Chapter 4: Code Editing in Flash Builder

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

About code editing in Flash Builder

When designing and developing Flex and ActionScript applications, you work in the Flash Development perspective, which contains the Flash Builder editors and all the views that support code editing and design tasks. The configuration of the Flash Development perspective depends on which editor and mode you're working in. For example, when you create a Flex project, the MXML editor works in two modes (Source and Design) and each mode contains its own collection of supporting views. For an overview of the Flash Development perspective, see "The Flash Development perspective" on page 8.

MXML, ActionScript, and CSS editors

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

The MXML editor lets you edit MXML files. The MXML editor contains two modes: Source and Design. In Source mode, you write MXML and embed ActionScript and CSS code contained within `<fx:Script>` and `<fx:Style>` tags. In Design mode, you lay out and design your applications (see "Building a user interface with Flash Builder" on page 177).

The ActionScript editor lets you edit AS files, which include main files for ActionScript projects, and class and interface files. For more information, see "ActionScript projects" on page 63.

You use the CSS editor to write Cascading Style Sheets (CSS files). For more information, see "Applying styles" on page 200 and Styles and themes.

MXML, ActionScript, and CSS content assistance

As you enter MXML, ActionScript, and CSS code, hints and ASDoc reference documentation displays to help you complete your code. This feature is called *Content Assist*. Flash Builder also assists you in quickly developing your code by including MXML tag completion, automatic import management, integration with *Adobe Flex Language Reference*, and the capability of choosing different colors and fonts to display your code in the workspace. For more information, see "Flash Builder coding assistance" on page 97.

Custom component and ActionScript class and interface code hints

In addition to the built-in support for the Flex framework, code hints are provided for all custom components and ActionScript classes and interfaces that are included in your project. For more information, see "Using Content Assist" on page 101.

Selecting recommended types for namespaces

Flash Builder provides filters and Content Assist hints to help you identify recommended types for a namespace. You can also cycle through various filters or settings to locate all types, events, and properties available from the Flex framework. For more information, see "Using Content Assist" on page 101.

Generating accessor functions (getters and setters)

Get and set accessor functions let you keep class properties private to an ActionScript class. Users of the class can access those properties as if they were accessing a class variable (rather than calling a class method). Flash Builder can generate ActionScript get and set accessor functions for class variables. For more information, see “Generating Accessor Functions” on page 65.

Streamlined code navigation

To easily navigate the many files and lines of code in your projects, Flash Builder provides convenient shortcuts, such as folding and unfolding code blocks, opening the source of external code definitions, browsing and opening class types, and so on. The Outline view also provides you with a convenient way to inspect the structure of and navigate to lines of code in your documents. For more information, see “Navigating and organizing code” on page 103.

Find references and code refactoring

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

Automatic syntax error checking

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

Syntax coloring

You specify the set of colors to be applied throughout your code in the MXML, ActionScript, and CSS editors on the Syntax Coloring Preferences page. Font style options can also be applied and previewed from the same page. For more information, see “Getting help while writing code” on page 103 and “Using Content Assist” on page 101

Editor and debugger integration

The MXML and ActionScript editors work with the Flash Debug perspective to assist you in debugging your code. You set breakpoints in your code to suspend your application at troublesome or otherwise crucial lines of code. When you begin a debugging session, the Flash Debug perspective is displayed when the first breakpoint is reached. You can then inspect the state of your application and isolate and resolve the problems in your code. For more information about debugging your code, see “Debugging your applications” on page 132.

Flash Builder coding assistance

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

Content Assist

Available in the MXML, ActionScript, and CSS editors, Content Assist provides code hints to help you complete your code expressions. This assistance includes help in selecting recommended classes, properties, and events available in a namespace. For more information, see “About Content Assist” on page 98.

Auto import management

As you enter code with Content Assist, the fully qualified type names are automatically imported into the document as needed. In an ActionScript document, the fully qualified type name is added to the beginning of the document with the other import statements. In an MXML document, the import statement is added to an existing script block if one exists; if not, Flash Builder creates a script block. In an ActionScript document, you can also optionally sort import statements. For more information, see "Organizing import statements" on page 109.

MXML tag completion

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

Working with view states

The MXML editor provides support in Source mode for editing view states. If you define view states at the root of your application, the editor provides a Show State pop-up menu. Use this menu to emphasize the selected state in the editor. Content Assist also provides support for selecting view states for components and properties. For information on this and other features for editing view states, see "Creating and editing view states in source code" on page 216.

ASDoc reference documentation

Flash Builder displays ASDoc reference documentation in the MXML and ActionScript editors when you use Content Assist or hover over a type in the editor. ASDoc documentation for selected classes also appears in the ASDoc view. This feature supports user-generated ASDoc documentation.

For more information, see "Providing ASDoc comments for custom components" on page 128.

Generating Event Handlers

Flash Builder editors provide assistance in the generation of event handlers for components. For each event available for a component, you can generate an event handler that is placed within `<Script>` tags. The event handler is referenced by the event property in the component tag.

For more information, see "Generating event handlers" on page 194.

Context-sensitive language reference Help

The *Adobe Flex Language Reference* is integrated into the editor and you can easily access it by selecting an ActionScript language element, an MXML component tag or attribute, and then pressing Shift+F2. For more information, see "Getting help while writing code" on page 103.

Formatting assistance

To streamline the work of coding your applications, the editors can help you reformat blocks of code and perform bulk edits. For more information, see "Formatting and editing code" on page 109.

About Content Assist

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

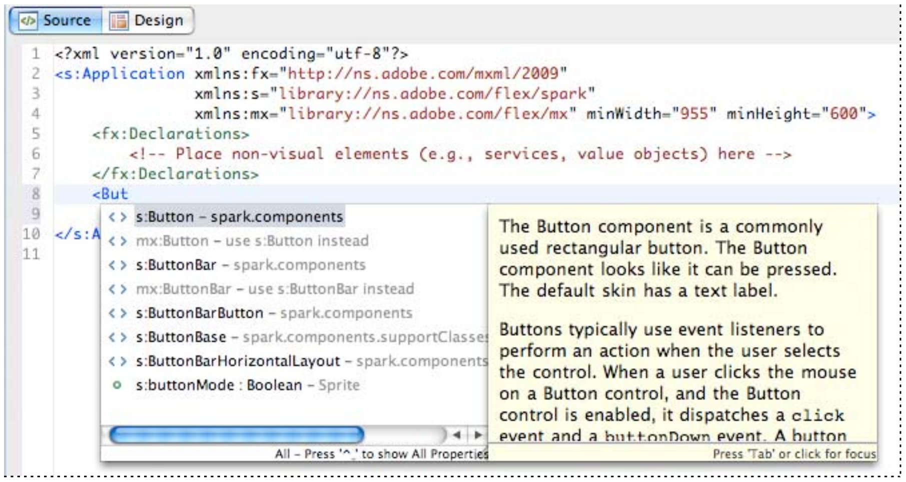
Content Assist also displays ASDoc reference documentation available for the MXML component or ActionScript code you are authoring. ASDoc documentation is also available by hovering over an MXML component or ActionScript class. If you select a class in the editor, the ASDoc content also appears in the ASDoc view.

Content Assist is available for the MXML editor, ActionScript editor, and CSS editor. How you use Content Assist varies with each editor. Code hints appear whenever the framework or language (MXML, ActionScript, and CSS) provides options for you to complete the current expression.

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

Displaying code hints with Content Assist

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



```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <But

</s:A
```

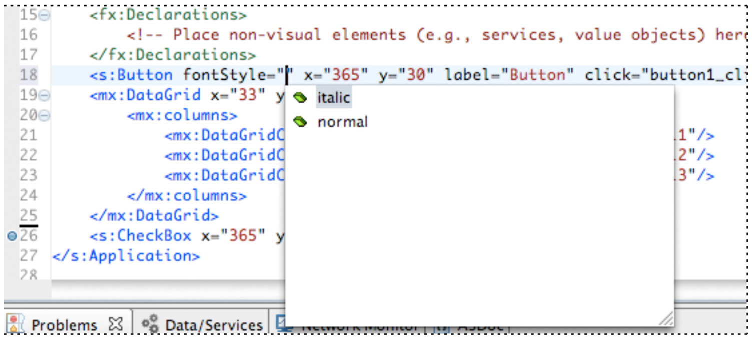
```
s:Button – spark.components
mx:Button – use s:Button instead
s:ButtonBar – spark.components
mx:ButtonBar – use s:ButtonBar instead
s:ButtonBarButton – spark.components
s:ButtonBase – spark.components.supportClasses
s:ButtonBarHorizontalLayout – spark.components
s:buttonMode : Boolean – Sprite
All – Press '^ ' to show All Properties
```

The Button component is a commonly used rectangular button. The Button component looks like it can be pressed. The default skin has a text label.

