating MXML or ActionScript files" on page 77
- "Working with the code in Flex files" on page 78
- "Testing and debugging Flex files" on page 86

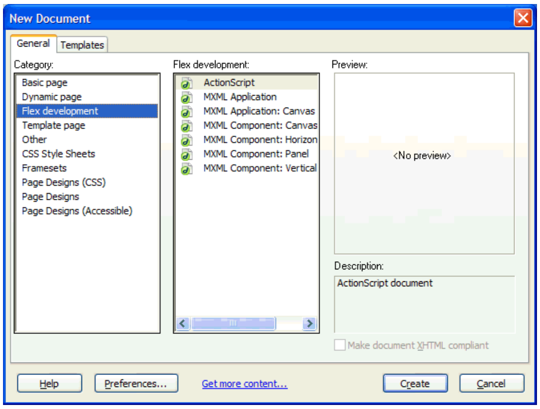
Creating MXML or ActionScript files

You can create MXML or ActionScript files in Flex Builder.

To create a new MXML or ActionScript file:

1. Select File > New.

   The New Document dialog box appears.
2. In the Category list, select Flex Development.



3. In the Document list, select one of the MXML options or ActionScript, and do one of the following:
   - Click Create.
   - Double-click the item in the Document list.
   - Press Enter.

   The dialog box closes, and a new MXML or ActionScript file appears in the Document window.
4. Save and name the new file (File > Save).

Related topics

- "Creating MXML component files" on page 123
- "About Flex files" on page 148
- "Setting a default new document type" in Using Dreamweaver Help

Working with the code in Flex files

Flex Builder provides a number of coding tools for developing Flex applications.

While this section provides information on the main coding features in Flex Builder, it does not cover every coding feature in the product. For more information, see "Coding in Dreamweaver" in Using Dreamweaver Help.

This section covers the following topics:

- "Viewing and editing code in a file" on page 78
- "Using code hints" on page 80
- "Using the Tag inspector" on page 81
- "Setting coding preferences" on page 82

Related topics

- "About Flex files" on page 148

Viewing and editing code in a file

You can view and edit the code in your files by turning on Code view. You can also split the view or work in the Code inspector, a separate coding window. The Code inspector works just like Code view; you can think of it as a detachable Code view for the current file.

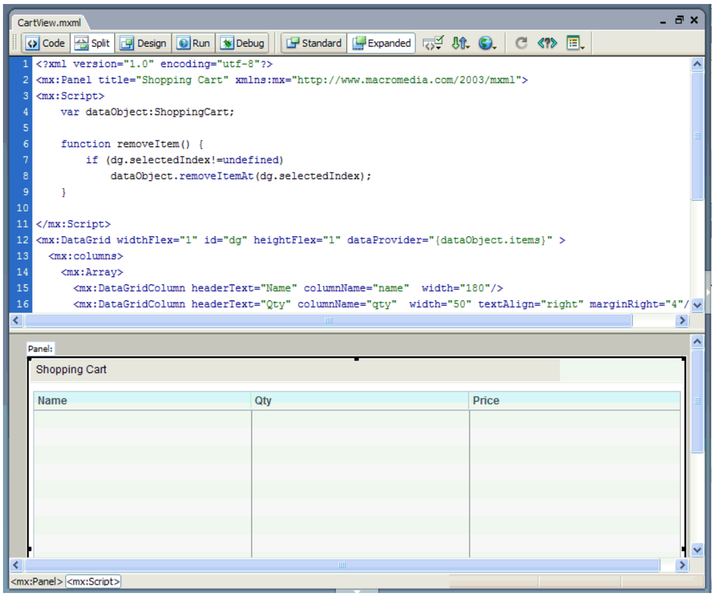
To view the code in the file, do one of the following:

- Click the Code button on the Document toolbar.
- Select View > Code.
- From Design view, press Control+' (the slanted quotation mark that usually shares the tilde [~] key).

To view both Code and Design views:

1. Do one of the following:
   - Click the Split button on the Document toolbar.
   - Select View > Code and Design.

   The code appears in the top pane while a visual representation of the file appears in the bottom pane.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel title="Shopping Cart" xmlns:mx="http://www.macromedia.com/2003/mxml">
<mx:Script>
    var dataObject:ShoppingCart;

    function removeItem() {
        if (dg.selectedIndex!=undefined)
            dataObject.removeItemAt(dg.selectedIndex);
    }

</mx:Script>
<mx:DataGrid widthFlex="1" id="dg" heightFlex="1" dataProvider="{dataObject.items}" >
  <mx:columns>
    <mx:Array>
      <mx:DataGridColumn headerText="Name" columnName="name"  width="180"/>
      <mx:DataGridColumn headerText="Qty" columnName="qty"  width="50" textAlign="right" marginRight="4"/
```

2. To adjust the size of the panes in the Document window, drag the splitter bar to the desired position.

   The splitter bar is located between the two panes.

3. To get a more accurate view of the layout, click the Standard button on the toolbar to remove the padding and borders around objects in Design view.

   Standard mode gives you a more accurate view of the layout. This mode is ideal in Split view if you prefer to work in the code but you still like to get visual feedback about the changes you make. For more information, see "Customizing the Document window" on page 98.

4. Click in either pane to start editing.

   Code view is updated automatically when you make changes in Design view. When you make changes in Code view, however, do one of the following to update Design view:
   - Click anywhere in Design view.
   - Click the Refresh Design View button on the toolbar.
   - Press F5.

To view and edit code in a separate window:

- Select Window > Code Inspector.

Related topics

- "Using the Tag inspector" on page 81
- "Setting code-coloring preferences" on page 83
- "Setting code formatting preferences" on page 84

Using code hints

You can use code hints to write MXML or ActionScript more rapidly and efficiently. If you're working with MXML, the code hints display lists of tags and properties as you type. If you're working with ActionScript, the code hints display function patterns of built-in ActionScript classes (function name, return type, names and types of arguments), as in the following example: `unshift(value:Object):Number`.

Note: If a method or property is inherited, the data type will not be specified.

Code hinting is not supported in custom classes.

To insert a code hint in your code:

- Select the item in the pop-up menu and press Enter, or double-click the item.

To clear the pop-up menu:

- Press Backspace or Escape, or keep typing.

Related topics

- "Setting code hints preferences" on page 85

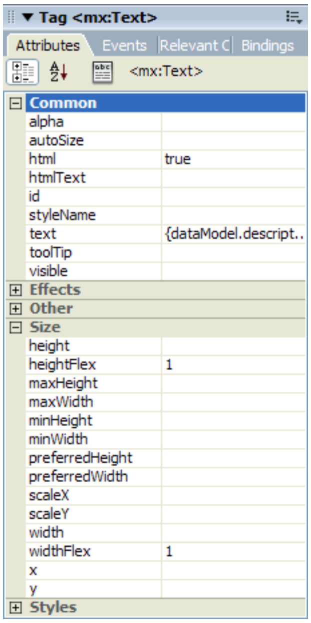
Using the Tag inspector

You can edit all the properties of an MXML tag in one place—the Tag inspector. You can also use the Tag inspector to apply Flex effects, assign event handlers, and modify any CSS styles applied to the tag.

To use the Tag inspector:

1. Click anywhere inside the MXML tag in Code view, or select the component or container in Design view.

   The properties of the tag appear in the Attributes panel of the Tag inspector (Window > Tag Inspector, or press F9).

Common	
alpha	
autoSize	
html	true
htmlText	
id	
styleName	
text	{dataModel.descript..
toolTip	
visible	
Effects	
Other	
Size	
height	
heightFlex	1
maxHeight	
maxWidth	
minHeight	
minWidth	
preferredHeight	
preferredWidth	
scaleX	
scaleY	
width	
widthFlex	1
x	
y	
Styles	

2. Click the value field of a property in the Attributes tab, and enter or edit the value.

   For more information on the various properties, see Flex ActionScript and MXML API Reference Help. For more information on the effect properties, see "Applying effects to Flex components" on page 111.

3. To list the properties alphabetically, click the Show List View button on the panel.

   To list the properties by category again, click the Show Category View button.

Related topics

- "Modifying a CSS style applied to a component" on page 109
- "Applying effects to Flex components" on page 111
- "Creating MXML component files" on page 123

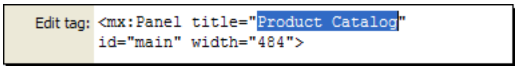
Using the Quick Tag Editor

You can use the Quick Tag Editor to quickly inspect and edit HTML tags without leaving Design view.

To use the Quick Tag Editor:

1. In Design view, double-click any standard MXML component in the file.

   The Quick Tag Editor appears. Depending on the component you double-clicked, the `title`, `text`, or `label` value in the tag may be highlighted and ready for your changes to help you work even faster.



```
<mx:Panel title="Product Catalog"
id="main" width="484">
```

2. Edit the tag and then press Enter when you're finished.

Setting coding preferences

You can customize the Flex coding environment in Flex Builder.

While this section covers some basic coding preferences, it does not cover every preference. For more information, see "Setting Up Your Coding Environment" in Using Dreamweaver Help.

This section contains the following topics:

- "Setting code-coloring preferences" on page 83
- "Setting code formatting preferences" on page 84
- "Setting code hints preferences" on page 85

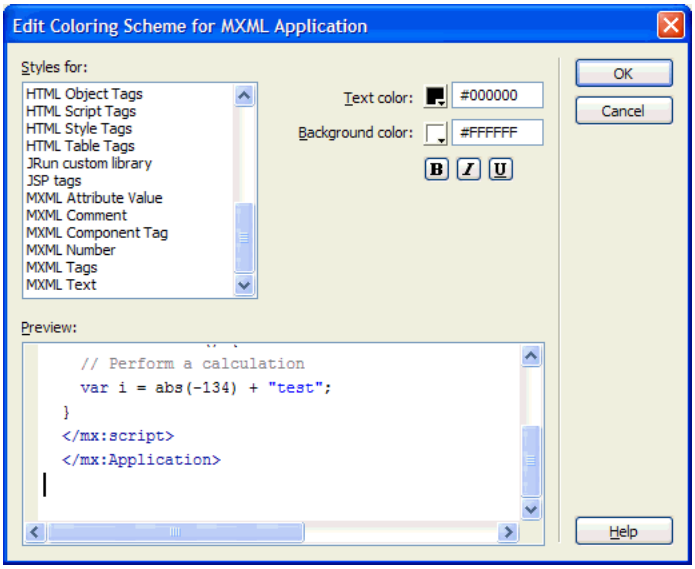
Setting code-coloring preferences

You can change the color of MXML and ActionScript code displayed in Flex Builder.

To set code-coloring preferences for MXML and ActionScript:

1. Select Edit > Preferences > Code Coloring.
2. Select one of the MXML or ActionScript options from the list of document types, and click Edit Coloring Scheme.

   The Edit Coloring Scheme dialog box appears.



Edit Coloring Scheme for MXML Application

Styles for:

- HTML Object Tags
- HTML Script Tags
- HTML Style Tags
- HTML Table Tags
- JRun custom library
- JSP tags
- MXML Attribute Value
- MXML Comment
- MXML Component Tag
- MXML Number
- MXML Tags
- MXML Text

Text color: #000000

Background color: #FFFFFF

Preview:

```
    // Perform a calculation
    var i = abs(-134) + "test";
  }
  </mx:script>
  </mx:Application>
```

3. Edit the code-coloring scheme and click OK.

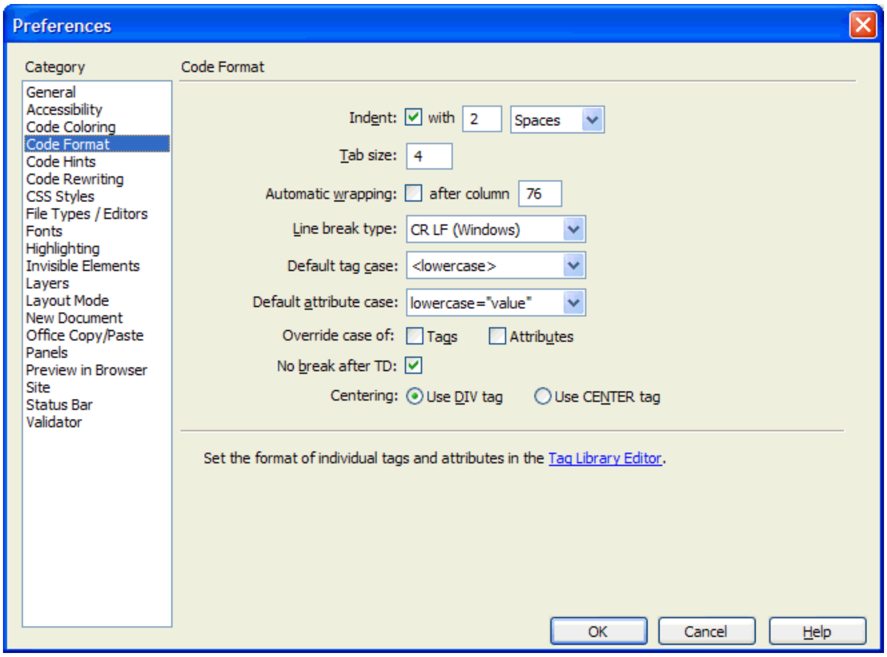
   For more information, click the Help button in the dialog box.

Setting code formatting preferences

You can change the look of your code by specifying formatting preferences such as indentation, line length, and the case of tag and property names.

To set code formatting preferences:

1. Select Edit > Preferences > Code Format.



Preferences

Category

- General
- Accessibility
- Code Coloring
- Code Format
- Code Hints
- Code Rewriting
- CSS Styles
- File Types / Editors
- Fonts
- Highlighting
- Invisible Elements
- Layers
- Layout Mode
- New Document
- Office Copy/Paste
- Panels
- Preview in Browser
- Site
- Status Bar
- Validator

Code Format

Indent: [✓] with 2 Spaces

Tab size: 4

Automatic wrapping: [ ] after column 76

Line break type: CR LF (Windows)

Default tag case: <lowercase>

Default attribute case: lowercase="value"

Override case of: [ ] Tags [ ] Attributes

No break after TD: [✓]

Centering: [✓ Use DIV tag] Use CENTER tag

Set the format of individual tags and attributes in the Tag Library Editor.

OK Cancel Help

2. Set your preferences.

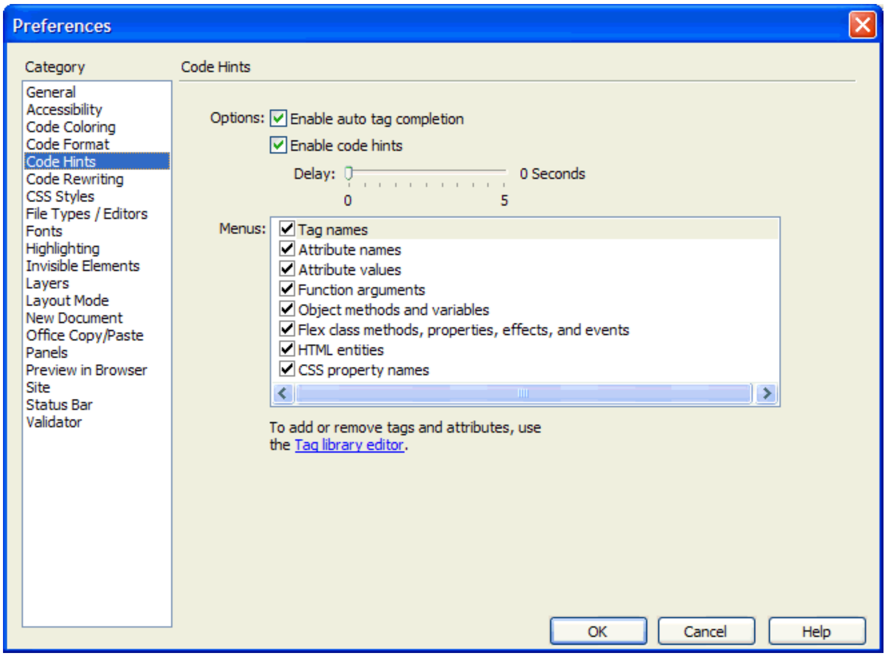
   For more information, click the Help button in the dialog box.

Setting code hints preferences

You can modify the way code hints work by enabling or disabling automatic tag completion, enabling or disabling code hints, changing the length of the delay before hints are displayed, or changing the type of hints you want to see while working.

To set code hints preferences:

1. Select Edit > Preferences > Code Hints.



Options: ☑ Enable auto tag completion
☑ Enable code hints
Delay: 0 Seconds

Menus:
☑ Tag names
☑ Attribute names
☑ Attribute values
☑ Function arguments
☑ Object methods and variables
☑ Flex class methods, properties, effects, and events
☑ HTML entities
☑ CSS property names

To add or remove tags and attributes, use the Tag library editor.

2. Set your preferences.

   For more information, click the Help button in the dialog box.

Related topics

- "Using code hints" on page 80

Testing and debugging Flex files

Flex Builder has several tools to help you test and debug Flex applications, including an MXML validator, an ActionScript debugger, and a network monitor.

This section covers the following topics:

- "Validating MXML" on page 86
- "Compiling files to check for errors" on page 87
- "Compiling and running Flex files" on page 88
- "Debugging ActionScript" on page 90
- "Debugging applications by monitoring interactions with the server" on page 93

Related topics

- "Optimizing and Debugging Your Code" in Using Dreamweaver Help

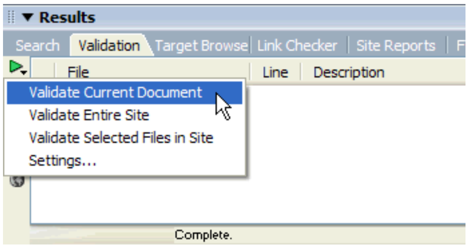
Validating MXML

Flex Builder can validate MXML and display syntax errors or warnings.

Flex Builder automatically validates an MXML file when you perform an action that synchronizes the file, such as switching to Design view, previewing the file, or switching to debug mode. Flex Builder displays any validation errors or warnings in the Validation panel in the Results panel group. You can also validate a file at any time.

To validate a file, do one of the following:

- Press Shift+F6.
- In Code view, after making a change, click the Refresh button on the coding toolbar.
- Select File > Check Page > Validate Markup.
- In the Validation panel in the Results panel group (Window > Results), click the Play button on the sidebar and select one of the validation options.



To jump to an error in your MXML:

- Double-click the error in the Validation panel.

Tip: To validate ActionScript, compile the project. ActionScript errors, if any, will appear in the Output panel.

Compiling files to check for errors

You can compile Flex files from time to time to check for errors and compilation warnings. After the server compiles the file, Flex Builder displays any compile-time errors or warnings in the Output panel. The panel includes ActionScript errors detected in ActionScript files attached to the MXML file or in ActionScript embedded in the MXML file. You can also use the Output panel to quickly jump to and fix errors.

You can also view the file in a browser after compiling it. For more information, see "Compiling and running Flex files" on page 88.

Before you start, make sure you correctly defined a Flex testing server for Flex Builder, and that the server is running. Flex Builder relies on the testing server to compile Flex files and return any error messages. For more information, see "Starting a new application in Flex Builder" on page 9.

To compile a Flex file to check for errors:

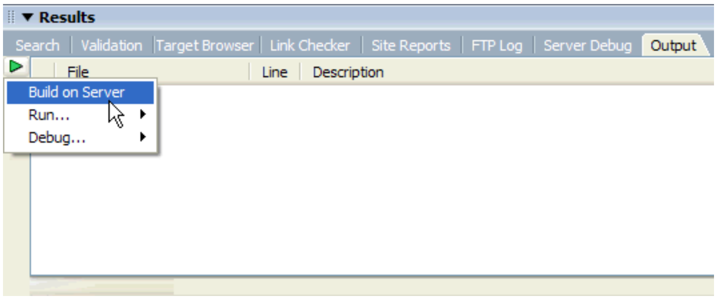
1. Open an MXML application file in Flex Builder.

   An MXML application file contains an `<mx:Application>` root tag.

   If you want to compile custom components, insert them in a test application file and compile the file. For more information, see "Inserting controls and containers in an MXML file" on page 119.

2. Compile the file by doing one of the following:

   - Select File > Build.
   - In the Output panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Build on Server.

   The server compiles the application file and Flex Builder displays any MXML and ActionScript compile-time error messages and warnings in the Output panel.

3. Review any compilation errors or warnings in the Output panel.

   For more information on the Output panel, click the Help button in the panel.

4. Double-click any error or warning message in the panel to open the file in which the error occurred and jump to the line that caused the problem.

Compiling and running Flex files

You can run compiled versions of Flex files in a browser from time to time to check the progress of your design.

If you want to compile and review error messages without running the file in a browser, see "Compiling files to check for errors" on page 87.

Before you start, make sure you correctly defined a Flex testing server for Flex Builder, and that the server is running. Flex Builder relies on the testing server to compile Flex files and display them in a browser. For more information, see "Starting a new application in Flex Builder" on page 9.

This section covers the following topics:

- "Running a file" on page 88
- "Uploading dependent files" on page 89

Running a file

You can run MXML files in a stand-alone browser or in the browser embedded in Flex Builder.

Note: To set up more than one browser, see "Setting the Preview in Browser Preferences options" in Using Dreamweaver Help.

The following is a quick reference, followed by more detailed information:

To do this task...	Perform this action
Run a file in the embedded browser	Press F6.
Exit the embedded browser and return to the editing environment	Click the Code, Split, or Design button on the Document toolbar.
Run a file in an external browser	Press F12.

To run a Flex file:

1. Open an MXML application file in Flex Builder.

   An MXML application file contains an `<mx:Application>` root tag.

   If you want to run custom components, insert them in a test application file and compile the file. For more information, see "Inserting controls and containers in an MXML file" on page 119.

2. Run the file in the browser embedded in Flex Builder by doing one of the following:

   - Press F6.
   - On the Document toolbar, click the Run button.
   - In the Output panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Run > Run in Flex Builder.
   - In the Network Monitor panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Run > Run in Flex Builder.

   A dialog box appears asking you if you want to upload dependent files.

3. Run the file in an external browser by doing one of the following:
   - Press F12.
   - Select File > Run in Browser, and select the browser you want to use.
   - In the Output panel in the Results panel group (Window > Results), click the Play button on the sidebar, select Run, and select the browser you want to use.
   - In the Network Monitor panel in the Results panel group (Window > Results), click the Play button on the sidebar, select Run, and select the browser you want to use.

   A dialog box appears asking if you want to upload dependent files.
4. If copies of dependent files such as images or other files are not already located on the testing server, select Yes in the Dependent Files dialog box.

   Flex Builder uploads most dependent files to the testing server. The server compiles the MXML application file using the files on the server. If a dependent file is missing, then the application file may not compile or may not run properly.

   Note: Flex Builder may not upload all dependent files. For more information, see "Uploading dependent files" on page 89.

   After the server is finished compiling, Flex Builder displays the resulting SWF file in a browser.
5. If you want, try the various components on the page to make sure they work as intended.
6. If you want, review any compilation errors or warnings in the Output panel in the Results panel group (Window > Results).

   You can use the panel to quickly find and fix the errors. Double-click the error message or the warning in the panel to open the file in which the error occurred and jump to the line that caused the error or warning.

   For more information on the Output panel, click the Help button in the panel.
7. To return to the editing environment if using the embedded browser, select one of the view buttons (Code, Split, Design) in the Document toolbar.

Uploading dependent files

Before running or debugging a file, Flex Builder asks if you want it to upload dependent files to the server. Dependent files include ActionScript files, MXML files, image files, SWF files, and other media files.

While Flex Builder will upload most dependent files, it will not upload the following files:

- Any component file referenced by ActionScript in the parent file, such as a pop-up window component or a cell renderer component
- New or modified ActionScript classes placed in ActionScript packages imported in the parent file

To upload these files, do the following:

- If you're working directly in the files on the server, click the Refresh button in the browser to force the server to recompile the files.
- If you're working on the files locally and uploading them to a remote server, upload the component file to the remote server by clicking it in the Files panel and clicking the Put button (the blue up arrow) on the panel toolbar.

Debugging ActionScript

You can use the ActionScript debugger to check your code for ActionScript syntax and logical errors. For example, you can set breakpoints in your script to stop the execution of the script so that you can inspect the values of variables and other information up to that point. You can also step to the next breakpoint or step into a function call to see the variable values change.

The debugger will only stop at breakpoints set on lines containing the following:

- MXML tags containing an ActionScript event handler, such as `<mx:Button click="dofunction()" ...>`
- ActionScript lines such as those enclosed in an `<mx:Script>` tag or in an AS file

Before you start, make sure you correctly defined a Flex testing server for Flex Builder, and that the server is running. Flex Builder relies on the testing server to compile ActionScript and to obtain information about the state of the application. For more information, see "Starting a new application in Flex Builder" on page 9.

Note: To use the ActionScript debugger, the debug version of Flash player is required. Flex Builder installs this version, but it can be overwritten if you install a new version of the player or if you install another Macromedia product such as Breeze. To reinstall the debug version of the player, see the Release Notes on the Macromedia website at www.macromedia.com/go/fb_releasenotes/.

The following table provides a quick debugging reference, followed by more detailed information:

To do the following task...	Perform this action
Set a breakpoint	In Code view, Control-click the file's gutter next to the line of ActionScript where you want to stop.
Start the debugger	Click the Debug button on the Document toolbar or press Alt+F6.
Step through the ActionScript	Click the Continue, Step In, Step Out, and Step Over buttons on the debugging toolbar.
Stop the debugger	Click Stop on the debugging toolbar.
Remove a breakpoint	In Code view, click the breakpoint in the gutter of the file.

To debug ActionScript:

1. Set one or more breakpoints by doing one of the following in Code view:
   - Control-click in the gutter beside the ActionScript line where you want the script to stop executing.
   - Click anywhere inside the line of code and press Control+Alt+B.
   - Click anywhere inside the line of code and select Edit > Set Breakpoint.
   - Right-click anywhere inside the line of code and select Set Breakpoint from the context menu.

   A small dot representing the breakpoint appears in the gutter beside the line.

   Note: Before you can set breakpoints in a file in Code view, the file must exist on the hard disk. Save the file before setting breakpoints.

   To remove one or more breakpoints, do one of the following:
   - To remove one breakpoint, Control-click the breakpoint in the gutter of the file.
   - To remove all breakpoints in the file, select Edit > Remove All Breakpoints in File.
   - To remove all breakpoints in internal files, select Edit > Remove All Internal Breakpoints.

   Tip: You can also right-click the file and select the Edit options in the context menu that appears.

2. If you want, enter one or more traces in the ActionScript and set a breakpoint on the line immediately following the trace.

   For example, you might enter the following trace (shown in bold type) in your ActionScript file:

```
5  ...
6  function myFunc()
7  {
8    trace("Inside my function");
9    btn.label="OK";
10 }
11 ...
```

   If you set a breakpoint at line 9 (`btn.label="OK";`) and start the debugger, the Output panel should display the item “Inside my function” when Flex Builder stops at the breakpoint.

3. Switch to the project's application file (the file with the `<mx:Application>` root tag) and start the debugger by doing one of the following:

   - Press Alt+F6.
   - On the Document toolbar, click the Debug button.
   - Select File > Debug.
   - Select File > Debug in Browser, and select a browser.
   - On the Output panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Debug > Debug in Flex Builder.
   - On the Network Monitor panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Debug > Debug in Flex Builder.

   A dialog box appears asking you if you want to upload dependent files. Dependent files include any ActionScript file, MXML file, or image file referenced in the file.

   If copies of dependent files such as images or other files are not already located on the testing server, select Yes in the Dependent Files dialog box.

   Flex Builder will upload most dependent files to the testing server. The server compiles the MXML application file using the files on the server. If a dependent file is missing, then the application file may not compile or may not run properly.

   Note: Flex Builder may not upload all dependent files. For more information, see "Uploading dependent files" on page 89.

   The Flex server attempts to build the project. If the build is successful, Flex Builder switches to debug mode and the ActionScript executes up to the first breakpoint and stops. If you set a breakpoint in an event handler, you must trigger the event in the application to reach the breakpoint.

   After reaching a breakpoint, Code view becomes read-only and debugging information appears in some of the Results panels.

4. Review the information about the application up to the breakpoint in the Results panels.

   Flex Builder provides you with the following information:

   The Variables panel shows the names and values of all variables up to that point. For simple data types, you can edit a value and your change will take effect immediately on the next code step. Curly brackets ({}) denote complex data types.

   Tip: You can add a listed variable to the Watch panel by right-clicking it and selecting Add to Watch Panel from the context menu.

   The Watch panel lets you monitor the value of variables you specify. To specify a variable to watch, click in the Name column of the panel and enter the variable's name. You can also specify a variable by right-clicking a variable in the Variables panel and selecting Add to Watch Panel from the context menu.

The Call Stack panel 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 jump to it in the script; Flex Builder updates the information in the Variables panel to reflect the new location in the script.

The Output panel displays runtime errors, warnings, and traces. For more information, select Help from the context menu in the panel.

5. If you want to set breakpoints in any other file used by your application, such as custom classes and internal Flex classes, do the following:
   - Click the Files button on the debugging toolbar and select the internal file from the list.
   - After the internal file opens in Code view, set one or more breakpoints in the file.
   - Restart the debugger.
6. If you want to return to Code view to fix a bug, click the Stop button in the debugging toolbar.

   Because Code view is read-only during debugging, you must stop the debugger to fix a bug.
7. To continue stepping through the code, do any of the following:
   - To step to the next breakpoint, click the Continue button in the debugging toolbar, or press Control+5.
   - To step into a function, click the Step In button on the debugging toolbar, or press Control+6.
   - To step over a function, click the Step Over button on the debugging toolbar, or press Control+7.
   - To step out of a function, click the Step Out button on the debugging toolbar, or press Control+8.

   Note: You can also use the Debug menu for all these tasks.

Debugging applications by monitoring interactions with the server

If your Flex application isn't working as expected, you can use the Network Monitor to look for clues to the problem. When enabled, the Network Monitor monitors and records all data transactions with the server and displays the information in the Results panel.

The Network Monitor records and displays the following information:

- Events relating to the MXML service tags, including the `<mx:WebService>`, `<mx:HTTPService>`, and `<mx:RemoteObject>` tags
- Transport layer information, including SOAP envelopes and AMF/FlashRemoting packets
- Interactions resulting from the use of the ActionScript XML object, such as XML.send() and XML.sendAndLoad()
- HTTP traffic

The Network Monitor displays data that comes from the server in its native ActionScript equivalent.

In general, the Network Monitor captures and stores all event data until you either quit the application or explicitly clear the data. The events are displayed in chronological order. You can also selectively filter out specific events.

This section contains the following topics:

- "Turning on the Network Monitor" on page 94
- "Monitoring interactions with the server" on page 95
- "Using trace statements with the Network Monitor" on page 96
- "Saving event data" on page 96

Turning on the Network Monitor

The Network Monitor inserts itself between the application and the server to gain complete access to the data passing between them. You must enable it before using it.

To enable the Network Monitor:

1. Make sure a Flex server is defined as the testing server.

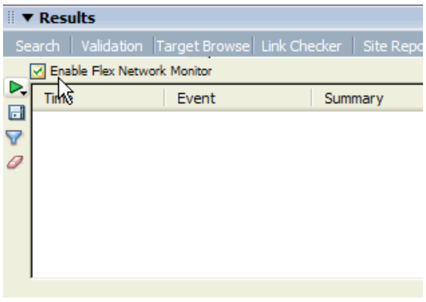
   For more information, see "Starting a new application in Flex Builder" on page 9.

2. In the Network Monitor panel in the Results panel group (Window > Results), click the Play button on the sidebar and select Settings.

   The Network Monitor Settings dialog box appears.

3. In the Network Monitor Settings dialog box, accept the default port values or specify the local communication ports that the Network Monitor should listen in on.

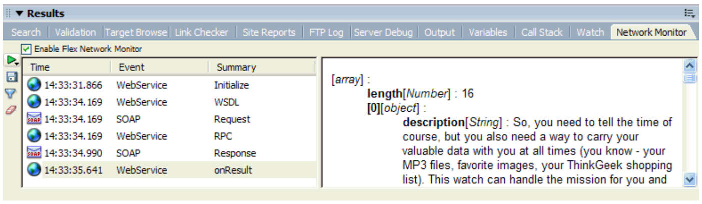
4. In the Network Monitor panel, click the Enable Flex Network Monitor option.



Monitoring interactions with the server

Once enabled, the Network Monitor automatically monitors and records interactions with the server. In Flex Builder, these interactions occur when you run or debug a file.

Captured events appear in the Network Monitor panel in the Results panel group (Window > Results). The left side of the panel displays all the events while the right side displays specific information for the selected event.



Time	Event	Summary
14:33:31.866	WebService	Initialize
14:33:34.169	WebService	WSDL
14:33:34.169	SOAP	Request
14:33:34.169	WebService	RPC
14:33:34.990	SOAP	Response
14:33:35.641	WebService	onResult

```
[array] :
    length[Number] : 16
    [0][object] :
        description[String] : So, you need to tell the time of
        course, but you also need a way to carry your
        valuable data with you at all times (you know - your
        MP3 files, favorite images, your ThinkGeek shopping
        list). This watch can handle the mission for you and
```

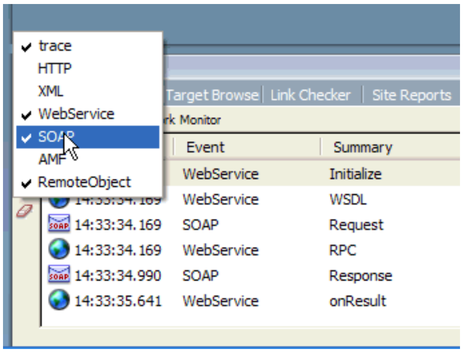
To view details about an event:

- Click an event in the list.

  The details (right) side of the panel supports the standard context menu and shortcut keys for selecting and copying text.

To filter out events:

- Click the Filter Settings button in the panel's sidebar and deselect event types from the pop-up menu.



Event	Summary
WebService	Initialize
WebService	WSDL
SOAP	Request
WebService	RPC
SOAP	Response
WebService	onResult

The types of events that you deselected are no longer displayed in the panel. To view them again, select them in the Filter Settings pop-up menu.

To clear all the event data in the panel:

- Click the Clear Log Data button in the panel's sidebar.

Using trace statements with the Network Monitor

You can insert trace statements in your ActionScript code and use the Network Monitor to track them at runtime.

The syntax for Network Monitor trace statements is as follows:

```
NetworkDebugger.NetworkMonitor.trace("string");
```

or

```
NetworkDebugger.NetworkMonitor.trace(var);
```

You cannot use a concatenation expression such as the following:

```
NetworkDebugger.NetworkMonitor.trace("hello" + myVar);
```

After you insert the trace statement in your ActionScript, the trace statement should appear as a trace event in the Network Debugger when the application calls the server—as long as your code doesn't have logic or some bug that prevents the section of code containing the trace statement from executing.

Note: You can also use plain ActionScript trace statements and view them in the Output panel after setting breakpoints in the file and starting the debugger. The syntax of these trace statements differs from Network Monitor trace statements. For more information, right-click the tab of the Output panel and select Help from the context menu.

Saving event data

You can save the event data generated by a particular interaction for analysis or printing.

To save the event data:

- Click the Save button (the diskette) in the panel's sidebar and save the file to a location on your hard disk.

  The event data is stored in an XML file.

CHAPTER 4
Building a Flex User Interface Visually

You can use Macromedia Flex Builder to build Flex user interfaces for multi-tier enterprise applications. MXML controls and containers are the building blocks of Flex user interfaces. You use containers to position the controls in your user interface.

This chapter contains the following topics:

- "Customizing the Document window" on page 98
- "Laying out your user interface with MXML containers" on page 99
- "Adding Flex components to the user interface" on page 100
- "Repositioning Flex components" on page 101
- "Resizing Flex components" on page 102
- "Cutting, copying, and pasting Flex components" on page 102
- "Editing Flex component properties" on page 103
- "Applying CSS styles to Flex components" on page 104
- "Applying effects to Flex components" on page 111
- "Assigning event handlers to components" on page 116

Related topics

- "Tutorial: Creating a layout with Flex Builder" on page 26
- "About Flex containers, components, and controls" on page 150
- Chapter 5, "Working with Components," on page 119
- Chapter 6, "Working with Data," on page 129

Customizing the Document window

You can customize the Document window in Flex Builder to suit the job at hand or your personal preferences.

To work in Design view, do one of the following:

- Click the Design button on the Document toolbar.
- Select View > Design.
- From Code view, press Control+' (the slanted quotation mark that usually shares the tilde [~] key).

You can split the view to work in the visual and coding environments at the same time.

To split the view, do one of the following:

- Click the Split button on the Document toolbar.
- Select View > Code and Design.

Flex Builder provides two modes for Design view: Standard and Expanded. In the default Expanded mode, Flex Builder adds a 10-pixel padding and a border around containers and controls to help you select, move, and resize them with the mouse. The padding and borders are not visible at runtime. You can remove the padding and border at design time by switching to Standard mode.

Tip: If you prefer working in code, combine Standard mode with Split view (click the Split button on the Document toolbar) to get quick visual feedback about the edits you make without compiling the file. For more information, see "Viewing and editing code in a file" on page 78.

To remove the padding and borders around containers and controls:

- Click the Standard button on the Document toolbar.

To restore the padding and borders around containers and controls:

- Click the Expanded button on the Document toolbar.

To remove the panels to maximize the window:

- Press F4.

To restore the panels:

- Press F4.

The following are some suggestions for customizing the Document window:

- If you're creating an MXML layout from scratch, try working in Design view in Expanded mode.

  This environment is ideal for the iterative process of adding controls to an MXML file and directly manipulating them in the layout. You can use the mouse to select objects from the Insert bar and position them in the file. You can also use the mouse to resize, reorder, and nest objects. You can double-click objects in the layout to rapidly edit their `title`, `text`, or `label` properties.

- If you're tweaking a layout, try working in Design view in Standard mode.

  This environment provides the highest possible fidelity for your layout (short of compiling and running the file). As such, the environment is well-suited for fine-tuning an MXML layout. You can make small modifications to the size and position of elements by using a combination of the mouse and the Tag inspector. You can select a nested component by clicking its tag in the tag selector at the bottom of the page.

Laying out your user interface with MXML containers

You can use MXML containers to lay out the user interface of your Flex application, and then insert controls in the containers.

To create a layout with MXML containers:

1. Create an MXML application file by selecting File > New and double-clicking MXML:Application in the New Document dialog box.

   Flex Builder creates a file that contains an `<mx:Application>` tag. This is the parent tag of all MXML applications. All MXML elements, controls, containers, and components ultimately reside in this tag.
2. If you want, change the default size of the Application container by clicking it and dragging a resize handle, or by clicking it and changing its width and height properties in the Attributes panel of Tag inspector (Window > Tag Inspector).
3. If you want, drag other MXML containers from the Insert bar into the Application container.

   Flex Builder inserts a graphical representation of each container vertically, starting in the upper left corner of the Application container. The Application container automatically arranges controls and containers in a vertical column.
4. If you want, remove the padding and border Flex Builder adds to containers by clicking the Standard button on the Document toolbar.

   By default, Flex Builder adds borders and padding around containers to help you manipulate them at design time. The padding and border are not visible at runtime. For more information, see "Customizing the Document window" on page 98.

The following are some useful containers and their layout management rules:

VBox automatically arranges components in a single column that stretches vertically to fit the components.

HBox automatically arranges components in a single row that stretches horizontally to fit the components.

Canvas supports absolute positioning. Similar to the Stage in Macromedia Flash, you can drag and position components anywhere in this container.

Accordion organizes information in a series of child panels, where one panel is active at any time.

Grid is similar to an HTML table tag. You can use the GridRow and GridItem containers together with the `rowSpan` and `colSpan` properties to specify the number of rows and grid cells.

For more information, see “Introducing Containers” and “Using Layout Containers” in Developing Flex Applications Help.

Related topics

- “Repositioning Flex components” on page 101
- “Resizing Flex components” on page 102
- “Cutting, copying, and pasting Flex components” on page 102

Adding Flex components to the user interface

You can visually add MXML controls and Flash components to your user interface. Your ability to position them depends on the layout management rules of the container. For example, if you drag a control into a VBox container, the control will snap into a position above or below the other controls in the container. However, if you drag a control into a Canvas container, which has absolute positioning, you can position it anywhere in the container. For more information on a container’s layout management rules, see “Using Layout Containers” in Developing Flex Applications Help.

To add a component to your user interface:

1. Make sure the active file is an MXML file.

   Components can only be inserted in MXML files.

2. In Design view (View > Design View), drag the component from the Insert bar to a container in the file.

   The insertion point (the cursor) changes depending on the layout management rules of the container:

   - If you drag a component into a container with automatic positioning (such as an Application or Form container), a line appears to indicate where the component will be positioned.
   - If you drag a component into a Canvas container, which has absolute positioning, a cross-hair appears to indicate where the component will be positioned. Alignment guides appear as well to help you with positioning. When you release the mouse button, the component is inserted with its upper left corner positioned on the cross-hair.

You can also click inside the container first, and then select the component from the Insert bar or Insert menu. Flex Builder inserts the component at the insertion point in the container. If you can’t easily click inside a container to insert the control, click the Expanded button on the Document toolbar to add padding around the container. For more information, see “Customizing the Document window” on page 98.

Related topics

- “Inserting controls and containers in an MXML file” on page 119
- “Laying out your user interface with MXML containers” on page 99.
- “Repositioning Flex components” on page 101
- “Resizing Flex components” on page 102
- “Cutting, copying, and pasting Flex components” on page 102

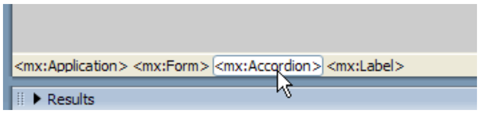
Repositioning Flex components

You can visually move MXML and Flash components from one container to another, or within a container.

For more information on a container's layout management rules, see "Using Layout Containers" in Developing Flex Applications Help.

To reposition components on a page:

1. Select the component as follows:
   - To select a container, click any of the following: the container's tab, its border, or any white space in it.
   - To select any other component, click it on the page or in the tag selector at the bottom of the Document window.

   - To select multiple components, drag to select a group of components, or Control-click each component to add it to the group.
2. Drag the component or components to a new position on the page.

   The insertion point changes depending on the layout management rules of the container:
   - If you drag a component into a container with automatic positioning (such as an Application or Tile container), a line appears to indicate where the component will be positioned.
   - If you drag a component into a Canvas container, which has absolute positioning, a cross-hair appears to indicate where the component will be positioned. When you click, the component is inserted with its upper left corner positioned on the cross-hair.

Related topics

- "Laying out your user interface with MXML containers" on page 99
- "Adding Flex components to the user interface" on page 100
- "Customizing the Document window" on page 98

Resizing Flex components

You can resize MXML containers and controls visually with your mouse.

Note: When you resize a component with this method, Flex Builder sets absolute width and height values. The component will not grow to accommodate its children, nor will it change size when the user resizes the application.

To resize an MXML component with the mouse:

1. Select the component by clicking it in the layout or on the tag selector at the bottom of the Document window.
2. Click a handle on the component to resize it.

   Holding the Shift key while resizing constrains the item to its current proportions.

   If you have trouble grabbing a handle, click the Expanded button on the Document toolbar to add padding around the component.
3. To rapidly clear the width or height properties of controls and containers, right-click the component and select Size to Content from the context menu.

   This command is useful if you want the container to resize automatically according to its contents.

Related topics

- "Laying out your user interface with MXML containers" on page 99
- "Adding Flex components to the user interface" on page 100
- "Editing Flex component properties" on page 103

Cutting, copying, and pasting Flex components

You can cut, copy, and paste components in Design view.

To cut, copy, and paste components in Design view:

1. Select one or more components on the page and cut or copy them using your preferred cutting or copying method (for example, Control+X or Control+C).

   You can use the keyboard shortcuts, the Edit menu, or right-click and use the context menu.

   You can select multiple components by Control-clicking them. You can also select one or more components by clicking outside the outermost container and dragging.

   If you have trouble selecting a component, click the Expanded button on the Document toolbar to add padding around objects on your page.
2. Paste the components elsewhere on the page or in another page using your preferred pasting method (for example, Control+V).

Related topics

- "Laying out your user interface with MXML containers" on page 99
- "Adding Flex components to the user interface" on page 100
- "Repositioning Flex components" on page 101
- "Resizing Flex components" on page 102

Editing Flex component properties

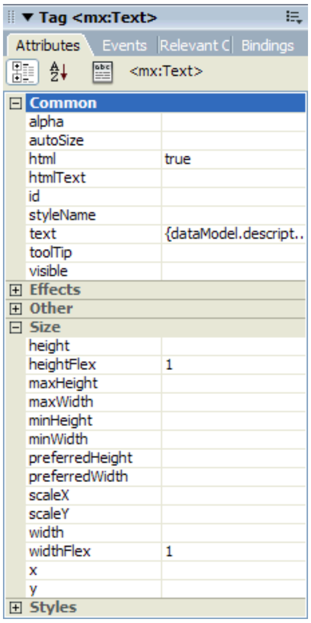
You can visually edit the properties and events of MXML or Flash components in one central location—the Attributes panel. In addition, you can use the Quick Tag Editor to make quick edits to MXML tags without leaving Design view.

Note: You cannot edit component properties after you start a debugging session. For more information, see "Debugging ActionScript" on page 90.

To edit component properties with the Attributes panel:

1. Click the component in the page.

   The properties of the component appear in the Attributes panel of the Tag inspector (Window > Tag Inspector).



▼ Tag <mx:Text>

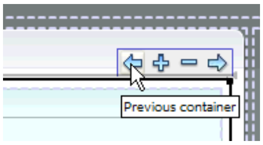
Attributes | Events | Relevant C | Bindings

<mx:Text>

⊟ Common	
alpha	
autoSize	
html	true
htmlText	
id	
styleName	
text	{dataModel.descript..
toolTip	
visible	
⊞ Effects	
⊞ Other	
⊟ Size	
height	
heightFlex	1
maxHeight	
maxWidth	
minHeight	
minWidth	
preferredHeight	
preferredWidth	
scaleX	
scaleY	
width	
widthFlex	1
x	
y	
⊞ Styles	

2. If the item you want to edit is in a navigator container such as an Accordion or TabNavigator, click the tab of the navigator component to move between panes.

3. If the item you want to edit is hidden in a ViewStack container, click the ViewStack container and then click the left or right arrows that appear at the top of the container to cycle through the items in the container.



   You can also right-click the container and select Next View or Previous View from the pop-up menu.

4. Set the properties of the component in the Attributes panel.

To edit the properties of a standard MXML component in the Quick Tag Editor:

1. Double-click the component in Design view.

   The Quick Tag Editor appears. Depending on the component, the `title`, `text`, or `label` value in the tag may be highlighted to help you work even faster.

   Note: If you double-click a custom component, the component's file opens in Flex Builder instead.

2. Change the property values in the Quick Tag Editor and then press Enter.

To edit the properties of a custom MXML component:

1. Double-click the component in Design view.

   Flex Builder opens the associated MXML file in a new Document window with its own Code and Design views.

   Note: If you double-click a standard MXML component, the Quick Tag Editor appears instead.

2. Set the properties of the component in the Attributes panel.

   For more information, see "Creating MXML component files" on page 123.

3. Save the file.

   After saving the component file, Flex Builder updates the component in the parent file's Design view.

Related topics

- "Applying CSS styles to Flex components" on page 104
- "Applying effects to Flex components" on page 111
- "Creating MXML component files" on page 123
- "Using the Tag inspector" on page 81

Applying CSS styles to Flex components

You can use CSS styles to define, maintain, and easily modify the general appearance of components.

This section contains the following topics:

- "About CSS styles in Flex" on page 105
- "Importing styles from an external CSS file" on page 106
- "Creating new CSS styles" on page 107
- "Applying a class style to a component" on page 108
- "Applying an inline style to a component" on page 109
- "Modifying a CSS style applied to a component" on page 109
- "Deleting a CSS style" on page 110

Related topics

- “Editing Flex component properties” on page 103
- “Applying effects to Flex components” on page 111
- “Creating MXML component files” on page 123

About CSS styles in Flex

You can use CSS styles to modify your Flex user interface more efficiently. As your design evolves, you can add, delete, or change styles, giving you greater freedom to achieve your design goals.

CSS styles in Flex can be defined at the site level with external styles, at the document level with embedded styles, or at the component level with inline styles.

External styles are defined in a separate file and can be used in any MXML file that references the CSS file. You reference a CSS file into an MXML file with the `source` property of the `<mx:Style>` tag, as follows:

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

Embedded styles are defined in an MXML file and can only be used in that file. Embedded styles are defined with the `<mx:Style>` tag, as follows:

```
<mx:Style>
  .myclass { background-color: xFF0000 }
  TextInput { font-family: Helvetica; font-size: 12pt }
</mx:Style>
```

The previous example shows the two types of selectors supported by Flex: class selectors and tag selectors. A class selector defines a new named style, such as myclass. Once defined, you can apply the myclass style to any component using the `styleName` property, as in the following example:

```
<mx:Canvas>
  <mx:Button styleName="myclass">
</mx:Canvas>
```

A tag selector applies to all components of a particular type. For example, the `<mx:Style>` tag above defines a tag selector that applies a background color to all TextInput components in the current file.

Inline styles are defined in an MXML tag and can only be used in that tag. Inline styles are defined as follows:

```
<mx:Button color="red"...>
```

Flex uses slightly different names for multi-word CSS property names such as font-family or font-size. Hyphens are removed, the letter following each hyphen is capitalized, and equal signs instead of colons are used for property values. For example, font-family and font-size become fontFamily and fontSize in Flex, as follows:

```
<mx:TextInput fontFamily="Helvetica" fontSize="12"...>
```

Related topics

- “Using Styles” in Developing Flex Applications Help

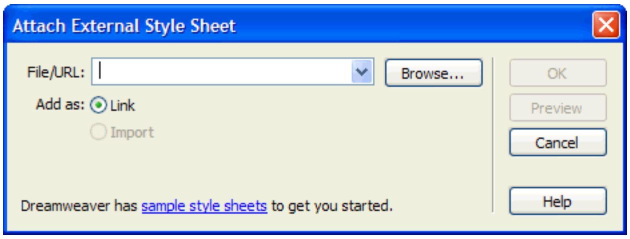
Importing styles from an external CSS file

You can import styles defined in an external CSS file by using the `source` property of the `<mx:Style>` tag.

To import styles from an external CSS file:

1. Make sure the CSS file is located in your site root folder or in any of its subfolders.
2. Open the CSS Styles panel (Window > CSS Styles) and click the Attach Style Sheet button at the bottom of the panel.

   The Attach External Style Sheet dialog box appears.

   Dreamweaver has sample style sheets to get you started.

3. Click Browse and browse to the external CSS style sheet.
4. Click OK.

The name of the external CSS style sheet appears in the CSS Styles panel.

Tip: If you hand-code a reference to an external style sheet in an MXML component file, you must specify the path relative to the application in the source property of the <mx:Style> tag.

Related topics

- "About CSS styles in Flex" on page 105

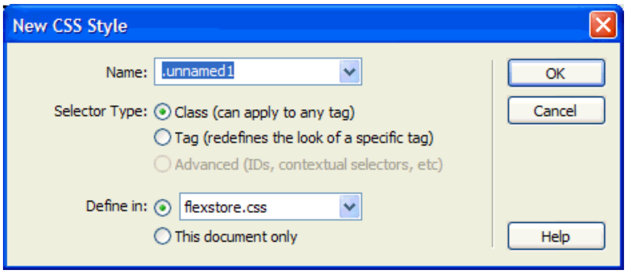
Creating new CSS styles

You can visually create new CSS styles for Flex components. You can define them in an external CSS file or in the MXML file containing the components you want to format.

To create new styles:

1. In the CSS Styles panel (Window > CSS Styles), click the New CSS Style button (+) located in the lower right area of the panel.

   The New CSS Style dialog box appears.

   New CSS Style

   Name: .unnamed1

   Selector Type: [✓ Class (can apply to any tag)] Tag (redefines the look of a specific tag) Advanced (IDs, contextual selectors, etc)

   Define in: [✓ flexstore.css] This document only

2. Select a selector type.

   Flex supports class selectors and tag selectors. A class selector defines a new named style, such as myclass, that you can apply to any component.

   A tag selector applies to all components of a particular type. For example, if you select the TextInput component and set the background color to blue, a blue background is automatically applied to all TextInput objects.

3. Select the location where you want to define the new style as follows:
   - To define the style in a new external style sheet, select New Style Sheet File.
   - To define the style in an existing style sheet attached to the current file, select the CSS file from the Define In pop-up menu.
   - To embed the style in the current file, select This Document Only.

4. Click OK.

   The New CSS Style dialog box closes and the Style Definition dialog box opens.

5. Use the dialog box options to define the new CSS style.

6. When you're finished, click OK.

7. If you created the style in an external style sheet, make sure to upload the style sheet to the server to ensure your changes are reflected when you compile and run the file.

Related topics

- "Creating a new CSS style" in Using Dreamweaver Help
- "About CSS styles in Flex" on page 105
- "Importing styles from an external CSS file" on page 106

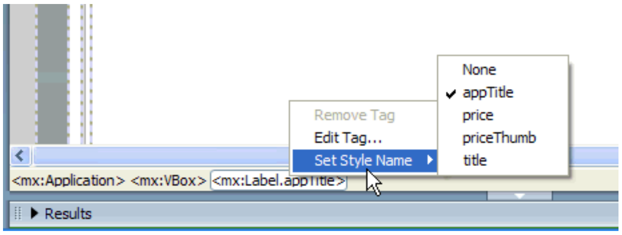
Applying a class style to a component

You can visually apply internal or external CSS styles to one or more components.

Styles defined as class selectors are the only type of styles you can apply. You can't apply styles defined as tag selectors because these styles are automatically applied to each instance of the specified tag.

To apply a class style to one component:

- Select the component in Design view and do one of the following:
  - Right-click the component in Design view and select the style from the CSS Styles menu in the context menu.
  - In the CSS Styles panel (Window > CSS Styles), right-click the style you want to apply and select Apply from the context menu.
  - In the tag selector at the bottom of the Document window, right-click the tag of the component (it should be highlighted) and select the class from the Set Style Name menu.



To apply a class style to several components at the same time:

1. Select the components in Design view by Control-clicking each one of them.
2. Right-click any of the selected components and select the style from the CSS Styles menu in the context menu.

To remove a class style from a component:

- Select the component in Design view and do one of the following:
  - Right-click the component and select CSS Styles > None from the context menu.
  - In the tag selector at the bottom of the Document window, right-click the tag of the component (it should be highlighted) and select Set Style Name > None from the context menu.
  - In the Tag inspector (Window > Tag Inspector), click the Relevant CSS tab, right-click the applied rule you want to remove, and select Set Style Name > None from the context menu.

Related topics

- "About CSS styles in Flex" on page 105

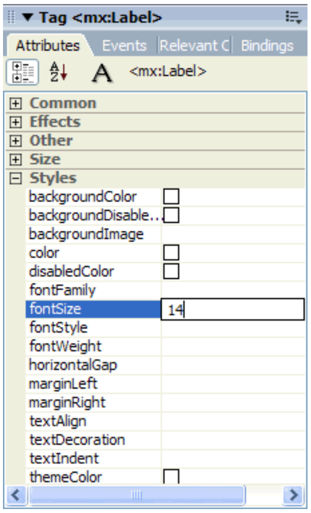
Applying an inline style to a component

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

To apply an inline style to a component:

1. Click the component in the page.
2. In the Tag inspector (Window > Tag Inspector), click the Attributes tab and scroll to the Styles category of properties.

   The Styles category lists all the styles that can be applied to the selected component.



backgroundColor	☐
backgroundDisable...	☐
backgroundImage	
color	☐
disabledColor	☐
fontFamily	
fontSize	14
fontStyle	
fontWeight	
horizontalGap	
marginLeft	
marginRight	
textAlign	
textDecoration	
textIndent	
themeColor	☐

   Note: Multiword style names in Flex are slightly different from their HTML counterparts, but they're the same styles. For example, fontFamily in Flex is the same style as font-family in HTML.

3. In the Styles category, set the style properties of the component.

Related topics

- "About CSS styles in Flex" on page 105
- "Applying a class style to a component" on page 108

Modifying a CSS style applied to a component

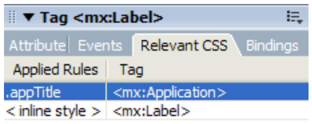
You can visually modify any CSS style applied to a component.

To modify a CSS style applied to a component:

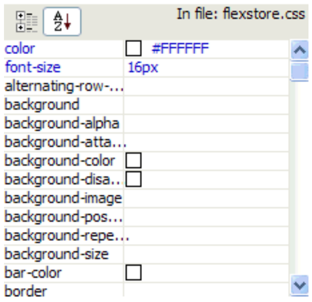
1. Select the component with the style.
2. In the Tag inspector (Window > Tag Inspector), click the Relevant CSS tab to view the CSS styles in the current file.

3. Select the applied rule from the left column in the top half of the panel.

   The bottom half of the panel displays the style properties for the applied rule.



Applied Rules	Tag
.appTitle	<mx:Application>
< inline style >	<mx:Label>


In file: flexstore.css

color	☐ #FFFFFF
font-size	16px
alternating-row-...	
background	
background-alpha	
background-atta...	
background-color	☐
background-disa..	☐
background-image	
background-pos...	
background-repe...	
background-size	
bar-color	☐
border	

4. Modify the properties in the bottom half of the panel.

   Press Enter to accept a modified property value and to see how the change affects the component in Design view.

   Note: If you modify CSS styles in a style sheet used by more than one file, those changes will be reflected in those files as well.

5. When you're finished editing the style, save your work.

6. If you modified a style in an external style sheet, make sure to save the file and upload it to the server to ensure your changes are reflected when you compile and run the file.

Related topics

- "Using the Relevant CSS tab" in Using Dreamweaver Help

Deleting a CSS style

You can visually delete CSS styles from an MXML file or from an external CSS file.

Note: Deleting a style removes the formatting from any element using the style.

To delete a style:

1. Select the style in the CSS Styles panel (Window > CSS Styles).
2. Do one of the following:
   - Click the Delete CSS Style button (the trash can icon) at the bottom of the panel.
   - Right-click the style in the panel and select Delete from the context menu.

Applying effects to Flex components

You can add effects to components. For example, you can cause a TextInput component to bounce slightly when it receives focus, or a Label component to fade in when the user passes the mouse over it. This section describes how to add these kinds of effects.

This section covers the following topics:

- "About Flex effects" on page 111
- "Adding an effect to a component" on page 111
- "Customizing a standard Flex effect" on page 114
- "Modifying a custom effect" on page 115

About Flex effects

An effect is a visible or audible change to the component occurring over a period of time, measured in milliseconds. Examples of effects are fading, resizing, or moving a component.

Effects are paired with triggers. A trigger is an action such as a mouse click, a component getting focus, or a component becoming visible. Flex uses CSS to define triggers that you reference as a property in an MXML tag or in an ActionScript function. The CSS property names of triggers use the following convention:

```
triggerEffect
```

where *trigger* is the trigger name. For example, the show trigger occurs when a component becomes visible; the CSS property name for the show trigger is showEffect. The hide trigger occurs when a component becomes invisible; its CSS property name is hideEffect.

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

Related topics

- "Using Behaviors" in Developing Flex Applications Help
- "Customizing a standard Flex effect" on page 114
- "Modifying a custom effect" on page 115

Adding an effect to a component

You can use Flex Builder to visually add an effect to an MXML component.

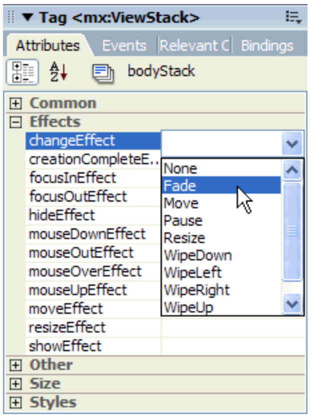
Note: Though you cannot visually add an effect to an ActionScript component, you can use Code view to write the code that adds the effect. For more information, see "Applying an effect in ActionScript" in Developing Flex Applications Help.

To add an effect to an MXML component:

1. Select the component in Design view by clicking it on the page or on the tag selector at the bottom of the Document window.
2. Define a trigger for the effect by selecting a CSS property name for the trigger in the Attributes panel of the Tag inspector (Window > Tag Inspector).

   For example, to define a mouse-over trigger, select the `mouseOverEffect` property.
3. Associate an effect with the trigger.

   In the field next to the property name for the trigger, select an effect from the pop-up menu.



   For example, for a Label component you could select WipeRight as the effect for the `showEffect` property. In a browser, the Label's text will become visible as if it were being "wiped" from left to right.
4. If you select showEffect, hideEffect, moveEffect, or resizeEffect, and you specified an ID for the component, a dialog box appears asking if you'd like help setting a trigger for this effect.

   If you want the current component to trigger the effect, click No. If you want another component to trigger the effect, click Yes. A dialog box appears to help you write and insert the ActionScript function for triggering the effect from the other component. For more information, see "Triggering effects from other components" on page 112.

Related topics

- "About Flex effects" on page 111
- "Customizing a standard Flex effect" on page 114
- "Modifying a custom effect" on page 115

Triggering effects from other components

An effect can be triggered from another component. For example, a Label component can have a WipeLeft effect that is triggered when the user clicks on a Button component.

Flex Builder can help you set up these interactive triggers for the following effect properties: showEffect, hideEffect, moveEffect, and resizeEffect.

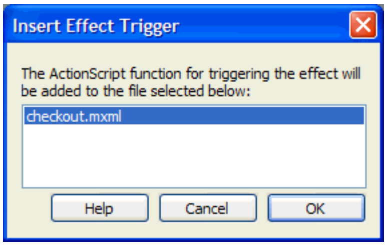
To trigger effects from other components:

1. In Design view, select the component that will have the effect.
2. In the Attributes panel (Window > Tag Inspector), make sure an ID value is specified for the component.

   If the component doesn't have an ID, you can't trigger the effect from another component.
3. Select an effect for any of the following effect properties in the Attributes panel: showEffect, hideEffect, moveEffect, or resizeEffect.

   A dialog box appears asking if you'd like help setting a trigger for this effect. Click Yes.

   The Insert Trigger Effect dialog box appears listing the current file and any files included from the current file.

   The ActionScript function for triggering the effect will be added to the file selected below:

   checkout.mxml
4. Select the file in which you want to insert the ActionScript function for triggering the effect.

   Flex Builder inserts an ActionScript function in the specified file. The function is designed to trigger the event when an event occurs in another component in the file.
5. In Code view, find the component that will trigger the effect and specify the newly inserted function as an event handler.

   For example, if you apply a WipeLeft hideEffect to a Label called lblHello, Flex Builder inserts the following ActionScript function:

   ```
   function triggerHideEffectlblHello()
   {
     // if lblHello is visible, this triggers the hideEffect on lblHello.
     lblHello.visible = false;
   }
   ```

   If you want the effect to be triggered when users click a certain button, locate the Button tag in the code and specify the function name (including the parentheses) as its `click` property, as follows:

   ```
   <mx:Button id="triggerButton" click="triggerHideEffectlblHello()" />
   ```

   For more information, see "Applying an effect in ActionScript" in Developing Flex Applications Help.

Related topics

- "Adding an effect to a component" on page 111

Customizing a standard Flex effect

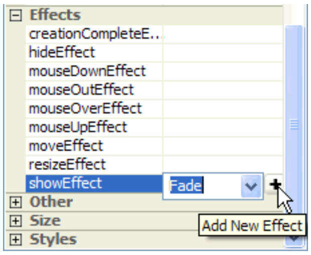
The standard effects in Flex may not provide the exact effect that you want. A Fade effect might be too fast or too slow, or a Pause effect might not hold long enough. You can use Flex Builder to help you customize these effects.

Note: This feature works only with standard Flex effects applied to MXML components. It does not work with effects applied to ActionScript components or with custom effects.

To customize a standard Flex effect:

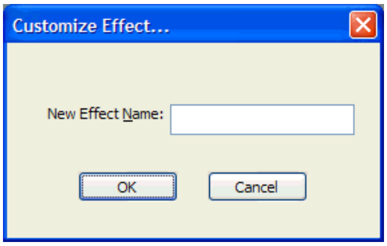
1. In Design view, select the MXML component with the effect you want to customize.
2. In the Attributes panel of the Tag inspector (Window > Tag Inspector), click the effect.

   A Plus (+) icon appears beside the effect name.



3. Click the Plus (+) icon beside the effect name.

   The Customize Effect dialog box appears.



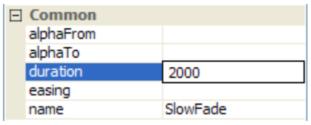
4. Enter a name for the new effect and click OK.

   Flex Builder performs the following tasks:

   - Switches to Code view and adds an `<mx:Effect>` tag containing the tag for the custom effect.
   - Changes the name of the original effect in the component tag to match the name of the custom effect.
   - Shows the properties of the custom effect in the Tag inspector.

5. Set the properties of the custom effect tag in the Tag inspector or in the code.

   The following example creates a new version of the Fade effect called SlowFade, which uses a two-second duration expressed in milliseconds:



Common	
alphaFrom	
alphaTo	
duration	2000
easing	
name	SlowFade

   Flex Builder inserts the following code in your file:

```
<mx:Effect>
  <mx:Fade name="SlowFade" duration="2000"/>
</mx:Effect>
```

   For more information, see "Customizing an effect" in Developing Flex Applications Help.

6. Save the file.
7. If you want, apply the custom effect to other components in the file.

   Flex Builder automatically applies the custom effect to the original component, but you can also apply it to other components in the file by selecting it from the list of effects in the Attributes panel. Custom effects are listed in the panel when they're defined in the current file.

Related topics

- "About Flex effects" on page 111
- "Adding an effect to a component" on page 111

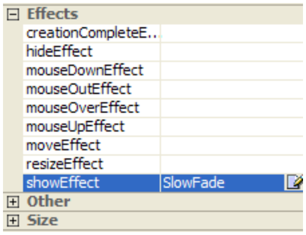
Modifying a custom effect

If you defined a custom effect in the current file, you can use Flex Builder to quickly modify its properties.

To modify a custom effect:

1. In Design view, select an MXML component with the custom effect you want to modify.
2. In the Attributes panel of the Tag inspector (Window > Tag Inspector), click the custom effect.

   A pencil icon appears beside the effect name.



Effects	
creationCompleteE..	
hideEffect	
mouseDownEffect	
mouseOutEffect	
mouseOverEffect	
mouseUpEffect	
moveEffect	
resizeEffect	
showEffect	SlowFade
Other	
Size	

3. Click the pencil icon beside the effect name.

   Flex Builder performs the following tasks:

   - Switches to Split view and selects the custom effect tag in the code
   - Shows the properties of the custom effect in the Tag inspector.

4. Set the properties of the custom effect tag in the Attributes panel or in the code.

   For more information, see "Customizing an effect" in Developing Flex Applications Help.

5. Save the file.

Related topics

- "Customizing a standard Flex effect" on page 114
- "About Flex effects" on page 111
- "Adding an effect to a component" on page 111

Assigning event handlers to components

Flex applications are event-driven. An event is generated when the user interacts with the user interface (with the mouse, for example), or when the application interacts with data. You can use Flex Builder to specify an event handler to execute when a specific event occurs. Event handlers include object methods such as ws.method.send(), and ActionScript functions.

To assign an event handler to a component:

1. If you want to use a custom event handler, define the event handler using ActionScript.

   You can write the ActionScript directly in the MXML file or in an external ActionScript file that you include in the MXML file.

   If the code for the event handler is short, you can include it as a property value in the component tag. See step 3.

2. Select the component in the MXML file to call the event handler.

   For example, a Button component is commonly assigned a handler for the click event.

3. In the Events panel of the Tag inspector, select the event you want and enter one of the following in the value field:

   - An object method you want to run in response to the event.
   - A function call to the ActionScript event handler you defined or referenced elsewhere in the MXML file.
   - An ActionScript statement or group of statements separated by semi-colons that can handle the event.

4. If you want the event to invoke a Flex data service such as a web service, click the lightning bolt icon beside the event's value field and then select the data service and the method that invokes it in the dialog box that appears.

   For more information, see "Calling the data service" on page 143. You must insert a Flex data service in the MXML file before you can select it in this dialog box. For more information, see "Working with Flex data services" on page 138.

Related topics

- “Editing Flex component properties” on page 103
- “Applying CSS styles to Flex components” on page 104
- “Handling events” in Developing Flex Applications Help

CHAPTER 5
Working with Components

An application typically consists of an MXML application file (a file with an `<mx:Application>` parent tag) and one or more components defined in separate MXML, ActionScript, or Macromedia Flash component (SWC) files. You lay out the components in the application file using Macromedia Flex containers.

You can use Macromedia Flex Builder to build and extend components for your applications. Using components promotes code reuse, simplifies the process of building a complex application, and lets other developers contribute to your project.

Flex Builder also lets you work with Flash components and custom Flex components installed on the Flex server.

This chapter contains the following topics:

- "Inserting controls and containers in an MXML file" on page 119
- "Creating MXML component files" on page 123
- "Synchronizing with custom components on the server" on page 127

Inserting controls and containers in an MXML file

You can visually insert MXML controls and containers in an MXML file. If you prefer to insert them manually, make sure you specify the component namespace.

This section contains the following topics:

- "Visually inserting components" on page 120
- "Manually inserting components" on page 121
- "Specifying component namespaces in your code" on page 121

Related topics

- "About Flex containers, components, and controls" on page 150

Visually inserting components

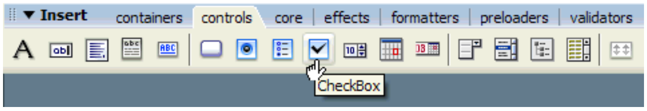
You can visually insert MXML components in MXML files. Flex Builder writes the appropriate code—including the correct namespace—to reference the component. The techniques described in this section work in both Design and Code views.

To visually insert a component:

1. Open the parent MXML file in Code view (View > Code) or Design view (View > Design).
2. Click inside an MXML container in the file.

   Always insert components in MXML containers to lay out the components in your file. If your file does not have any containers, select one from the Containers category of the Insert bar or from the Insert menu. For more information, see "Laying out your user interface with MXML containers" on page 99.
3. Do one of the following:
   - Click the component in the Insert bar.
   - Drag the component from the Insert bar to the page.
   - Select the component from the Insert menu.

   To find a component or container in the Insert bar, position the mouse pointer over a button and wait for a tooltip to appear.

   Note: Custom components won't be visible on the Insert bar until you synchronize Flex Builder with the server. For more information, see "Synchronizing with custom components on the server" on page 127.
4. If prompted by a dialog box, set or clear basic property values for the component.

   A dialog box appears when you click certain components on the Insert bar. To bypass these dialog boxes, Control-click the components on the Insert bar.
5. If the component is a custom ActionScript or MXML component located in your site's local root folder or in any of its subfolders, select Insert > Local Component and specify the location of the component in the Insert Local Component dialog box.

   For more information, click the Help button on the dialog box.

   Note: ActionScript components won't render in Design view, but the code will be inserted in the file.
6. Save the file.

Related topics

- Chapter 4, "Building a Flex User Interface Visually," on page 97

Manually inserting components

If you prefer to insert components by hand in Code view, make sure you specify the component namespace.

To manually insert a component:

1. Open the parent MXML file in Code view (View > Code) and enter the component tag.

   For standard Flex components, you can use code hints to work faster. You can also get code hints for custom components if you package them locally or synchronize them from the server. For more information, see "Including the component's properties in code hints" on page 126 and "Synchronizing with custom components on the server" on page 127.

   For custom components, the tag name and case must be identical to the name and case of the component's MXML file (see "Creating MXML component files" on page 123). For example, if the component's MXML file is called LoginForm.mxml, then the component tag's name and case is `<LoginForm...>`. If you enter `<loginForm...>`, the compiler will generate an error.

   The following is an example of a custom component tag:

   ```
   <xyz:LoginForm/>
   ```

   where `xyz` is the tag prefix used to reference components in a specified namespace (for more information, see the next step).

2. Specify the namespace of the component.

   Namespaces describe the location of component files on the Flex server. You must specify the namespace so that your application knows where to look for the file. For more information, see "Specifying component namespaces in your code" on page 121.

   Flex Builder will automatically specify the namespace if you package the component locally or synchronize it from the server.

3. Save the file.

Specifying component namespaces in your code

If you insert a component in an MXML file manually, you need to specify the component's namespace. Namespaces describe the location of component files on the Flex server so that your application knows where to look for the files.

You specify the namespace with the `xmlns` (for "XML namespace") property of either the `<mx:Application>` tag or the custom component tag.

To specify component namespaces in your code:

- Depending on the type and location of the component you want to insert, specify the namespace as follows:

  Standard Macromedia controls and containers If working with a standard Macromedia control or container, you don't need to specify a namespace. The namespace for all Macromedia components is http://www.macromedia.com/2003/mxml/ and is specified in the `<mx:Application>` tag.

  For example, if you want to insert a ComboBox control, which is a Macromedia component, enter the ComboBox tag between the opening and closing `<mx:Application>` tags, as follows:

  ```
  <mx:Application xmlns:mx="http://www.macromedia.com/2003/mxml">
    <mx:ComboBox/>
  </mx:Application>
  ```

  You can omit the `mx` prefix from the component tag if you don't specify it in the `xmlns` property name (you use `xmlns="..."` instead of `xmlns:mx="..."`). The following example is identical to the previous one, except that the prefix is omitted from the component tag:

  ```
  <Application xmlns="http://www.macromedia.com/2003/mxml">
    <ComboBox/>
  <Application>
  ```

  Custom MXML and ActionScript components If you want to insert a custom MXML or ActionScript component, the namespace is defined by the location of the component file in the application's root folder on the server.

  For example, if you write a custom MXML component called myButton.mxml and save it on the Flex server in the assets/components/mycontrols subfolder in your application root folder, enter the following namespace for the component:

  ```
  xmlns:xyz="assets.components.mycontrols.*"
  ```

  The property suffix `:xyz` tells you what tag prefix to use to reference the components in the specified namespace. If the suffix is `:xyz`, then use the `xyz:` tag prefix in your code to reference a component in the namespace, as follows:

  ```
  <xyz:myButton xmlns:xyz="assets.components.mycontrols.*" />
  ```

  If the component file is located in the application root folder itself, the namespace is * (asterisk), as follows:

  ```
  ... xmlns:xyz="*"
  ```

  The namespace is also * (asterisk) if the component file is located in the /WEB-INF/flex/user_classes subfolder in your application's root folder on the server. For example, if you place the myButton.mxml component file in the /WEB-INF/flex/user_classes folder on the server, enter the following tag to insert the component into an MXML file:

  ```
  <xyz:myButton xmlns:xyz="*" />
  ```

  Note: If two components have the same name and one is located in the application root folder and the other is located in the WEB-INF folder, the component in the application root folder is used first.

If the namespace contains several components that you want to insert in the file or if you want to insert several instances of the same custom component in the file, you can specify the namespace once in the Application tag and omit it from each component tag, as follows:

```
<mx:Application xmlns:mx="http://www.macromedia.com/2003/mxml"
  xmlns:xyz="assets.components.controls.*">
  <xyz:myComboBox />
  <xyz:myButton />
</mx:Application>
```

Flash components (SWCs) If you want to insert a Flash component, the namespace is specified by the component's class package. For example, suppose you want to insert a Flash component called foo.swc that belongs to the my.comps.foo class. The namespace is as follows:

xmlns:xyz="my.comps.*"

For more information, see "About SWC files" in Developing Flex Applications Help.

Related topics

- "Defining component namespaces" in Developing Flex Applications Help

Creating MXML component files

You can use Flex Builder to create MXML component files.

This section covers the following topics:

- "Creating an MXML component file" on page 124
- "Creating and immediately using a component" on page 125
- "Creating a component from an existing component" on page 126
- "Including the component's properties in code hints" on page 126

Related topics

- "Inserting controls and containers in an MXML file" on page 119

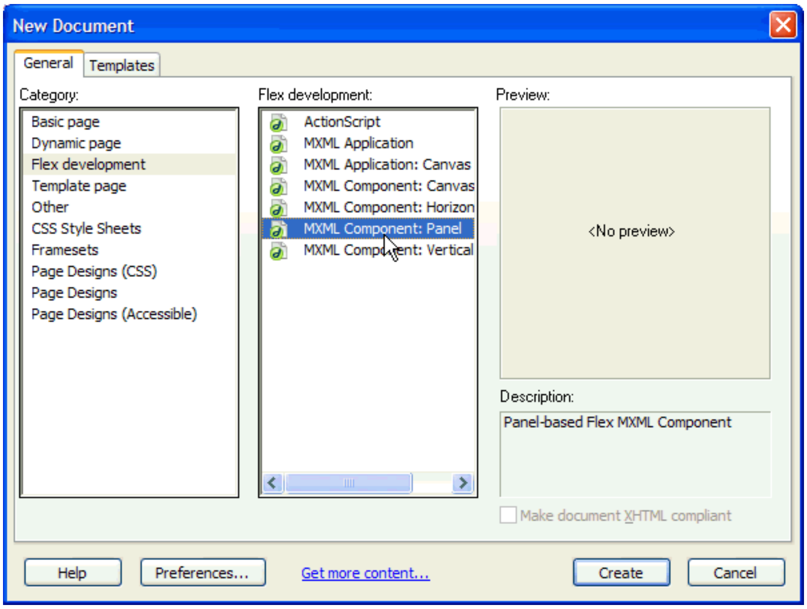
Creating an MXML component file

You create an MXML component file in the same way that you create any new file.

To create a component file:

1. Select File > New to open the New document dialog box, and then double-click one of the component options in the Flex Development category.

   Flex Builder gives you the option of creating components derived from common container classes.

   Panel-based Flex MXML Component

2. Name the component by selecting File > Save As and naming the file.

   The filename defines the component name. For example, if you name the file LoginForm.mxml, the component is named LoginForm. The Flex server uses the spelling and capitalization of the filename to generate a new class representing the component.

3. Save the file in the local root folder of your application or in any of its subfolders.

   Flex applications can't find components located outside of the application root folder. For more information on the application root folder, see "Identifying a Flex application root folder" on page 11.

4. Write the code for the component.

   For more information, see "Creating MXML components" in Developing Flex Applications Help.

5. Save the component file.

6. If the Flex server is running on a remote computer, copy the component file to the application folder on the server.

   Make sure the application folder structure on the server matches the folder structure on your local computer, and that you copy the file to the same folder on the server. The folder structure on the server determines the component's namespace. For more information, see "Specifying component namespaces in your code" on page 121.

   If you specified a remote folder when you defined your site in Flex Builder, you can use the Put command (Site > Put) to upload the file to the server. The Put command recreates the local folder structure on the server if it doesn't exist.

   If you want to share the component across applications and teams, you can move the file to the /WEB-INF/flex/user_classes folder. The /WEB-INF/flex/user_classes folder is created in your site's root folder on the server the first time you compile any part of your application. Synchronize Flex Builder with the server (Site > Update Flex Components) so that you can use the Insert bar to insert the components in your files or the Tag inspector to edit their properties. For more information, see "Synchronizing with custom components on the server" on page 127.

Creating and immediately using a component

You can create a custom component file and immediately insert it in an MXML file.

To create a new component file and immediately insert it in an MXML file:

1. Open the parent MXML file and place the insertion point where you want to insert the new component.

   If you're using a newly created MXML file, make sure to save it before completing the next step.

2. Select Insert > New Local Component.

   The Insert New Local Component dialog box appears.

3. Complete the dialog box and click OK.

   For more information, click the Help button on the dialog box.

   Flex Builder inserts a tag at the insertion point that references the new custom component. At the same time, it creates a new component file in the tabbed Document window.

4. Click the new component's file tab in the Document window and write the code for the component.

5. Save the component file.

Creating a component from an existing component

You can create a new component from any component in an existing MXML file. Flex Builder creates the new component file based on the component you select.

To create a component file from a component in an existing MXML file:

1. In Flex Builder, open an MXML file containing the component you want to use.
2. Do one of the following:
   - If working in Design view, select and right-click the standard component you want as the basis of your new custom component and then select Create Local Component from the context menu.
   - If working in Code view, select the code for a component you want as the basis of your new custom component, right-click the selected code, and then select Create Local Component from the context menu.

   The Save Component File dialog box appears.
3. Name and save the component file.
4. Insert your new component by selecting Insert > Local Component and selecting the component file.

   Flex Builder will not replace the code you selected in Step 2. If you want, delete the code after inserting the new component.
5. If the Flex server is running on a remote computer, copy the component file to the application folder on the server.

   Make sure the application folder structure on the server matches the folder structure on your local computer, and that you copy the file to the same folder on the server. If you specified a remote folder when you defined your site in Flex Builder, you can use the Put command (Site > Put) to upload the file to the server. The Put command recreates the local folder structure on the server if it doesn't exist.

   If you move the component file to another folder on the server, such as the /WEB-INF/flex/user_classes folder, then the namespace Flex Builder wrote in your MXML file might not be valid anymore. Modify the namespace as necessary by modifying the `xmlns` property of the component tag. For more information, see "Specifying component namespaces in your code" on page 121.

Including the component's properties in code hints

Code hints in Flex Builder let you write MXML or ActionScript more rapidly. You can enhance this feature by including the properties of your custom components in the code hints.

For MXML component files, Flex Builder will include properties declared using MXML syntax. For ActionScript component files, Flex Builder will include only properties that are declared with an Inspectable statement. The Inspectable statement must immediately precede the property's variable declaration. The syntax for the Inspectable statement is as follows:

```
[Inspectable(attribute=value[,attribute=value,...])]
property_declaration name:type;
```

For more information, see “Component metadata” in Developing Flex Applications Help.

To include the properties of custom components in code hints:

1. Open the MXML or ActionScript component file in Flex Builder.
2. Select File > Package Component.

Synchronizing with custom components on the server

The Flex Builder authoring environment has a default set of Flex components. If you install new or updated custom components on your Flex server, Flex Builder can synchronize its authoring environment with them so that you can work with them. For example, you can use the Insert bar to insert a custom component in your page and then use the Tag inspector to edit its properties.

Aside from updating the authoring environment, synchronizing with updated components on the Flex server is important when developing Flex applications. If the authoring environment supports a component that’s out of date, you might inadvertently create invalid MXML. For example, you might insert a component on your page and edit a property that no longer exists.

To synchronize with new or updated components on the server:

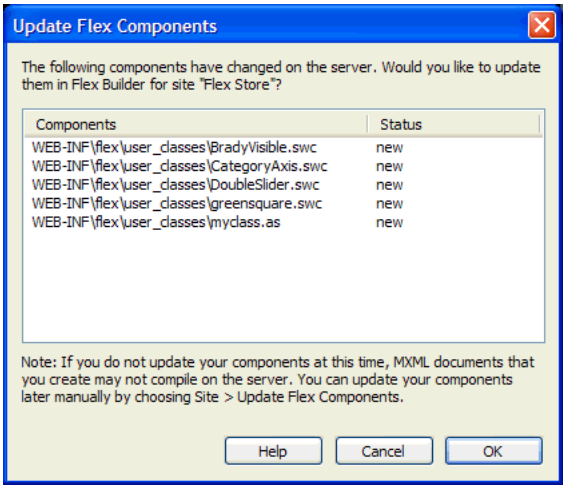
1. Make sure you correctly defined a Flex testing server for your site.

   For more information, see “Starting a new application in Flex Builder” on page 9.
2. Make sure the new or updated custom components are installed on the testing server in one of the following folders in the application’s root folder:
   - /WEB-INF/flex/user_classes/
   - any folder specified by any lib-path or actionscript-classpath entries in the flex-config.xml file in the /WEB-INF/flex folder

   Note: You can also install components in your application’s root folder (the folder containing the MXML file with the <mx:Application> tag). However, Flex Builder does not import these components like other installed components. Instead, they’re downloaded to Flex Builder as dependent files when you perform a put or a get operation. Once the dependent file is downloaded, Flex Builder doesn’t show the components on the Insert bar or in the Insert menu.

3. Select Site > Update Flex Components.

   The Update Flex Components dialog box appears.



The following components have changed on the server. Would you like to update them in Flex Builder for site "Flex Store"?

Components	Status
WEB-INF\flex\user_classes\BradyVisible.swc	new
WEB-INF\flex\user_classes\CategoryAxis.swc	new
WEB-INF\flex\user_classes\DoubleSlider.swc	new
WEB-INF\flex\user_classes\greensquare.swc	new
WEB-INF\flex\user_classes\myclass.as	new

Note: If you do not update your components at this time, MXML documents that you create may not compile on the server. You can update your components later manually by choosing Site > Update Flex Components.

4. If you want to update the listed component or components, click OK.

   For more information, click the Help button in the dialog box.

Flex Builder integrates a new component in its authoring environment as follows:

- The component appears on the Insert bar and in the Insert menu.
- In Code view, the component tag appears in the code hints, and all its properties and property types appear in the property code hints.
- If the component is designed to be visible at runtime and you insert it in an MXML file, Flex Builder displays a visual representation of it in Design view.

  Note: Visible ActionScript components do not render in Design view

- If you select the new component on the page, the Tag inspector lists all its properties and events.

If Flex Builder finds that a component has been deleted from the server, it removes it from the authoring environment.

This synchronization is site-specific. If you switch to a Flex site that uses a different set of components, Flex Builder adjusts the authoring environment accordingly.

CHAPTER 6
Working with Data

You can use Macromedia Flex Builder to bind Flex components, Flex data models, or Flex data services in order to pass data between them, or between the user interface and the back-end system.

This chapter covers the following topics:

- "Binding a control or container" on page 129
- "Editing a binding visually" on page 131
- "Working with Flex data models" on page 131
- "Working with Flex data services" on page 138

Related topics

- "Tutorial: Binding components to data with Flex Builder" on page 59

Binding a control or container

After inserting the components that make up your user interface, (see Chapter 4, "Building a Flex User Interface Visually," on page 97), you can use Flex Builder to visually bind them together. For more information on data binding, see "Data binding in Flex" on page 152.

If you want to bind a component to a data model, see "Working with Flex data models" on page 131.

If you want to bind a component to a web service, an HTTP service, or a Java object, see "Working with Flex data services" on page 138.

To bind a control or container:

1. Make sure your MXML file contains one component to supply the data and one component to accept the data.

   For example, your file can have a TextInput component to let users enter a value and a Label component to display the value entered in the TextInput component.
2. In Design view, select the component that will supply the data.

   For example, select the TextInput component if you want it to supply the data to the Label component.

   You can also click anywhere in the component tag in Code view to select it.

3. In the Bindings panel (Window > Bindings), click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears.

   Tip: You can also open the Add Binding dialog box by right-clicking the component in Design view or its tag in Code view, and then selecting Create New Binding from the context menu.

4. Select the second option ("Data Will Flow From") and then select the property to supply the data.

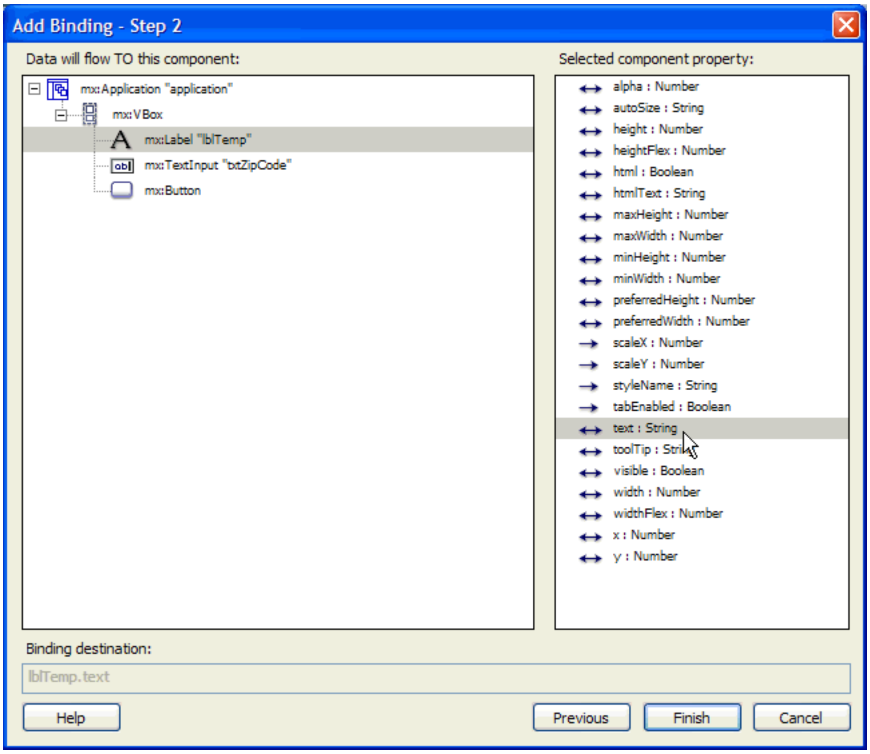
   For example, if you want the source of the data to be the values entered by users in the selected TextInput component, select the text:String property.

5. Click Next to specify the destination of the data.

   The Add Binding - Step 2 dialog box appears.

6. Select the component and property that will accept the data.

   For example, if you have a Label component that you want to use to display the value entered in a TextInput component, select the Label component in the left pane and then select the text:String property in the right pane.



7. Click Finish to create the binding.

   If the source of the binding doesn't have an ID, you will be prompted to set one.

   The new binding appears in the Bindings panel.

If you want, you can start with the destination of the binding and then specify the source.

Related topics

- “Binding a Flex data model to an object” on page 133
- “Binding a Flex data service to an object” on page 139

Editing a binding visually

After you create bindings in Flex Builder, you can edit them visually.

To edit a binding:

1. Select the bound object in Design view or the Data panel.
2. In the Bindings panel, double-click the binding you want to edit.

   The Edit Binding dialog box appears.
3. Make your changes and click Finish.

   For more information, click the Help button on the dialog box.

To change the source or destination property of a binding:

1. Select the bound object in Design view or the Data panel.
2. In the Bindings panel, select the binding you want to change.
3. Click anywhere in the Bound To row of the panel.

   Depending on the direction of the binding (in or out), the Select Source or the Select Destination dialog box appears.
4. Select another source or destination property for the binding, or edit the binding expression in the Binding Source text box (if available).

Related topics

- “Binding a Flex data model to an object” on page 133
- “Binding a Flex data service to an object” on page 139

Working with Flex data models

You can use Flex Builder to insert and bind a Flex data model to a component, a data service, or even another data model. A data model is an object you can use to temporarily store data in memory so that you can more easily manipulate the data. For more information, see “Flex data models” on page 153.

If you want to bind a data model to a web service, an HTTP service, or a Java object, see “Working with Flex data services” on page 138.

This section covers the following topics:

- “Inserting a Flex data model into the MXML file” on page 132
- “Binding a Flex data model to an object” on page 133

Related topics

- "Data binding in Flex" on page 152

Inserting a Flex data model into the MXML file

You can manually insert a Flex data model in an MXML file or you can visually import it.

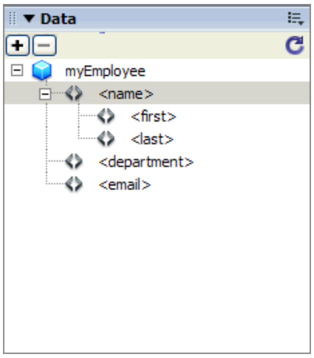
To manually insert a data model into an MXML file:

1. Open the file in Code view (View > Code).
2. Write an `<mx:Model>` tag.

   For example, the following data model stores a person's name, age, and phone number:

   ```
   <mx:Model id="myEmployee">
     <name>
       <first>John</first>
       <last>Doe</last>
     </name>
     <department>Accounting</department>
     <email>jdoe@goodcompany.com</email>
   </mx:Model>
   ```

   For more information, see "Using data models" in Developing Flex Applications Help.
3. Save the file.
4. Use the Data panel (Window > Data) to inspect the data model.



Note: If the data model doesn't appear in the Data panel, click the refresh button on the panel.

To visually import a data model into an MXML file:

1. Open the file in Design or Code view.
2. In the Data panel (Window > Data), click the Plus (+) button and select Model from the pop-up menu.
3. In the dialog that appears, enter the relative path and filename of the XML file, or click the Browse button and select the XML file.

   This specifies the location of an XML file to use as a data model in the current MXML file.

   After you click OK, Flex Builder inserts the data model in the file.

4. Use the Data panel to inspect the data model.

You could also hand-code an `<mx:Model>` tag with a `source` property that specifies an XML file or a URL that returns XML. The following are some examples:

```
<mx:Model source="content.xml" id="contact"/>
```

```
<mx:Model source="http://www.somesite.com/content.xml" id="company"/>
```

Note: If you create a data model with the `source` property, you will be unable to visually create a binding to the model because the binding destination is in an external file.

Binding a Flex data model to an object

After inserting a Flex data model in the MXML file, you can bind it to an object in the file.

You create one binding to retrieve data from a data model and a separate binding to pass data to a data model. The procedures in this section describe how to create each type of binding.

This section covers the following topics:

- "Retrieving data from a Flex data model" on page 133
- "Passing data to a Flex data model" on page 136

Retrieving data from a Flex data model

You can create bindings that retrieve data from data models. For example, you can create a binding that retrieves employee information from a data model so that you can display the information in your MXML file.

To create a binding that retrieves data from a data model:

1. Make sure your MXML file contains a component, data model, or data service that can accept the data from the data model.

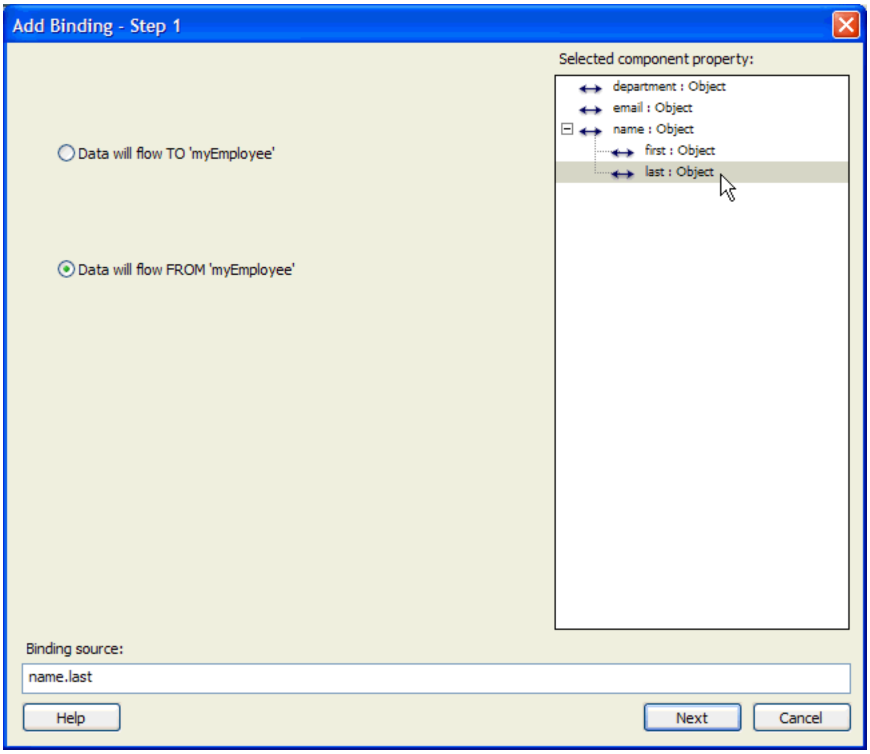
   For example, your file can have a Label component that will display an employee name received from the data model.

2. In the Data panel, select the data model.
3. In the Bindings panel (Window > Bindings), click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears.

4. Select the second option ("Data Will Flow From") and then select the property of the data model to supply the data.

   For example, if you want to retrieve an employee name from the data model, select the appropriate name property of the data model.



Data will flow TO 'myEmployee'

[✓ Data will flow FROM 'myEmployee']

Binding source:

name.last

   The Binding Source text box displays the binding expression Flex Builder will insert in the code. If you want you can modify the expression. For more information, see "Binding data" in Developing Flex Applications Help.

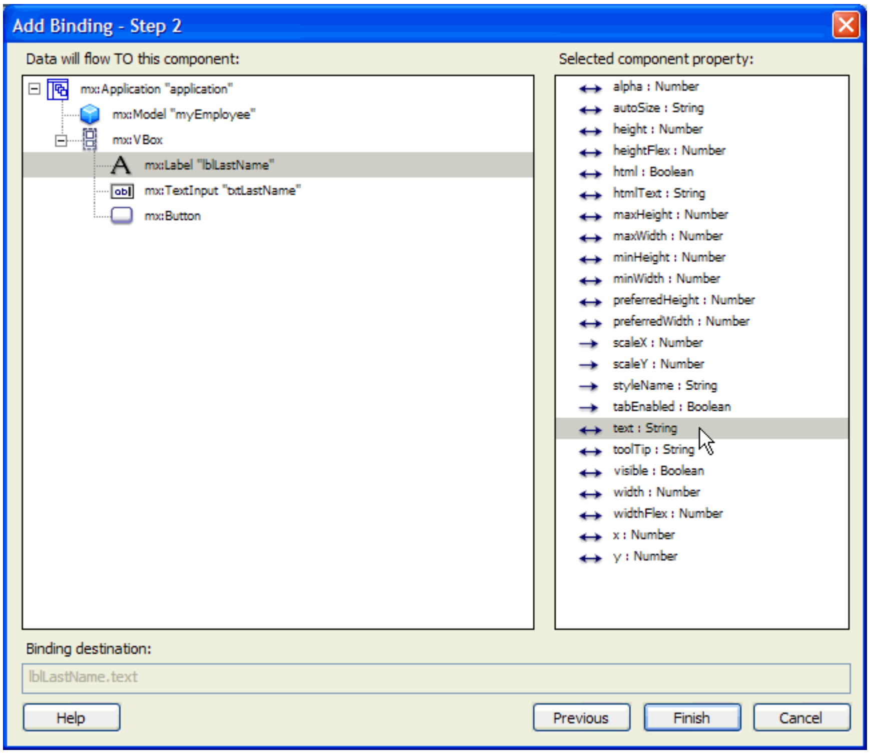
5. Click Next.

   The Add Binding - Step 2 dialog box appears.

6. Select the destination of the data supplied by the data model.

   The destination can be the property of a component, a data service, or even another data model.

   For example, if you have a Label component that you want to use to display the employee name, select the Label component in the left pane and then select the text:String property in the right pane.



Binding destination: lblLastName.text

7. Click Finish to create the binding.

   The new binding appears in the Bindings panel.

8. If the destination of the data is a data service such as a web service, specify the event that invokes the data service.

   For instructions, see “Calling the data service” on page 143.

Passing data to a Flex data model

You can create bindings to pass data to data models. For example, you can create a binding to pass user input to a data model.

To create a binding to pass data to a data model:

1. Make sure your MXML file contains a component, data model, or data service that can supply data to the data model.

   For example, your file can have a TextInput component that lets users enter employee information.
2. In the Data panel, select the data model.
3. In the Bindings panel (Window > Bindings), click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears.
4. Make sure the first option ("Data Will Flow To") is selected and then select the property of the data model to receive the data.

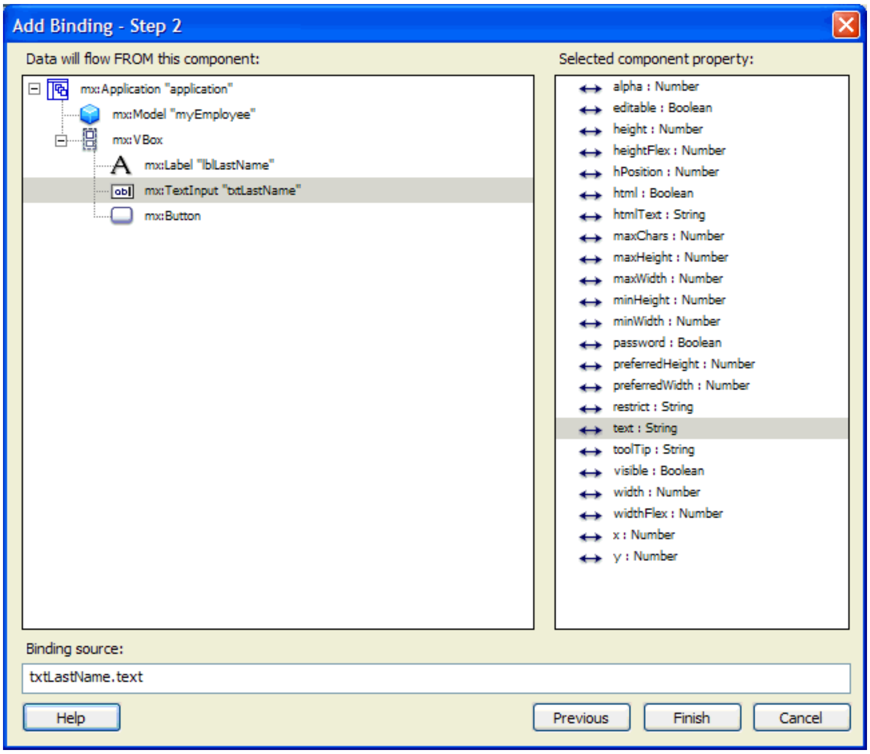
   For example, if you want to pass an employee name to the data model, select the appropriate name property of the data model.
5. Click Next.

   The Add Binding - Step 2 dialog box appears.

6. Select the source of the data passed to the data model.

   The source can be the property of a component, data service, or even another data model.

   For example, if you have a TextInput component that lets users enter an employee name, select the TextInput component in the left pane and then select the text:String property in the right pane.



txtLastName.text

   If you want, you can modify the binding expression in the Binding Source text box.

7. Click Finish to create the binding.

   The new binding appears in the Bindings panel.

8. If the source of the data is a data service such as a web service, specify the event that invokes the data service.

   For instructions, see "Calling the data service" on page 143.

Working with Flex data services

After inserting the components that make up your user interface, (see Chapter 4, "Building a Flex User Interface Visually," on page 97), you can use Flex data services to bind the user interface to the business layer. A Flex data service acts as a bridge to the business layer. For more information, see "Flex data services" on page 154.

After binding a data service to an object in the business layer, you can pass data between them. For example, you can collect information from users and submit it to the business layer, or retrieve and display information from the business layer.

Before you begin, make sure Flex Builder can connect to your Flex server. Certain features discussed in this section will not work if a Flex testing server is not properly defined in Flex Builder. For information on defining a testing server, see "Starting a new application in Flex Builder" on page 9.

This section covers the following topics:

- "Inserting a Flex data service in the MXML file" on page 138
- "Binding a Flex data service to an object" on page 139

Related topics

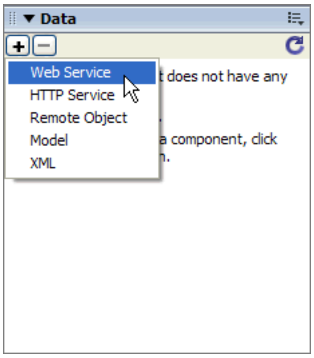
- "Data binding in Flex" on page 152

Inserting a Flex data service in the MXML file

To pass data to and from the business layer, you must first insert a Flex data service into the MXML file. A Flex data service encapsulates a web service, an HTTP service, or a Java object. The role of a Flex data service is to communicate with the business layer.

To insert a Flex data service in an MXML file:

1. With the MXML file open in Flex Builder, open the Data panel (Window > Data).
2. Click the Plus (+) button on the panel and select the type of object in your business layer that will provide or receive the data.



Web Service inserts a service that lets you access the methods of a SOAP-compliant web service.

HTTP Service inserts a service that lets you retrieve data from an HTTP address.

Remote Object inserts a service that lets you access the methods of a Java object.

After clicking one of the items, a dialog box appears to let you specify the web service, HTTP service, or Java object you want to use.

3. If you selected Web Service, select a named service from the pop-up menu or enter the URL of a web service's WSDL file in the text box.

   For more information, click the Help button on the dialog box.

4. If you selected HTTP Service, select a named service from the pop-up menu or enter the address of an HTTP data source in the text box.

   For more information, click the Help button on the dialog box.

5. If you selected Remote Object, select a named service from the pop-up menu or enter the location of the Java object or Flash Remoting service in the text box.

   For more information, click the Help button on the dialog box.

After closing the dialog box, the web service, HTTP service, or remote object appears in the Data panel indicating that a reference to it has been inserted into the MXML file and that you can bind it to a component or a data model in the file. A Flex web service is easier to use than the other Flex data services because you can visually bind to the web service's results or parameters without knowing them in advance—Flex Builder retrieves and displays the schema for you. Flex Builder does not retrieve and display the schema of HTTP services and remote objects. You must know the schema of these objects in advance to recreate it in your code (for visual binding), or to write the binding expression yourself.

Related topics

- "Working with Flex data services" on page 138

Binding a Flex data service to an object

After inserting a Flex data service in the MXML file, you can bind it to a property of a component or data model in the file.

You must create one binding to retrieve data from the data service and a separate binding to pass data to the data service. In either case, you must create an event handler to call the data service.

This section covers the following topics:

- "Retrieving data from a Flex data service" on page 139
- "Passing data to a Flex data service" on page 141
- "Calling the data service" on page 143

Retrieving data from a Flex data service

You can create bindings that retrieve data from Flex data services. For example, you can create a binding that retrieves weather information from a web service so that you can display it in your MXML file.

To create a binding that retrieves data from a data service:

1. Make sure your MXML file contains a component or data model that can accept the data from the data service.

   For example, your file can have a Label component that will display a temperature received from the data service.

2. In the Data panel, select the data service.

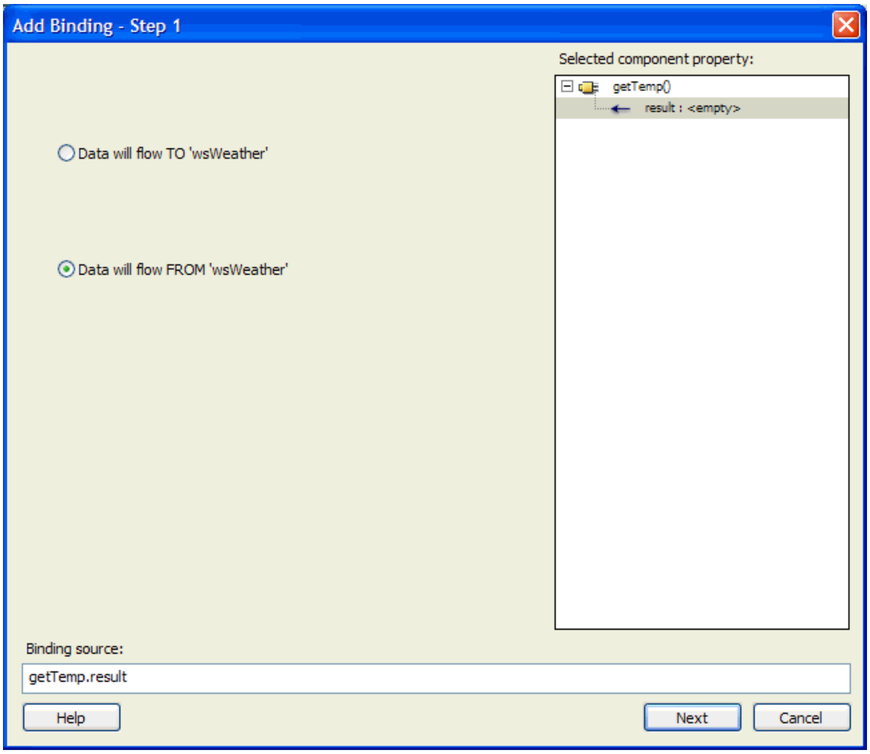
   To add a data service to the panel, see "Inserting a Flex data service in the MXML file" on page 138.

3. In the Bindings panel (Window > Bindings), click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears.

4. Select the second option ("Data Will Flow From") and then select the property of the data service to supply the data.

   For example, if you want to retrieve weather data from a web service, select one of the service's weather properties (for example, temperature) from the results returned by one of the web service's methods.



( ) Data will flow TO 'wsWeather'

[✓ Data will flow FROM 'wsWeather']

Binding source: getTemp.result

Flex Builder doesn't retrieve and display the schema of HTTP services or remote objects, so you'll need to know the source property of these services in advance and specify it in the Binding Source text box. The Binding Source text box shows the binding expression Flex Builder will insert in the code. For more information, see "Binding data" in Developing Flex Applications Help.

5. Click Next.

   The Add Binding - Step 2 dialog box appears.

6. Select the destination of the data supplied by the data service.

   The destination can be the property of a component or data model.

   For example, if you have a Label component that you want to use to display the current temperature, select the Label component in the left pane and then select the text:String property in the right pane.

7. Click Finish to create the binding.

   The new binding appears in the Bindings panel.

8. Create an event handler to call the data service.

   You can create this event handler rapidly with Flex Builder. For instructions, see "Calling the data service" on page 143.

Passing data to a Flex data service

You can create bindings to pass data to Flex data services. For example, you can create a binding to pass a zip code to a web service so that you can obtain the current temperature at that zip code location.

To create a binding to pass data to a data service:

1. Make sure your MXML file contains a component or data model that can supply data to the data service.

   For example, your file can have a TextInput component that lets users enter a zip code.

2. In the Data panel, select the Flex data service.

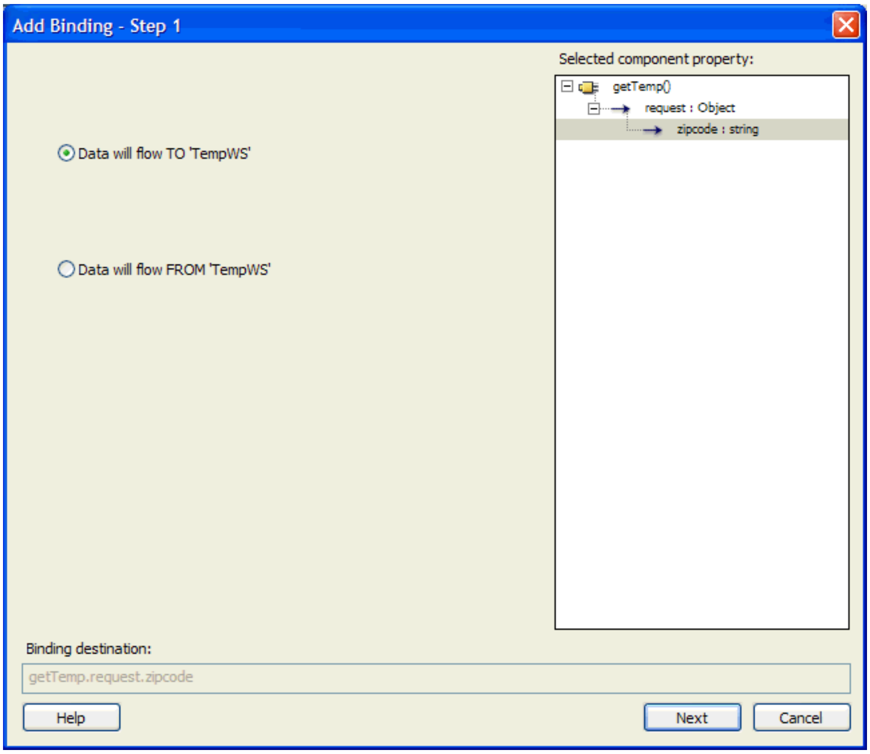
   To add a data service to the panel, see "Inserting a Flex data service in the MXML file" on page 138.

3. In the Bindings panel (Window > Bindings), click the Plus (+) button.

   The Add Binding - Step 1 dialog box appears.

4. Make sure the first option ("Data Will Flow To") is selected and then select the property of the data service to receive the data.

   For example, if you want to pass a zip code entered by a user to a web service, select the appropriate zip code property of the web service's request object.



[✓ Data will flow TO 'TempWS'] Data will flow FROM 'TempWS'

Binding destination: getTemp.request.zipcode

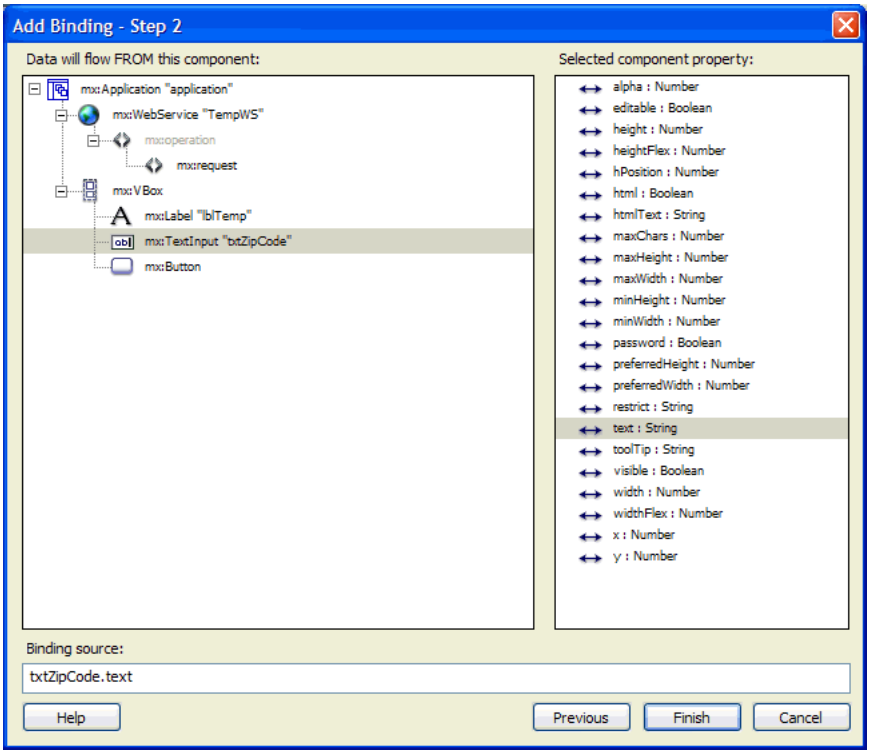
5. Click Next.

   The Add Binding - Step 2 dialog box appears.

6. Select the source of the data passed to the data service.

   The source can be the property of a component or data model.

   For example, if you have a TextInput component that lets users enter a zip code, select the TextInput component in the left pane and then select the text:String property in the right pane.



   If you want, you can modify the binding expression in the Binding Source text box.

7. Click Finish to create the binding.

   The new binding appears in the Bindings panel.

8. Create an event handler to call the data service.

   You can create this event handler rapidly with Flex Builder. For instructions, see "Calling the data service" on page 143.

Calling the data service

If you create a binding that passes data to, or retrieves data from, a data service, then you must create an event handler that calls the data service. A component can call a data service such as a web service only in response to an event. Otherwise, the data service is never invoked. You can create this event handler rapidly with Flex Builder.

To create an event handler to call a data service:

1. Make sure your MXML file contains a binding to a data service such as an web service, an HTTP service, or a remote object.

   The binding must appear in the Bindings panel (Window > Bindings). For more information, see "Working with Flex data services" on page 138.

2. In Flex Builder, select the component that will generate the event to invoke the data service.

   For example, you can select a Button component for users to click.

3. In the Events panel of the Tag Inspector (Window > Tag Inspector), click the field next to the event that will trigger the service call.

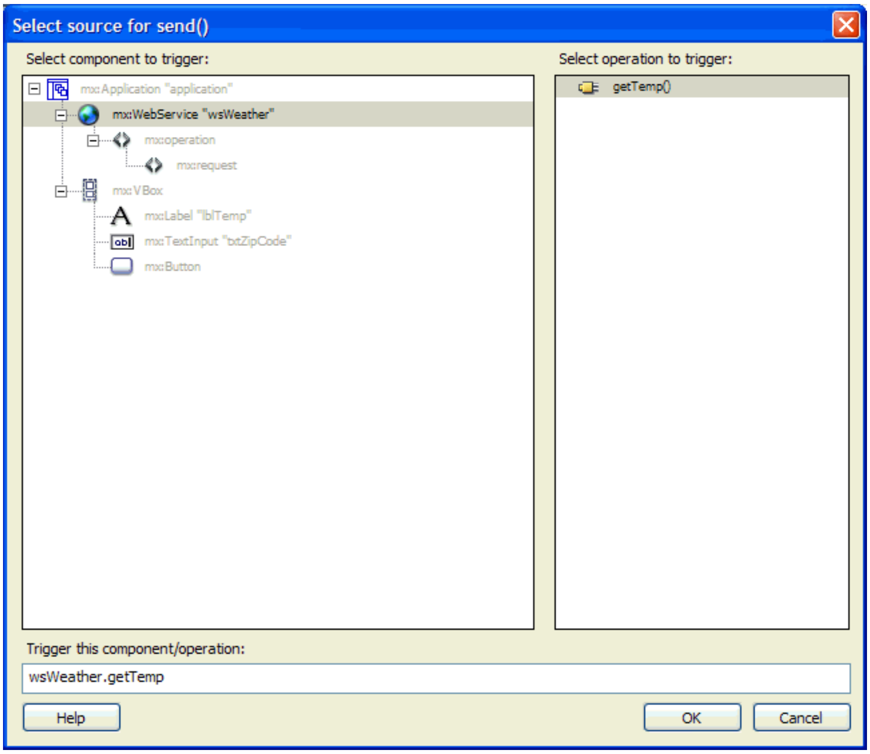
   A lightning bolt icon appears next to the field.

4. Click the lightning bolt icon.

   The Select Source for send() dialog box appears.

5. Specify the data service to invoke.

   For example, if you want to invoke a weather-related web service, select the web service in the left pane (you must define the web service in the Data panel first) and then select the operation to invoke in the right pane.


Select source for send()
Select component to trigger:
Select operation to trigger:
mx:Application "application"
mx:WebService "wsWeather"
mx:operation
mx:request
mx:VBox
mx:Label "lblTemp"
mx:TextInput "txtZipCode"
mx:Button
getTemp()
Trigger this component/operation:
Help
OK
Cancel

wsWeather.getTemp

6. Click OK to create the event handler.

Related topics

- “Inserting a Flex data service in the MXML file” on page 138
- “Assigning event handlers to components” on page 116

APPENDIX A
Basic Flex Concepts

This appendix is intended for Macromedia Flex Builder users who are unfamiliar with Flex technology. It explains basic Flex concepts, but not specific development techniques. For more detailed information, see Developing Flex Applications Help.

This appendix contains the following sections:

- "About the Flex application model" on page 147
- "Typical application development steps" on page 148
- "About Flex files" on page 148
- "About Flex containers, components, and controls" on page 150
- "About Flex data bindings" on page 152

About the Flex application model

When you build an application using Flex, you describe its user interface using containers and controls. A container is a rectangular region of the screen that contains controls and other containers. Examples of containers are a Form container used for data entry, a Box, and a Grid. A control is a form element, such as a Button or Text Input field.

Containers and controls define the application's user interface. In an MVC design pattern, those pieces of the application model represent the view. The model is represented by the data model. Flex data models let you separate your application's data and business logic from the user interface.

Data binding is the process of tying the data in one object to another object. The data model supports bidirectional data binding for writing data from Flex controls to the data model, or for reading data into controls from the model. You can also bind server data to a data model or directly to Flex controls. For example, you can bind the results returned from a web service to the data model, and then have that data propagate to your form controls.

The data model supports automatic data validation. This means that you can use the Flex ZipCode validator to make sure that the value in a model field is a valid ZIP code. If the data is invalid, you can display a message to the user so that the user can correct the error.

Typical application development steps

You typically develop a Flex application using the following steps:

1. Create an MXML file with the `<mx:Application>` root tag.
2. Add one or more containers.
3. Add controls to a container, such as input fields, buttons, and output fields.
4. Define a data model.
5. Add a web service, HTTP service, or request to a remote object such as a Java object or a Flash Remoting service.
6. Add validation to input data.
7. Add a script to extend a component.

About Flex files

Flex application development is based on the same iterative process used for developing other web applications. Developing a Flex application is as easy as opening Flex Builder, inserting some containers and components, and running the file. The basic difference between Flex and other server technologies is that Flex generates Flash files (SWF files)—not HTML code.

Flex applications can consist of a mix of MXML and ActionScript. MXML is a tag-based language for building Flex applications. ActionScript is a procedural-based language for developing applications in Flash.

This section covers the following topics:

- “About MXML files” on page 148
- “About ActionScript files in Flex applications” on page 149

About MXML files

MXML is a XML-based language for building Flex applications. Like HTML, you can use MXML to build web user interfaces. Unlike HTML, you can use MXML to pass data between components, and between the presentation layer (the user interface) and the business layer (back-end systems). For more information on the MXML language, see “Using MXML” in Developing Flex Applications Help.

The following MXML code creates an input field, a button, and a text box. When the user clicks the button, any text entered in the input field is copied to the text box.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.macromedia.com/2004/mxml"
  width="200" height="100">

  <!-- vertical box container for the application. -->
  <mx:VBox id="myvbox">

    <!-- input field -->
    <mx:TextInput id="myInput" width="150" />
```

```
        <!-- button that triggers the copy -->
        <mx:Button id="myButton" label="Copy Text"
          click="myText.text=myInput.text" />

        <!-- output text box -->
        <mx:TextArea id="myText" width="150" height="20" />

    </mx:VBox>

</mx:Application>
```