Buttons typically use event listeners to perform an action when the user selects the control. When a user clicks the mouse on a Button control, and the Button control is enabled, it dispatches a click event and a buttonDown event. A button

Press 'Tab' or click for focus

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



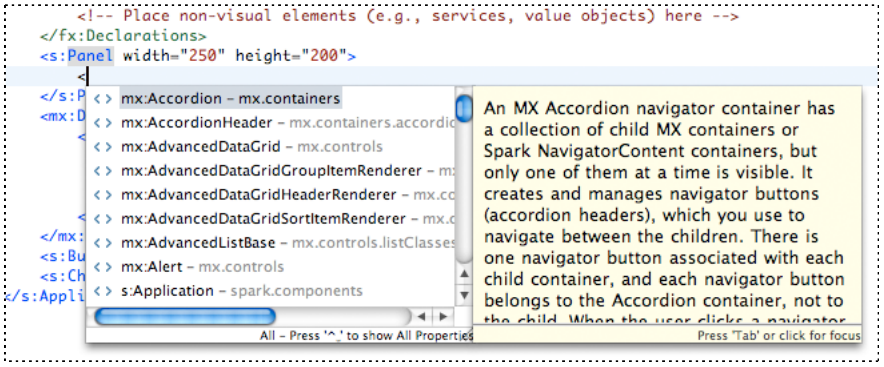
```
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) her
    </fx:Declarations>
    <s:Button fontStyle="" x="365" y="30" label="Button" click="button1_cl
    <mx:DataGrid x="33" y
        <mx:columns>
            <mx:DataGridC                                         1"/>
            <mx:DataGridC                                         2"/>
            <mx:DataGridC                                         3"/>
        </mx:columns>
    </mx:DataGrid>
    <s:CheckBox x="365" y
</s:Application>
```

```
italic
normal
```

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

Content Assist in the MXML editor

In the MXML editor, code hints appear automatically as you enter your code. The following example shows the code hints that are displayed when you add a tag to a Panel tag. It also shows the ASDoc reference documentation.



```
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:Panel width="250" height="200">
        <
    </s:P
    <mx:D
        <

        <
    </mx:
    <s:Bu
    <s:Ch
</s:Appli
```

```
< > mx:Accordion – mx.containers
< > mx:AccordionHeader – mx.containers.accordic
< > mx:AdvancedDataGrid – mx.controls
< > mx:AdvancedDataGridGroupItemRenderer – m:
< > mx:AdvancedDataGridHeaderRenderer – mx.cc
< > mx:AdvancedDataGridSortItemRenderer – mx.c
< > mx:AdvancedListBase – mx.controls.listClasses
< > mx:Alert – mx.controls
< > s:Application – spark.components
All – Press '^_' to show All Properties
```

An MX Accordion navigator container has a collection of child MX containers or Spark NavigatorContent containers, but only one of them at a time is visible. It creates and manages navigator buttons (accordion headers), which you use to navigate between the children. There is one navigator button associated with each child container, and each navigator button belongs to the Accordion container, not to the child. When the user clicks a navigator

Press 'Tab' or click for focus

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

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

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

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

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

Which code hints to display, and the order for cycling through code hints, is a user preference. To change the default setting, from the Preferences dialog:

1 Open the Preferences Dialog and select Flash Builder > Editors > MXML code > Advanced.

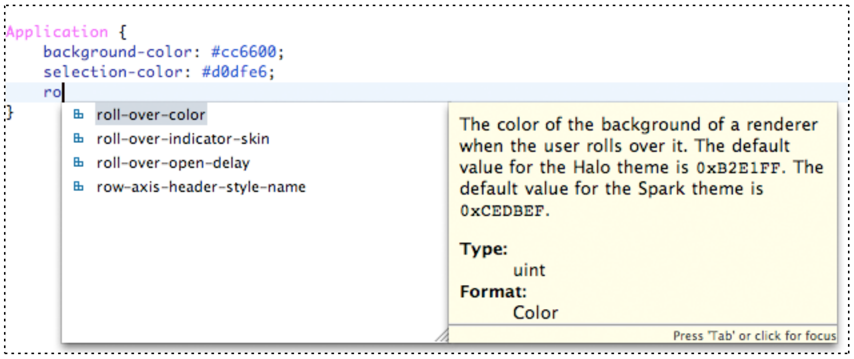
2 Specify which types of hints to display.

3 Select a type of hint, and click Up or Down to change the order of cycling.

   The first type of hint in the list is the initial code hint that is displayed.

Displaying code hints in the CSS editor

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



```
Application {
    background-color: #cc6600;
    selection-color: #d0dfe6;
    ro
}
```

The color of the background of a renderer when the user rolls over it. The default value for the Halo theme is `0xB2E1FF`. The default value for the Spark theme is `0xCEDBEF`.

Type:
uint
Format:
Color

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

Using Content Assist

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

Display Content Assist and insert code hints:

1 Begin entering a line of code.

- In an MXML document, begin entering a tag:

  ```
  <
  ```

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

  Press Control+Space to cycle through available code hints.

- In an ActionScript document or a Script tag in an MXML document, enter a typical language construct:

  ```
  public var myVar:
  ```

- In a CSS document or a Style tag in an MXML document, enter a style name construct and press Control+Space to display a list of attributes that can be added.

  You can also display code hints while you enter a line of code by pressing Control+Space.

2 Navigate the list of code hints with the Up and Down Arrow keys.

3 Select a code hint, press Enter, and the code is added to the editor.

  As you continue to enter code, additional code hints are displayed.

Display Content Assist and view ASDoc comments:

1 Begin entering a line of code that contains either an MXML or ActionScript class. You can also hover over the class.

  As you type, ASDoc content for the class displays next to the code hints. If you hover over a class, just the ASDoc comment displays.

2 Click inside the ASDoc content, or press F2 to display the content in a separate, scrollable window.

3 When you finish reading the documentation, click outside the window. The ASDoc window closes.

Determining call hierarchy

Flash Builder provides a Call Hierarchy view that displays all the callers of a selected ActionScript or MXML function, variable, class, or interface identifier. The list of callers is hierarchical. You can display the callers for each reference found.

For each caller, you can view the line number and line code from the caller. Double-click the caller to open the source file, with the function call or reference highlighted.

You can change the orientation of the displayed callers and specify the scope for the search.

Display the call hierarchy of a language element and open the source file of a caller

1. In Source mode of the MXML editor or in the ActionScript editor, place the insertion point inside a function, variable, class, or interface identifier.
2. From the Navigation menu, select Open Call Hierarchy.

   You can also use the context menu or the keyboard shortcut, Control+Alt+H.
3. In the Call Hierarchy view, expand each caller to view the call hierarchy.
4. Double click a caller to open the calling source file in the editor, with the call highlighted.

Change the orientation and scope for call hierarchy

1. Display the call hierarchy of a language element.
2. From the Call Hierarchy menu, select Layout.

   Choose an option to modify the layout.
3. From the Call Hierarchy menu, select Search Scope.

   Select Workspace, Project, or File. The default value is Workspace.

Generating event handlers for components

You can generate event handlers for Flex components by clicking the Generate Event Handler button for events listed in the Property Inspector. Before generating the event handler, you can specify a custom name for the event handler. If you do not specify a custom name, Flash Builder generates a name based on the id property of the component. If the id property is not defined, Flash Builder generates a unique name derived from the name of the component.

Generate an event handler for a component

1. In Design View of the code editor, select a component.
2. In the Property Inspector, select Category View, expand the list of events for the selected item, and locate the event for which you want to generate an event handler.
3. (Optional) Type a name for the event handler in the Value field.
4. Click the Generate Event Handler button.

   The editor switches to Source View. Flash Builder inserts the event handler into a <Script> block and adds the event property to the component. The event property references the generated event.

   Note: *You can also use the option menu to generate an event handler for common events of a component. For example, for a Button, the option menu specifies Generate Click Handler.*
5. Add the code for your implementation of the event handler to the generated event handler function.

Getting help while writing code

The *Adobe Flex Language Reference* is integrated into the MXML and ActionScript editors and you can quickly review the reference Help for an MXML tag or property, a class, or other Flex framework element.

You can also use Dynamic Help, which is docked next to the current perspective and displays reference and usage topics related to the currently selected MXML tag or ActionScript class.

Display language reference Help

1 In the MXML or ActionScript editor, select a Flex framework element (a word in your code) by highlighting or placing the mouse pointer in the word.

2 To open the Help topic directly in the Help viewer, press Shift+F2 or select Help > Find in Language Reference.

Enable Dynamic Help

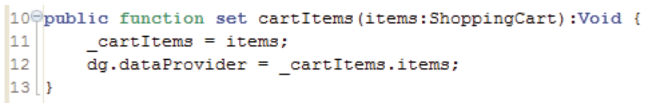
❖ Select Help > Dynamic Help.

Navigating and organizing code

The Flash Builder editors provide many shortcuts for navigating your code, including folding and unfolding code blocks, opening the source of code definitions, and browsing and opening types. Multiple line code blocks can be collapsed and expanded to help you navigate, view, and manage complex code documents. In Flash Builder, expanding and collapsing multiple-line code statements is referred to as *code folding* and *unfolding.*

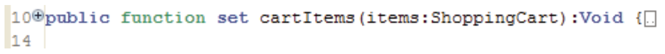
Setting, folding, and unfolding code blocks

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



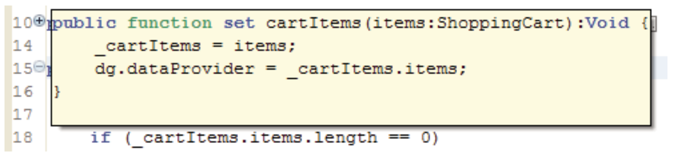
```
public function set cartItems(items:ShoppingCart):Void {
    _cartItems = items;
    dg.dataProvider = _cartItems.items;
}
```

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



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

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



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

    if (_cartItems.items.length == 0)
```

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

Using the Outline view to navigate and inspect code

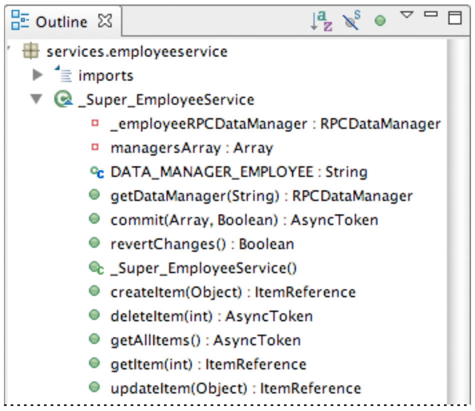
The Outline view is part of the Flash Development perspective (see “The Flash Development perspective” on page 8), and, therefore, is available when you edit code and design your application. You use the Outline view to more easily inspect and navigate the structure of your MXML, ActionScript, and CSS documents.

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

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

Outline view in Class mode

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

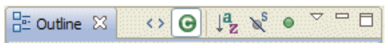


```
services.employeeservice
  imports
  _Super_EmployeeService
      _employeeRPCDataManager : RPCDataManager
      managersArray : Array
      DATA_MANAGER_EMPLOYEE : String
      getDataManager(String) : RPCDataManager
      commit(Array, Boolean) : AsyncToken
      revertChanges() : Boolean
      _Super_EmployeeService()
      createItem(Object) : ItemReference
      deleteItem(int) : AsyncToken
      getAllItems() : AsyncToken
      getItem(int) : ItemReference
      updateItem(Object) : ItemReference
```

In the Outline view, nodes and items in the tree structure represent both the different types of language elements and their visibility. For example, red icons indicate private elements, green indicates public elements, and yellow indicates that the element was neither explicitly marked private nor public.

Outline view toolbar in Class mode

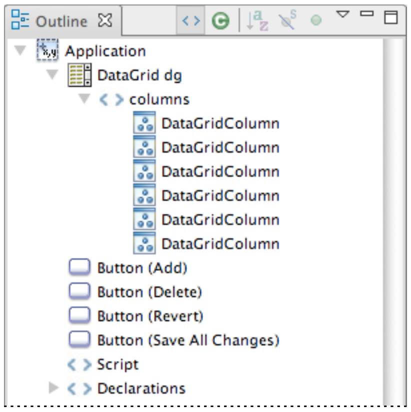
In Class mode, the Outline view toolbar contains the sort and filter commands, as the following example shows:



Outline view in MXML mode

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

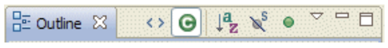
In MXML mode, each item in the Outline view represents an MXML tag and the following types of tags are displayed: components, controls, nonvisual tags such as `WebService` or `State`, component properties that are expressed as child tags (layout constraints, for example), and compiler tags such as `Model`,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



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

Outline view toolbar in MXML mode

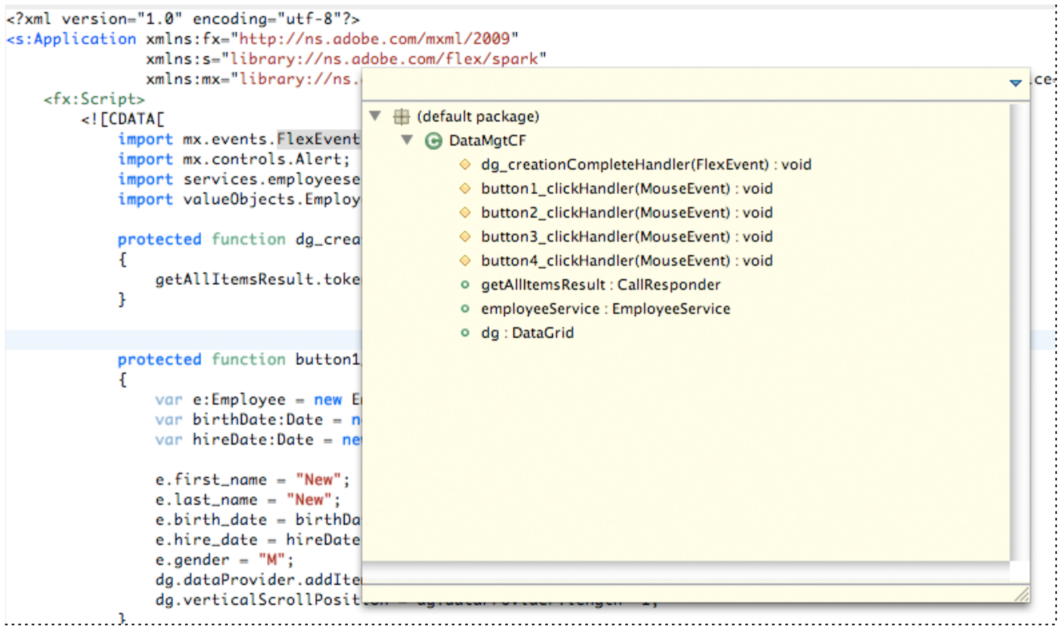
When the Outline view is in MXML mode, the toolbar menu contains additional commands to switch between the MXML and class views.



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

Using Quick Outline view in the editor

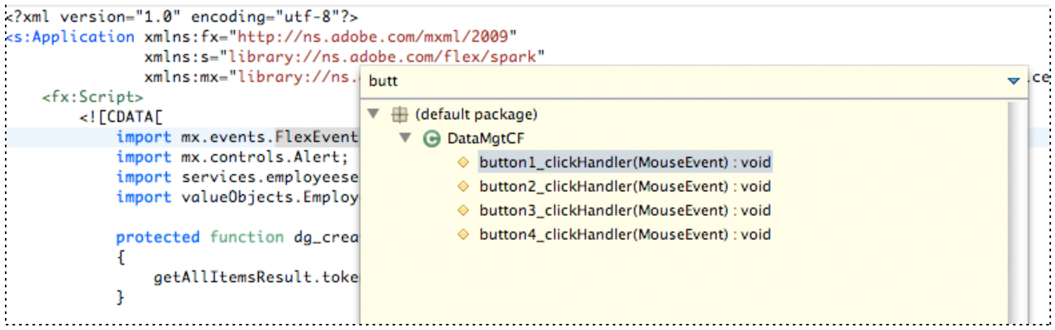
Within the ActionScript and MXML editors, you can access the Quick Outline view, which displays the Outline view in Class mode. The Quick Outline view is displayed in a pop-up window within the editor itself, not as a separate view, and you can use it to quickly navigate and inspect your code.