The content between the starting and closing `<mx:Application>` tags defines a Flex application.

Related topics

- "Creating MXML or ActionScript files" on page 77
- "Testing and debugging Flex files" on page 86
- Flex ActionScript and MXML API Reference help

About ActionScript files in Flex applications

Although MXML is the primary authoring language for Flex applications, you use ActionScript to manipulate and extend Flex applications. ActionScript is a procedural-based language similar to JavaScript for manipulating Flash. For more information, see "Using ActionScript" in Developing Flex Applications Help.

To use ActionScript in Flex applications, you must either add script blocks to the MXML file using the `<mx:Script>` tag, or reference external ActionScript files using the same tag. The following example shows both methods.

```
<mx:Script>
  var myVar:Number;
  function doSomething() {
    myVar = myVar + 1;
  }
</mx:Script>

<mx:Script source="myscripts/file1.as"></mx:Script>
```

In general, maintaining an application is easier if you place the ActionScript in a separate file and use the `source` property of the `<mx:Script>` tag to import it in the MXML file.

If you add script blocks to an MXML file using the `<mx:Script>` tag, you must be careful about using special characters within the script block.

A good coding practice is to "escape" all ActionScript blocks in your MXML files so that the MXML parser ignores them. Because MXML is an XML-based language, the MXML parser will interpret special XML characters such as angled brackets as MXML, not ActionScript.

To escape an ActionScript block, enclose it within a CDATA block. A CDATA block is an XML technique used to escape character data from the XML parser. The block is created as follows in the code:

```
<![CDATA[
  ...
]]>
```

The parser ignores everything in the CDATA block, including special symbols, strings, and numbers. The Flex compiler can still read and compile the ActionScript in the block. The following example shows the technique:

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

    var myVar:Number;
    function doSomething() {
      myVar = myVar + 1;
    }

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

Related topics

- "Creating MXML or ActionScript files" on page 77
- "Debugging ActionScript" on page 90
- Flex ActionScript Language Reference help

About Flex containers, components, and controls

Building a Flex user interface consists of inserting Flex containers, components, and controls in an MXML file.

This section covers the following topics:

- "Flex components" on page 150
- "Flex controls" on page 151
- "Flex containers" on page 151

Flex components

Flex includes a component-based development model that you use to develop your application and its user interface. You can use the prebuilt components included with Flex, you can extend components to add new properties and methods, and you can create new components as required by your application.

Components are extremely flexible and allow you a large amount of control over the component's appearance, how the component reacts to user interactions, the font and font size of any text included in the component, the size of the component in the application, and many other characteristics.

The following are some of the characteristics of components:

Events Application or user actions that require a component response. Events include component creation, mouse actions such as a mouse over, and button clicks.

Styles Characteristics such as font, font size, and text alignment. These are the same styles as defined and used with Cascading Style Sheets (CSS).

Behaviors Visible or audible changes to the component triggered by an application or user action. Examples of behaviors are moving or resizing a component based on a mouse click.

Skins Symbols that control a component's appearance.

Size Height and width of a component. All components have a default size. You can use the default size, specify your own size, or allow Flex to resize a component as part of laying out your application.

Related topics

- Chapter 5, "Working with Components," on page 119
- Chapter 4, "Building a Flex User Interface Visually," on page 97

Flex controls

Controls are user-interface components, such as Button, TextArea, and ComboBox controls. You place controls in containers, which are user-interface components that provide a hierarchical structure for controls and other containers. Typically, you define a container, and then insert controls or other containers in it.

At the root of a Flex application is the `<mx:Application>` tag, which represents a base container that covers the entire Macromedia Flash Player drawing surface. You can place controls or containers directly under the `<mx:Application>` tag or in other containers.

Most controls have the following characteristics:

- MXML API for declaring the control and the values of its properties and events
- ActionScript API for calling the control's methods and setting its properties at runtime
- Customizable appearance using styles, skins, and fonts

Related topics

- Chapter 5, "Working with Components," on page 119
- Chapter 4, "Building a Flex User Interface Visually," on page 97

Flex containers

A container defines a rectangular region of the Macromedia Flash Player drawing surface. Within a container, you define the components, both controls and containers, that you want to appear within the container. Components defined within a container are called children of the container.

At the root of a Macromedia Flex application is a single container, called the Application container, that represents the entire Flash Player drawing surface. This Application container holds all other containers, which can represent dialog boxes, panels, and forms.

A container has predefined rules to control the layout of its children, including sizing and positioning. Flex defines layout rules to simplify the design and implementation of rich internet applications, while also providing enough flexibility to let you create a diverse set of applications.

One advantage to having predefined layout rules is that your users will soon grow accustomed to them. That is, by standardizing the rules of user interaction, your users will not have to think about how to navigate the application, but can instead concentrate on the content that the application offers.

Another advantage is that you do not have to spend time defining navigation and layout rules as part of the design process. Instead, you can concentrate on the information that you want to deliver, and the options that you want to provide for your users, and not worry about implementing all the details of user action and application response. In this way, Flex provides the structure that lets you quickly and easily develop an application with a rich set of features and interactions.

If you do want a greater level of control over sizing and positioning, Flex provides the Canvas container. This container has no built-in layout rules, but instead lets you explicitly set the position and size of its children. For more information, see Canvas layout container.

Related topics

- Chapter 5, "Working with Components," on page 119
- Chapter 4, "Building a Flex User Interface Visually," on page 97

About Flex data bindings

Building a Flex user interface consists not only of inserting containers and components but of defining event-based, programmatic relationships between the components as well as between the components and the back-end system. For example, you may want component "A" to pass data to component "B" when a certain event such as a mouse click is triggered. Flex data bindings let you define these relationships.

This section covers the following topics:

- "Data binding in Flex" on page 152
- "Flex data models" on page 153
- "Flex data services" on page 154

Data binding in Flex

A data binding copies the value of a property in one object to a property in another object. You can bind the properties of following objects: Flex components, Flex data models, and Flex data services.

The object property that provides the data is known as the source property. The object property that receives the data is known as the destination property.

The following example binds the text property of a TextInput component (the source property) to the text property of a Label component (the destination property) so that text entered in the TextInput component is displayed by the Label component:

```
<mx:TextInput id="LNameInput"></mx:TextInput>
...
<mx:Label text="{LNameInput.text}"></mx:Label>
```

Flex Builder binds data with the curly braces syntax. To understand this syntax, see "Binding data with the curly braces syntax" in Developing Flex Applications Help. If you hand-code your pages, you can also use the `<mx:Binding>` tag to bind data. For more information, see "Binding data with the <mx:Binding> tag" in Developing Flex Applications Help.

Related topics

- "Binding data" in Developing Flex Applications Help
- "Flex data services" on page 154
- "Working with Flex data services" on page 138
- "Binding a control or container" on page 129
- "Working with Flex data models" on page 131

Flex data models

A data model is an object you can use to temporarily store data in memory so that you can more easily manipulate the data. For example, a data model could store information such as a person's name, age, and phone number, as follows:

```
<mx:Model id="myEmployee">
  <name>
    <first>John</first>
    <last>Doe</last>
  </name>
  <department>Accounting</department>
  <email>jdoe@goodcompany.com</email>
</mx:Model>
```

The fields of a data model can contain static data as above, or they can be bound to other objects. This binding allows you to pass data to and from the data model.

You can also define the data model in a separate XML file or a URL that returns XML. In that case, you can use the `source` property in the `<mx:Model>` tag to specify the XML file in the local web application directory, or to specify the HTTP URL that returns XML.

In the following examples, the content of the myContacts data model is an XML file named content.xml stored in the same folder as the MXML file. The content of the myCompany data model is a fictional HTTP URL that returns XML:

```
<mx:Model source="content.xml" id="myContacts"/>

<mx:Model source="http://www.somesite.com/companyinfo.xml" id="myCompany"/>
```

If you hand-code your pages, you can use ActionScript or the `<mx:Model>` tag to define a data model. In general, you should use a tag for simple data structures and ActionScript for more complex structures and client-side business logic. However, Flex Builder only recognizes data models defined with the `<mx:Model>` tag.

Related topics

- "Using data models" in Developing Flex Applications Help
- "Working with Flex data models" on page 131

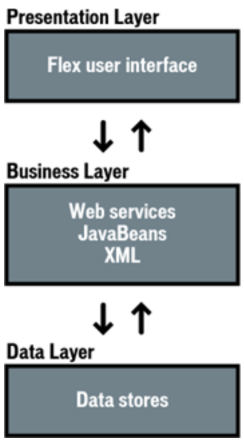
Flex data services

A Flex data service is an object you insert in an MXML file to communicate with the business layer of a multi-tier application. You use data services to send and receive data from web services, HTTP URLs, and remote objects such as server-based Java objects.

In most enterprise applications, the presentation layer (the user interface) is separated from the business layer containing the code that implements the functionality or business rules of the application, and the data layer containing databases and other data stores. The presentation layer interacts with the business and data layers to produce an integrated application.

Note: The development of the business and data layers is beyond the scope of this guide. Flex is a presentation technology designed to work with existing business and data layers.

As a presentation layer technology, Flex does not support direct access to databases. The code contained in the business layer is responsible for communicating with databases. In other words, Flex must work with the business layer to get access to the underlying data.



The Flex presentation layer communicates with the business layer by using Flex data services, which are objects you insert in a Flex file. Specifically, you can use Flex data services to interact with the following:

- Web services
- HTTP services
- Remote objects

A web service is a software system designed to support machine-to-machine interaction over a network. It has an interface described in a machine-processable format called WSDL. Other systems interact with the web service in a manner prescribed by its description.

An HTTP service is nothing more than an HTTP request to a URL. The primary purpose of HTTP services is to retrieve XML data from an external source.

A remote object is a Java object in the Flex application's classpath.

Related topics

- "Working with Flex data services" on page 138
- "Using Data Services" in Developing Flex Applications Help
- "Working with web services" in Developing Flex Applications Help
- "Working with remote object services" in Developing Flex Applications Help

INDEX

A

ActionScript
- about 149
- CDATA blocks 150
- creating new files 77
- debugger 90
- language reference 17

ActionScript and MXML API Reference 17
adding effects 111
application file 119
application root folder 11
applying class styles 108
applying inline styles 109
Attributes panel 81, 103
authoring environment 5

B

backend systems. *See* business layer
BEA WebLogic 20
bindings
- data models 133
- data services 139
- defined 152
- endpoint 152
- source 152
- to business layer 138
- to other components 129

Bindings panel 7, 140
breakpoints 91
business layer 154

C

Call Stack panel 93
CDATA block 150
child containers, selecting 103
class selectors 105
class styles, applying 108
client-server interactions, monitoring 93
code formatting preferences 84
code hint preferences 85
code hints 126
Code inspector 78
Code view 6, 78
coding
- code hint preferences 85
- code hints 80
- code-coloring preferences 83
- customizing the editor 82
- editing tags 81
- formatting 84
- other information 78
- preferences 82
- views 6, 78

compiling 9, 87
components
- copy and paste 102
- creating 123
- custom 122
- editing properties 103
- effects 111
- importing 127
- inserting 100, 119
- local 120
- Macromedia 122
- namespaces 121
- packaging 126
- removing padding and border 98
- repositioning 101
- resizing 102
- SWC 123
- synchronizing 127

containers, inserting 99
context-sensitive help 17
copying and pasting components 102

CSS
  about 105
  applying class styles 108
  applying inline styles 109
  class selectors 105
  creating styles 107
  deleting styles 110
  embedded styles 105
  external styles 105
  importing stylesheet 106
  inline styles 105
  modifying styles 109
  tag selectors 105
custom components 122
customer support 18

D

data models
  binding to objects 133
  defined 153
  inserting 132
Data panel 8, 138
data services
  binding to components 139
  defined 154
  inserting 138
databases 154
debugging
  Actionscript debugger 90
  compiling files 87
  Network Monitor 93
  previewing in a browser 88
  setting breakpoints 91
  validating MXML 86
defining CSS styles 107
deleting CSS styles 110
dependent files 89
Design view 6
  Expanded mode 98
  Standard mode 98
Developer Edition of Flex 20
developer guide 17
developer resources 18
Developing Flex Applications 17
documentation 17
Dreamweaver, using 17

E

Edit Coloring Scheme dialog box 83
editing CSS styles 109
effects, adding 111
event handlers 116

F

Flash components 123
  inserting 119
  namespaces 121
Flex
  ActionScript and MXML API reference 17
  ActionScript language reference 17
  application root folder 11
  CSS styles 105
  data bindings. *See* bindings
  data models. *See* data models
  data services 138
  data services. *See* data services
  developer guide 17
  documents 77, 148
  HTTP services 155
  installing 20
  server 20
  web services 155
Flex data bindings. *See* bindings
Flex Developer Edition 20
Flex Sample Apps 12

H

handling events 116
help 17
HTTP services 155

I

IBM WebSphere 20
importing components 127
inline styles, applying 109
Insert bar 6
inserting
  components 100, 119
  containers 99
  data models 132
  data services 138
  local components 120

J

Jakarta Tomcat 20
Java objects 155
JRun 4 20

L

local component 120

M

Macromedia components 122
Macromedia Developer Center 18
Macromedia JRun 4 20
managing files 21
models. *See* data models
modifying CSS styles 109
monitoring interactions with the server 93
MXML
  about 148
  API Reference 17
  application file 119
  component namespaces 121
  containers 99
  copy and paste components 102
  creating components 123
  creating new files 77
  CSS styles 105
  editing component properties 103
  importing components 127
  inserting
    components 100, 119
  reference 17
  repositioning components 101
  resizing components 102
  validating 86

N

namespaces 121
navigator containers, cycling through child containers 103
Network Monitor 93
New Document dialog box 77

O

Output panel 87

P

packaging components 126
presentation layer 154
previewing in a browser 88
Property inspector (Flex) 7

Q

Quick Tag Editor 103, 104

R

reference
  ActionScript and MXML API 17
  ActionScript language 17
Relevant CSS tab 109
repositioning components 101
requirements for development 18
resizing components 102

S

selecting child containers 103
services. *See* data services
setting up a site 21
Site Definition dialog box 10
site setup 21
Standard and Expanded modes 6
stylesheets, importing 106
support 18
SWC 123
synchronizing components 127

T

Tag inspector 7, 81, 103
tag reference 17
tag selectors 105
testing server 9
Tomcat 20
transferring files to the server 21
tutorial 21

U

Using Dreamweaver 17

V

validating MXML 86
Validation panel 86
Variables panel 92
visual environment 8

W
Watch panel 92
web servers 19
web services, Flex 155
WEB-INF 125, 127
WebLogic 20
WebSphere 20
workspace 5