```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent
            import mx.controls.Alert;
            import services.employeese
            import valueObjects.Employ

            protected function dg_crea
            {
                getAllItemsResult.toke
            }


            protected function button1
            {
                var e:Employee = new E
                var birthDate:Date = n
                var hireDate:Date = ne

                e.first_name = "New";
                e.last_name = "New";
                e.birth_date = birthDa
                e.hire_date = hireDate
                e.gender = "M";
                dg.dataProvider.addIte
                dg.verticalScrollPosit
```

```
▼ (default package)
    ▼ DataMgtCF
        dg_creationCompleteHandler(FlexEvent) : void
        button1_clickHandler(MouseEvent) : void
        button2_clickHandler(MouseEvent) : void
        button3_clickHandler(MouseEvent) : void
        button4_clickHandler(MouseEvent) : void
        getAllItemsResult : CallResponder
        employeeService : EmployeeService
        dg : DataGrid
```

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



```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent
            import mx.controls.Alert;
            import services.employeese
            import valueObjects.Employ

            protected function dg_crea
            {
                getAllItemsResult.toke
            }
```

```
butt
▼ (default package)
    ▼ DataMgtCF
        button1_clickHandler(MouseEvent) : void
        button2_clickHandler(MouseEvent) : void
        button3_clickHandler(MouseEvent) : void
        button4_clickHandler(MouseEvent) : void
```

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

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

Open the Quick Outline view

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

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

Close the Quick Outline view

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

Opening code definitions

With applications of any complexity, your projects typically contain many resources and many lines of code. To help simplify navigating and inspecting the various elements of your code, you can open the source of an external code definition from where it is referred to in your code. For example, if you create a custom MXML component and import it into your MXML application you can select the reference to the MXML component and open the source file in the editor.

Open the source of a code definition

1 Select the code reference in the editor.

2 From the Navigate menu, select Go To Definition.

  You can use the keyboard shortcut, F3.

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

Flash Builder also supports hyperlink code navigation.

Open the source of a code definition using hyperlink navigation

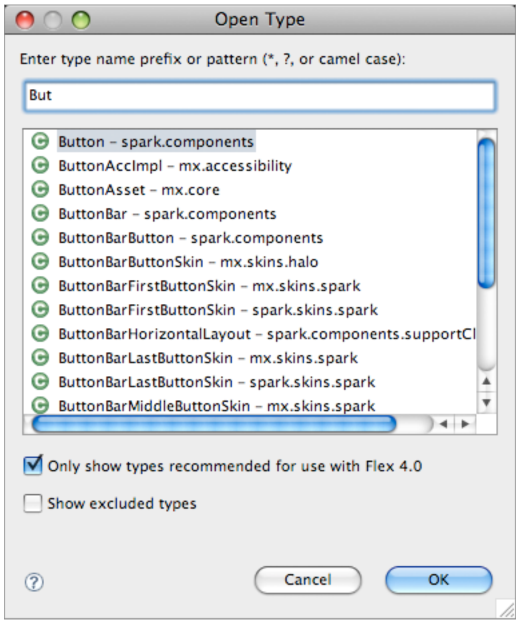
1 Locate the code reference in the editor.

2 Press and hold the Control key (Windows) or Command key (Mac OS) and hold the mouse over the code reference to display the hyperlink.

3 To navigate to the code reference, click the hyperlink.

Browsing and viewing classes

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


Open Type

Enter type name prefix or pattern (*, ?, or camel case):

But

Button – spark.components
ButtonAccImpl – mx.accessibility
ButtonAsset – mx.core
ButtonBar – spark.components
ButtonBarButton – spark.components
ButtonBarButtonSkin – mx.skins.halo
ButtonBarFirstButtonSkin – mx.skins.spark
ButtonBarFirstButtonSkin – spark.skins.spark
ButtonBarHorizontalLayout – spark.components.supportCl
ButtonBarLastButtonSkin – mx.skins.spark
ButtonBarLastButtonSkin – spark.skins.spark
ButtonBarMiddleButtonSkin – mx.skins.spark

☑ Only show types recommended for use with Flex 4.0

☐ Show excluded types

Open Type dialog

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

The Open Type dialog includes a text box to filter the classes that are displayed according to text and wild cards typed in the text box. The dialog uses color coding to indicate recommended types and excluded types. Recommended types display as gray. Excluded types appear brown.

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

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

Open the Open Type dialog

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

     You cannot modify the source code for classes in the Flex framework.
- (New ActionScript classes) When selecting a base class for a new ActionScript class:
  1. Select File > New > ActionScript class.
  2. For the Superclass field, click Browse.
  3. (Optional) Type text or select filters to modify the classes visible in the list.
  4. Select a base class from the list.
- (New MXML components) When selecting a base component for a new MXML component:
  1. Select File > New > MXML Component.
  2. From the list of projects in your workspace, select a project for new MXML component and specify a filename.

     The available base components vary, depending on the namespaces configured for a project.
  3. For the Based On field, click Browse.

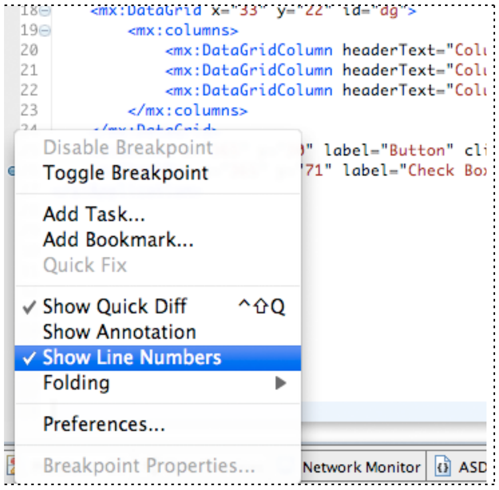
     Note: *Clear or modify the default base class listed in the Based On field to widen your choices.*
  4. (Optional) Type text or select filters to modify the classes visible in the list.
  5. Select a base component from the list.

Showing line numbers

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

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

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



```
<mx:DataGrid x="33" y="22" id="dg">
    <mx:columns>
        <mx:DataGridColumn headerText="Colı
        <mx:DataGridColumn headerText="Colı
        <mx:DataGridColumn headerText="Colı
    </mx:columns>
```

Formatting and editing code

The Flash Builder editors provide shortcuts for writing and formatting your code. These include quickly adding comment blocks, indenting code blocks, finding, and replacing text.

Organizing import statements

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

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

Sort import statements

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

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

Adding comments and comment blocks

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

- Source comment for ActionScript (`//`)
- Block comment for ActionScript (`/* */`)

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

Comments in ActionScript code can be toggled on or off.

Toggle comments in ActionScript code

1 In the editor, select one or more lines of ActionScript code.

2 Press Control+Shift+C (Windows) or Command+Shift+C (Mac OS) to add, or remove, C-style comments.

3 Press Control+/ (Windows) or Command+/ (Mac OS) to add, or remove, C++ style comments.

Add XML comments in MXML code

1 In the editor, select one or more lines of MXML code.

2 Press Control+Shift+C (Windows) or Command+Shift+C (Mac OS) to add a comment.

Add CDATA blocks in MXML code

1 In the editor, select one or more lines of MXML code.

2 Press Control+Shift+D (Windows) or Command+Shift+D (Mac OS) to add a comment.

Indenting code blocks

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

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

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

Shift a code block to the left or right

1 In the editor, select a block of code.

2 Select Source > Shift Right or Source > Shift Left.

3 Press Tab or Shift Tab to indent or unindent blocks of code.

Set indent preferences

1 Open the Preferences dialog and select Flash Builder > Indentation.

2 Select the indent type (Tabs or Spaces) and specify the IndentSize and Tab Size.

Finding and replacing text in the editor

To find and optionally replace text strings in your code, there are two options. You can search the document that is currently open in the editor, or you can search all resources in the projects in the workspace. For more information about searching the entire workspace, see “Finding references and refactoring code” on page 111.

1 Open the document to search.

2 Do either of the following:

- Press Control+F(Windows) or Command+F (Mac OS)

- Select Edit > Find/Replace.

3 Enter the text string to locate.

4 (Optional) Enter the replacement text string.

5 (Optional) Set the advanced search criteria.

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

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

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

Finding references and refactoring code

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

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

Mark references

1 In Source mode, click the Mark Occurrences button on the toolbar.

2 Click an identifier in the editor. All instances are marked, depending on settings in Preferences.

To change the appearance of marked references, in the Preferences dialog, select General > Editors > Text Editors > Annotations. For more information on Markers, see "About markers" on page 115.

Find all references or declarations

1 In Source mode, click on an identifier in the editor.

2 Select Search > References or Search > Declarations from the main menu. Then select File, Project, or Workspace. Matches appear in the Search view.

Refactor your code

1 In Source mode, click on an identifier in the editor.

2 Select Source > Refactor > Rename from the main menu.

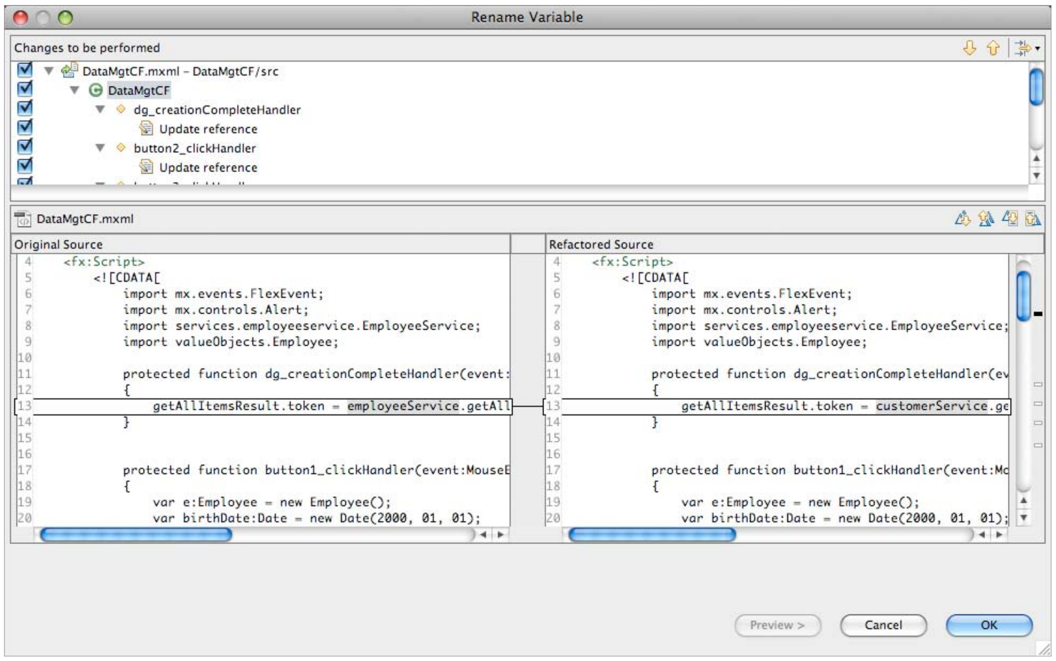
3 Enter a new name.

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

  - References cannot be renamed in read-only files.
  - All files must be saved.

- If a project has build errors, a warning appears.
- The new name must be within scope, which is determined by the type of element and its location. Name-shadowing errors are also noted.
- The new name must be a valid identifier.
- The reference defined in a SWC file must include a source attachment.

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



```
DataMgtCF.mxml - DataMgtCF/src
    DataMgtCF
        dg_creationCompleteHandler
            Update reference
        button2_clickHandler
            Update reference
```

Original Source

```
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.controls.Alert;
            import services.employeeservice.EmployeeService;
            import valueObjects.Employee;

            protected function dg_creationCompleteHandler(event:
            {
                getAllItemsResult.token = employeeService.getAll
            }


            protected function button1_clickHandler(event:MouseE
            {
                var e:Employee = new Employee();
                var birthDate:Date = new Date(2000, 01, 01);
```

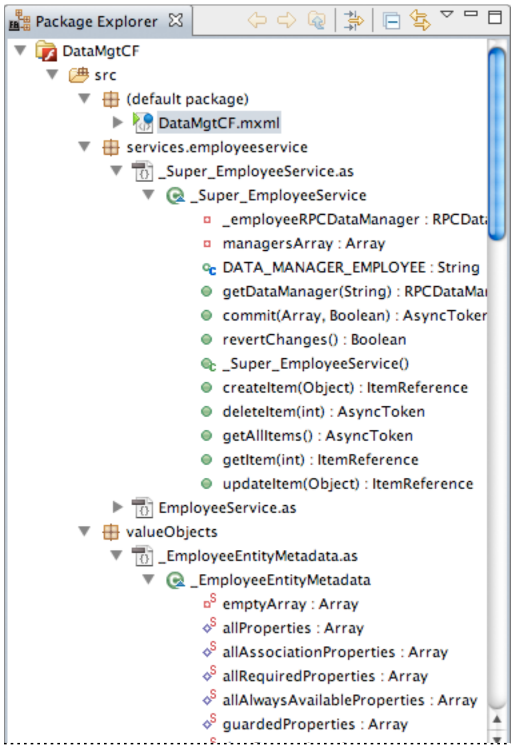
Refactored Source

```
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.controls.Alert;
            import services.employeeservice.EmployeeService;
            import valueObjects.Employee;

            protected function dg_creationCompleteHandler(ev
            {
                getAllItemsResult.token = customerService.ge
            }


            protected function button1_clickHandler(event:Mo
            {
                var e:Employee = new Employee();
                var birthDate:Date = new Date(2000, 01, 01);
```

Icons representing language elements

Flash Builder uses icons and overlays to provide visual cues for language elements. The icons are visible in the following Flash Builder features:

- Package Explorer
- Debugger Variables view
- Code hints in ActionScript code blocks
- ASDoc view
- Open Type dialog when creating or selecting classes



```
Package Explorer
DataMgtCF
  src
    (default package)
      DataMgtCF.mxml
    services.employeeservice
      _Super_EmployeeService.as
        _Super_EmployeeService
          _employeeRPCDataManager : RPCDat
          managersArray : Array
          DATA_MANAGER_EMPLOYEE : String
          getDataManager(String) : RPCDataMa
          commit(Array, Boolean) : AsyncToke
          revertChanges() : Boolean
          _Super_EmployeeService()
          createItem(Object) : ItemReference
          deleteItem(int) : AsyncToken
          getAllItems() : AsyncToken
          getItem(int) : ItemReference
          updateItem(Object) : ItemReference
      EmployeeService.as
    valueObjects
      _EmployeeEntityMetadata.as
        _EmployeeEntityMetadata
          emptyArray : Array
          allProperties : Array
          allAssociationProperties : Array
          allRequiredProperties : Array
          allAlwaysAvailableProperties : Array
          guardedProperties : Array
```

Package Explorer, showing icons for language elements

Icons represent namespaces, classes, interfaces, functions, and variables.

Icon	Shape	Color	Description
Ⓒ	Large Circle	Green	Class
Ⓘ	Large circle	Purple	Interface
Ⓝ	N	Purple	Namespace
○	Circle (outline)	Green	Variable, public scope
●	Circle (solid)	Green	Function, public scope
△	Triangle (outline)	Blue	Variable, internal scope
▲	Triangle (solid)	Blue	Function, internal scope
◇	Diamond (outline)	Yellow	Variable, protected scope
◆	Diamond (solid)	Yellow	Function, protected scope
◇	Diamond (outline)	Purple	Variable, custom namespace

Icon	Shape	Color	Description
◆	Diamond (solid)	Purple	Function, custom namespace
▣	Square (outline)	Red	Variable, private scope
🟥	Square (solid)	Red	Function, private scope

An overlay is a single letter or symbol displayed with an icon that provides additional information about the language element. For example, an overlay of 'D' indicates a dynamic variable or function

Overlay	Symbol	Color	Description
S	S	Red	Static
D	D	Red	Dynamic
C	C	Blue	Constant
C	C	Green	Constructor for a function
F	F	Blue	Final
L	L	Gray	Local
=	=	n/a	Accessor, either a getter or a setter.
●	=	n/a	Example of a public accessor function.

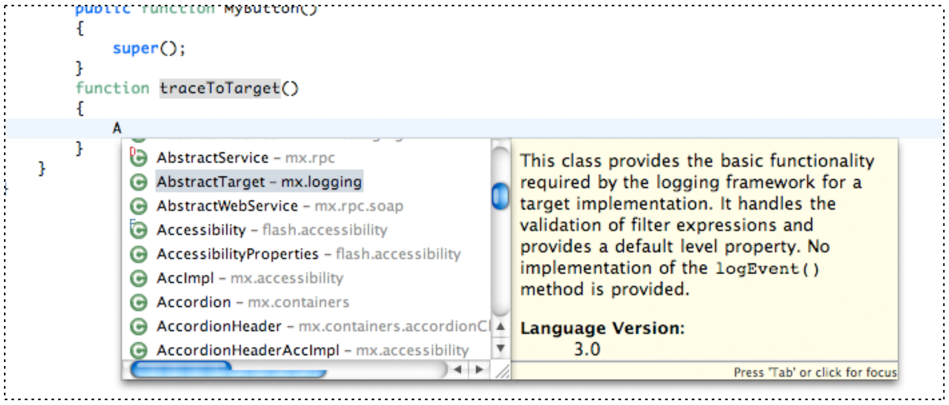
Flash Builder also uses icons to indicate events, effects, and styles.

Icon	Description
⚡	Events
▨	Effects
▦	Styles

Displaying code hints in the ActionScript editor

Content Assist for ActionScript uses icons with code hints to provide visual cues for the type of language elements available.

Content Assist displays code hints for ActionScript 3.0. The code hints are available in the ActionScript editor, in `<fx:Script>` tags in MXML documents, and in event attributes in MXML documents. Content Assist provides hints for all ActionScript 3.0 language elements (such as interfaces, classes, variables, functions, and return types), as the following example shows:



```
    public function MyButton()
    {
        super();
    }
    function traceToTarget()
    {
        A
    }
}
```

AbstractService - mx.rpc
AbstractTarget - mx.logging
AbstractWebService - mx.rpc.soap
Accessibility - flash.accessibility
AccessibilityProperties - flash.accessibility
AccImpl - mx.accessibility
Accordion - mx.containers
AccordionHeader - mx.containers.accordionCl
AccordionHeaderAccImpl - mx.accessibility
Press 'Tab' or click for focus

This class provides the basic functionality required by the logging framework for a target implementation. It handles the validation of filter expressions and provides a default level property. No implementation of the `logEvent()` method is provided.

Language Version:
3.0

About markers

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

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

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

Tasks Task markers represent a work item. Work items are generated automatically by the workbench. You can add a task manually to a specific line of code in a document or to the document itself. For example, to remind yourself to define a component property, you might create a task called “Define skinning properties.” You can also add general tasks that do not apply directly to resources (for example, “Create a custom component for the employee log-in prompt”). You use the Task view to manage all the task markers. For more information, see “Adding tasks” on page 116.

Problems Problem markers are generated by the compiler and indicate invalid states of various sorts. For example, syntax errors and warnings generated by the compiler are displayed as problem markers in the Problem view. For more information, see “Using the Problems view” on page 118.

Bookmarks You can manually add bookmarks to a line of code or a resource (folder or document). You use bookmarks as a convenience, to keep track of and easily navigate to items in your projects. You use the Bookmarks view to manage all bookmarks. For more information, see “Adding and deleting bookmarks” on page 117.

Note: *The Tasks and Bookmarks views are not displayed by default in the Flash Development perspective. For more information about adding these views, see “Opening views” on page 26.*

Navigating markers

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

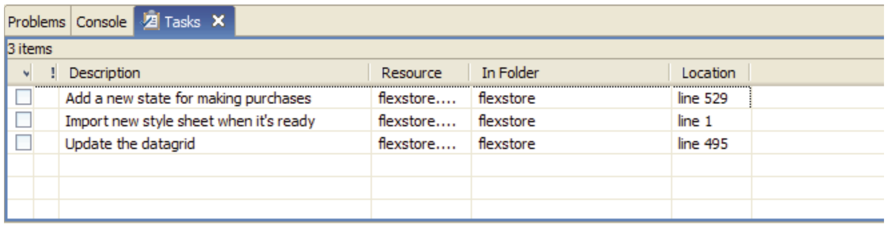
Go to a marker location

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

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

Adding tasks

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



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

Add a task to a line of code or a resource

1 Open a file in the editor, and then locate and select the line of code where you want to add a task; or in the Flex Package Explorer, select a resource.

2 In the Tasks view, click the Add Task button in the toolbar.

3 Enter the task name, and select a priority (High, Normal, Low), and click OK.

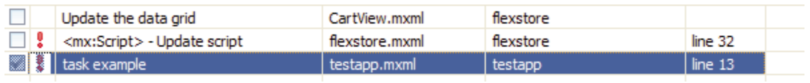
Note: *The resource, as shown in the Flex Package Explorer, does not indicate that it was marked. You can view and manage all task markers in the Task view.*

Completing and deleting tasks

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

Mark a task as complete

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



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

Delete a task

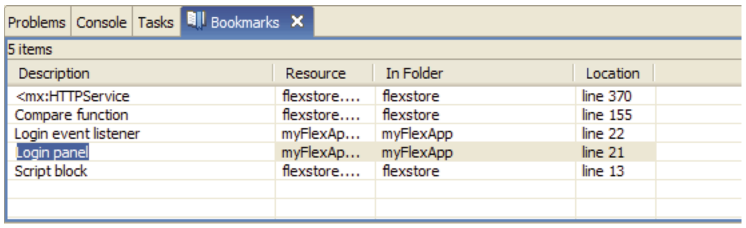
❖ In the Tasks view, open the context menu for a task and select Delete.

Delete all completed tasks

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

Adding and deleting bookmarks

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



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

Add a bookmark to a line of code or a resource

1 Open a file in the editor, and then locate and select the line of code to add a bookmark to.

2 From the main menu, select Edit > Add Bookmark.

3 Enter the bookmark name, and click OK.

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

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

Delete a bookmark

1 In the Bookmarks view, select the bookmark to delete.

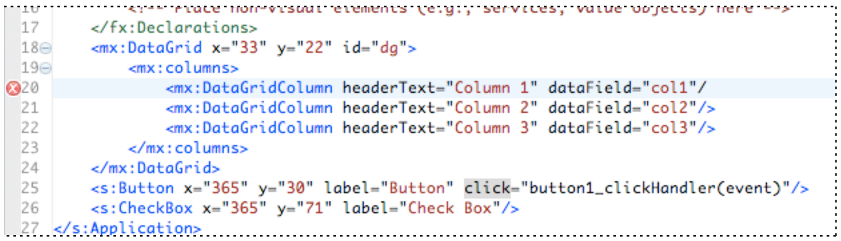
2 Right-click (Windows) or Control-click (Mac OS) the bookmark and select Delete.

About syntax error checking

The Flash Builder compiler identifies syntax errors and reports them to you so that you can correct them as you are working, before you attempt to run your application. You can easily adjust syntax coloring preferences.

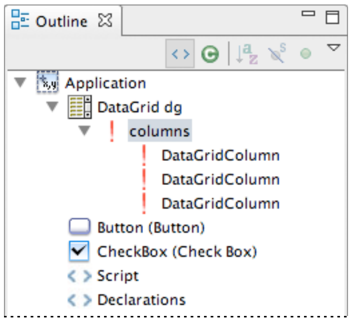
When code syntax errors are encountered, you are notified in the following ways:

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

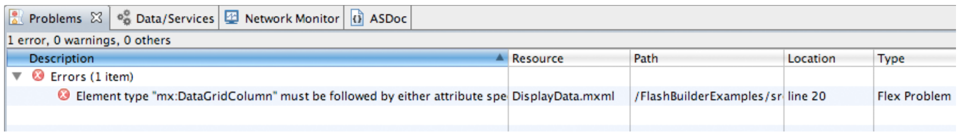


```
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <mx:DataGrid x="33" y="22" id="dg">
        <mx:columns>
            <mx:DataGridColumn headerText="Column 1" dataField="col1"/
            <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
            <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
        </mx:columns>
    </mx:DataGrid>
    <s:Button x="365" y="30" label="Button" click="button1_clickHandler(event)"/>
    <s:CheckBox x="365" y="71" label="Check Box"/>
</s:Application>
```

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



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



1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
Element type "mx:DataGridColumn" must be followed by either attribute spe	DisplayData.mxml	/FlashBuilderExamples/sr	line 20	Flex Problem

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

Apply syntax coloring preferences

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

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

Using the Problems view

As you enter and save your code, Flash Builder compiles it in the background and displays syntax errors and warnings (problems) to you in the Problems view. The Package Explorer marks nodes that contain errors.

Each error or warning contains a message, the file and folder in which it is located, and its line number in the file. Problems remain in the Problems view until you correct them or they are otherwise resolved.

Go to the line of code where an error or warning occurs

❖ Double-click a problem in the Problems view or select Go To from the context menu for the problem.

Flash Builder opens an editor with the offending line of code highlighted.

Code editing keyboard shortcuts

The following table contains a list of keyboard shortcuts that are useful when editing code.

For a complete list of available keyboard shortcuts, see "Accessing keyboard shortcuts" on page 33. For information about editing existing or creating new keyboard shortcuts, see "Changing keyboard shortcuts" on page 30.

Name	**Keyboard shortcut**	**Description**
Switch between Source and Design mode	Control+`(Left Quote)	Switches between the MXML editor's Source and Design modes.
Go to Documentation (Flash Builder plug-in) Find in API Reference (Flash Builder stand-alone)	Shift+F2	When you edit MXML or ActionScript, selecting a language element and pressing Shift+F2 displays language reference Help for the selected element. For more information, see "Getting help while writing code" on page 103.
Context-sensitive Help	F1 (Windows) Command+Shift+/ (Mac OS)	Displays context-sensitive Help for the currently selected workbench element (editor, view, dialog box, and so on).
Add Block Comment	Control+Shift+C (Windows) Command+Shift+C (Mac OS)	Adds block comment formatting to the currently selected lines of code or adds a comment at the insertion point. For more information, see "Adding comments and comment blocks" on page 109.
Add CDATA	Control+Shift+D (Windows) Command+Shift+D (Mac OS)	Adds a CDATA statement at the insertion point so that you can add ActionScript to an MXML document.
Find Matching Bracket	Control+Shift+P (Windows) Command+Shift+P (Mac OS)	Moves the cursor to the matching bracket of the selected code statement.
Content Assist	Control+Space (Windows) Command+Shift+Space (Mac OS)	Displays code hinting. For more information, see "Using Content Assist" on page 101.
Find All Declarations in Workspace	Control+G (Windows) Command+G (Mac OS)	Finds declarations in your code base. See "Finding references and refactoring code" on page 111.
Find All References in Workspace	Control+Shift+G (Windows) Command+Shift+G (Mac OS)	Finds references to identifiers in your code base. See "Finding references and refactoring code" on page 111
Go to Definition	F3	Open the source of an external code definition. For more information, see "Opening code definitions" on page 107.
Go to Line	Control+L (Windows) Command+L (Mac OS)	Displays the Go to Line dialog box where you can enter a line number and navigate to it in the editor.
Last Edit Location	Control+Q (Windows) Control+Q (Mac OS)	Highlights the last edited line of code.
Mark Occurrences	n/a	Marks every occurrence of the selected item in code.
Organize Imports	Control+Shift+O (Windows) Command+Shift+O (Mac OS)	When editing ActionScript, using this keyboard shortcut alphabetizes any import statements contained in the document. For more information, see "Organizing import statements" on page 109.
Open Type	Control+Shift+T (Windows) Command+Shift+T (Mac OS)	Quickly browse all class types. For more information, see "Browsing and viewing classes" on page 107.

Name	Keyboard shortcut	Description
Open Resource	Control+Shift+R (Windows) Command+Shift+R (Mac OS)	Displays the Open Resource dialog box where you can quickly search for and open a resource in the editor.
Quick Outline	Control+O (Windows and Mac OS)	Displays the Outline view in Quick mode in the editor. For more information, see "Using Quick Outline view in the editor" on page 106.
Toggle Comment	Control+/ (Windows) Command+/ (Mac OS)	Toggles ActionScript comments in ActionScript code. See "Adding comments and comment blocks" on page 109.
Toggle BlockComment	Shift+Control+C Shift+Command+C	Toggle MXML comment blocks. See "Adding comments and comment blocks" on page 109.
Toggle Folding	Control+Numpad_divide	Toggles the folding of code blocks. See "Setting, folding, and unfolding code blocks" on page 103.

Customizing File Templates

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

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

ActionScript	ActionScript file ActionScript class ActionScript interface

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

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

Modify a file template

1 Select Preferences > Flash Builder > File Templates

2 Expand the file categories and select a file template to modify.

3 Select Edit and modify the template.

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

4 Click OK to save the changes.

   Changes apply to new files.

Exporting and Importing File Templates

1 Select Preferences > Flash Builder > File Templates

2 Expand the file categories and select a file template.

3 Select Export to export a template to the file system, or Import to import a previously exported template.

   Templates are exported as XML files.

Restoring Defaults

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

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

Template Variables

Template Variables for All File Types

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

Template Variables for MXML Files

Variable	Description	Example
${application} ${component} ${module}	Specifies the application, component, or module MXML tag names. For a web application, ${application} expands to "Application." For a desktop application, ${application} expands to "WindowedApplication." ${component} expands to "Component." ${module} expands to "Module." These tags are typically used to position the starting and closing tags of a file.	The following: `<${application} ${xmlns}${wizard_attributes}${min_size}> ${wizard_tags}` `</${application}>` expands to: `<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009" xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024" minHeight="768"> <s:layout> <s:BasicLayout/> </s:layout>` `</s:Application>`
${xml_tag}	XML version	`<?xml version="1.0" encoding="utf-8"?>`
${xmlns}	Resolves to the namespace definition, based on the project's Flex SDK type and the namespace prefix defined in Preferences.	For a Flex 4 SDK project: xmlns="http://ns.adobe.com/mxml/2009"
${min_size}	Minimum size of an MXML web application.	`minWidth="1024" minHeight="768"`

Variable	**Description**	**Example**
${ns_prefix}	Namespace prefix for the project's Flex SDK. You cannot change the default values for this variable.	For Flex 3: `mx:` For Flex 4: `fx:`
${wizard_attributes}	Specifies the position of the attributes defined by the New *File* wizard.	For a new web application: ${application} ${xmlns}${wizard_attributes}> expands to: <Application xmlns="http://ns.adobe.com/mxml/2009" layout="vertical">
${wizard_tags}	Specifies the layout property for containers defined by the New *File* wizard	For a new application using Flex 4 SDK: <s:layout> <s:BasicLayout/> </s:layout>

Template Variables for ActionScript Files

Variable	**Description**	**Example**
${package_declaration}	Generates the package declaration.	For a file in the com/samples package, generates: package com.samples
${import_declaration}	For a new ActionScript class or ActionScript Interface, generates required import declarations .	For a subclass of TextBox, generates: import flex.graphics.TextBox;
${interface_declaration}	For a new ActionScript interface, generates the interface declaration.	For a new Interface that extends IButton interface, generates: public interface IMyButton extends IButton
${class_declaration}	For a new ActionScript class, generates the class declaration.	For a new subclass of CheckBox, generates: public class MyCheckBox extends CheckBox
${class_body}	Generates all the required statements for a new class.	For a new subclass of Button that implements the IBorder interface, generates the following for the class body: `public function MyButton()` `{` `super();` `}` `public function get borderMetrics():EdgeMetrics` `{` `return null;` `}`
${interface_name} ${class_name} ${package_name}	Specifies the interface, class, or package name. Typically used when generating comments.	For example, the following template specification: `/*` `* ${class_name} implements. . .` `*/` generates the following code: `/*` `* MyButton implements. . .` `*/`

Template Variable for CSS Files

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

Template File Example

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

Example File Template for an MXML Component file

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

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

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

New MXML Component file generated from the example template

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

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

    <fx:Script>
        <![CDATA[

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

Chapter 5: Creating Custom MXML Components

An application in Adobe® Flex® typically consists of an MXML application file (a file with an `<s:Application>` parent tag), one or more standard Flex components, and one or more custom components defined in separate MXML, ActionScript, or Flash component (SWC) files. By dividing the application into manageable chunks, you can write and test each component independently from the others. You can also reuse a component in the same application or in other applications, which increases efficiency.

You can use Adobe Flash® Builder™ to build custom MXML and ActionScript components and then insert them into your applications. For information on building ActionScript components, see "Creating an ActionScript class" on page 64.

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

About custom components

You might want to create custom components for a number of reasons. For example, you might simply want to add some functionality to an existing component, or you might want to build a reusable component, like a search box or the display of an item in a data grid. You might want to write a completely new kind of component that isn't available in the Flex framework.

If your component is composed mostly of existing components, it is convenient to define it using MXML. However, if it is a completely new kind of component, you should define it as an ActionScript component. For more information, see "Creating an ActionScript class" on page 64.

Creating MXML components using Flash Builder

You use Flash Builder to create custom MXML components.

1 Select File > New > MXML Component.

The New MXML Component dialog box appears:

2 Specify the parent folder for your custom component file.

Save the file in a folder in the current project folder or in the source path of the current project if you want the component to appear in the Components view.

3 Specify a filename for the component.

The filename defines the component name. For example, if you name the file `LoginBox.mxml`, the component is named LoginBox.

4 In the Based On field, select the base component of your custom component.

Custom components are typically derived from existing components. Containers are commonly used as the base components of layout custom components.

For Flex 4, Flash Builder suggests **spark.components.Group** as the base component.

Select Browse to open the Open Type dialog and select a component.

Modify or clear the suggested component to broaden the selection in the Open Type dialog. For example, specify **spark.components.** before clicking Browse.

You can filter the selection in the Open Type dialog, based on your needs. See “Browsing and viewing classes” on page 107 for information on using the Open Type dialog.

5 (Optional) If you base the component on any container, you get options to set the width and height of the component.

You can set these options to a fixed width and height or to percentages, or you can clear them. When you create an instance of the component, you can override the component’s width and height in the instance.

If you set a percentage width and height or if you set no width and height, you can preview how the component looks at different sizes using the Design Area pop-up menu in the toolbar of the MXML editor’s toolbar in Design mode. For more information, see “Designing components visually” on page 127.

6 Click Finish.

Flash Builder saves the file in the parent folder and opens it in the editor.

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

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

7 Create your custom component.

For more information, see Simple MXML Components.

Designing components visually

You can lay out a custom component visually in the MXML editor as you would a regular MXML application file. All the tools in Design mode are available. For example, you can add child controls from the Components view and set properties in the Properties view. For more information, see “Working with components visually” on page 184.

The size of a custom component displayed in Design mode is determined by the following rules:

- If your component has a fixed width and height, Flash Builder automatically sets the design area to that width and height.
- If the component has no width and height, or has a 100% width and height, you can select the size of the design area from the Design Area pop-up menu in the editor’s toolbar.

  Selecting different sizes allows you to preview the component as it might appear in different circumstances. The design area defaults to 400x300 pixels for containers and to Fit to Content for controls.
- If the component has a percentage width and height other than 100%, Flash Builder renders it as a percentage of the selected size in the Design Area menu.

Providing ASDoc comments for custom components

You can document your custom components by adding ASDoc comments to the code that implements the components. ASDoc comments are then available with Content Assist in the MXML and ActionScript editors. For more information, see "Using Content Assist" on page 101.

Add ASDoc comments to ActionScript source files to provide API reference documentation. You can also add ASDoc comments to document MXML elements. See ASDoc for details on creating ASDoc comments for your source files.

Editing and distributing custom MXML components

Flash Builder renders any visible custom components (components that inherit from UIComponent) in the MXML editor's Design mode. You can double-click a custom component in the layout to open its file and edit it.

1 Open an MXML file that uses a custom component.

2 In Design mode, double-click a custom component in the layout.

   The component file opens in the editor.

   You can also select Open Custom Component from the context menu.

3 Edit the custom component.

Distributing custom components

Distribute custom components by creating library projects. For more information, see "Library projects" on page 66.

Chapter 6: Developing AIR applications with Flash Builder

Adobe® Flash® Builder™ provides you with the tools to create Adobe® AIR® projects, work with the Flex AIR components, and debug and package Adobe AIR applications. The workflow for developing AIR applications in Flash Builder is similar to that for developing most Flex applications.

Creating AIR projects with Flash Builder

If you have not already done so, download and install AIR runtime.

1 Open Flash Builder.

2 Select File > New > Flex Project.

3 Enter the project name.

4 In Flex, AIR applications are considered an application type. You have two type options: an application that runs on the Web in Adobe® Flash® Player and a desktop application that runs in Adobe AIR. Select Desktop Application as the application type.

5 Select the server technology (if any) that you want to use with your AIR application. If you're not using a server technology, select None and then click Next.

6 Select the folder in which you want to place your application. The default is the bin-debug folder. Click Next.

7 Modify the source and library paths as needed and then click Finish to create your AIR project.

Converting Flex projects to Adobe AIR projects

Flex projects specify an application type of either Web (runs in Adobe Flash Player) or Desktop (runs in Adobe AIR). You can convert the application type of a Flex project from Web to Desktop. See "Changing Flex projects to Adobe AIR projects" on page 48.

Note: *If you import a Catalyst project from an FXP file, Flash Builder imports the project with the Web application type. You can convert the project to a Desktop application type (runs in Adobe AIR).*

Debugging AIR applications with Flash Builder

Flash Builder provides full debugging support for AIR applications. For more information about the debugging capabilities of Flash Builder, refer to "Debugging your applications" on page 132.

1 Open a source file for the application (such as an MXML file) in Flash Builder.

2 Click the Debug button on the main toolbar .

You can also select Run > Debug.

The application launches and runs in the ADL application (the AIR Debugger Launcher). The Flash Builder debugger catches any breakpoints or runtime errors and you can debug the application like any other Flex application.

You can also debug an application from the command line, using the AIR Debug Launcher command-line tool. For more information, see *Using the AIR Debug Launcher (ADL)* in the AIR documentation.

Packaging AIR applications with Flash Builder

When your application is complete and ready to be distributed (or tested running from the desktop), you package it into an AIR file. Packaging consists of the following steps:

- Selecting the AIR application you want to publish
- Optionally allowing users to view the source code and then selecting which of your application files to include
- Digitally signing your AIR application using a commercial code signing certificate or by creating and applying a self-signed signature
- Optionally choosing to create an intermediate AIR file, which can be signed at a later time

Package an AIR application

1 Open the project and ensure that the application has no compilation errors and runs as expected.

2 Select Project > Export Release Build.

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

4 Optionally select Enable View Source if you want users to be able to see the source code when they run the application. You can select individual files to exclude by selecting Choose Source Files. By default all the source files are selected. For more information about publishing source files in Flash Builder, see the Flash Builder Help.

5 You can also optionally change the name of the AIR file that is generated. When you're ready to continue, click Next to digitally sign your application.

Digitally signing your AIR applications

Before continuing with the Export Release Version, decide how you want to digitally sign your AIR application. You have several options. You can sign the application using a commercial code signing certificate, you can create and use a self-signed digital certificate, or you can choose to package the application now and sign it later.

Digital certificates issued by certification authorities such as VeriSign, Thawte, GlobalSign, and ChosenSecurity assure your users of your identity as a publisher and verify that the installation file has not been altered since you signed it. Self-signed digital certificates serve the same purpose but they do not provide validation by a third party.

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

For more information about signing applications, see *Digitally Signing an AIR File* in your AIR documentation.

Digitally sign your AIR application

You can digitally sign your AIR application by selecting an existing digital certificate or by creating a new self-signed certificate.

1 Select Project > Export Release Build.

  Select the AIR project to export and the file to export the project to. Click Next.

2 Select the Export and Sign an AIR File with a Digital Certificate option.

3 If you have an existing digital certificate, click Browse to locate and select it.

4 To create a new self-signed digital certificate, select Create.

5 Enter the required information and click OK.

6 (Optional) Click Next. Select output files to include in the exported AIRI file.

  By default, all the files are included.

7 Click Finish to generate the AIR file.

Create an intermediate AIR file

You can create an intermediate AIRI file that can be signed later. Use this option for testing only.

1 Select Project > Export Release Build.

  Select the AIR project to export and the file to export the project to. Click Next.

2 Select Export an Intermediate AIRI File that will be Signed Later option.

3 (Optional) Click Next. Select output files to include in the exported AIRI file.

  By default, all the files are included.

4 Click Finish.

  After you have generated an intermediate AIR file, it can be signed using the AIR Developer Tool (ADT). For information on the ADT command line tool, see *Signing an AIR Intermediate File with ADT* in your AIR documentation.

Create an AIR library project

To create an AIR code library for multiple AIR projects, create an AIR library project using the standard Flex library project wizard.

1 Select File > New > Flex Library Project.

2 Specify a project name.

3 Select Include Adobe AIR Libraries and then click Next.

  Note: *The Flex SDK version you select must support AIR. The Flex 2.0.1 SDK does not.*

4 Modify the build path as needed and then click Finish. For more information about creating library projects, see "About library projects" in the Flash Builder Help.

Chapter 7: Debugging, testing, and monitoring applications

Debugging your applications

Debugging is similar to running your applications. However, when you debug you control when the application stops at specific points in the code and whether you want it to monitor important variables, and you can test fixes to your code. Both running and debugging use a configuration to control how applications are launched. When you debug your application, you run the debug version of the application file.

For an overview of the debugging tools available in the Flash Debug perspective, see "The Flash Debug perspective" on page 17.

In some cases, you will be prompted to look at the Eclipse log file. For more information, see "Eclipse environment errors in the log file" on page 72.

Starting a debugging session

To begin a debugging session, you run the application launch configuration in the Flash Debug perspective.

Debug an application

1 In the Flex Package Explorer, select the project to debug.

2 Select the Debug button from the main workbench toolbar.

   ***Note:** The Debug button has two elements: the main action button and a drop-down list that shows the application files in the project that can be run or debugged. When you click the main action button, the project's default application file is debugged. You can alternatively click the drop-down list and select any of the application files in the project to debug. You can also access the launch configuration dialog box and create or edit a launch configuration by selecting the Debug command.*

   If your project has not been built yet, Adobe® Flash® Builder™ builds and runs it in debug mode.

3 Your application appears in your default web browser or the stand-alone Flash Player and you can then use the Flash Builder debugger to interact with it.

4 When a breakpoint is reached, the Flash Debug perspective is activated in the workbench.

Start a debugging session in the plug-in configuration

The Debug command works differently in the plug-in configuration of Flash Builder. Instead of running the selected project, it debugs the most recently launched configuration. You can also select from a list of recently launched configurations.

Adding and removing breakpoints

You use breakpoints to suspend the execution of your application so you can inspect your code and use the Flash Builder debugging tools to explore options to fix errors. You add breakpoints in the code editor and then manage them in the Breakpoints view when you debug your applications.

You add breakpoints to executable lines of code. The debugger stops only at breakpoints set on lines that contain the following:

- MXML tags that contain an ActionScript event handler, such as `<mx:Button click="`*dofunction*`()" ...>`
- ActionScript lines such as those enclosed in an `<mx:Script>` tag or in an ActionScript file
- Any executable line of code in an ActionScript file

You can set breakpoints as you write code or while you debug.

Set a breakpoint in the code editor

1 Open a project file that contains ActionScript code.

2 Locate the line of code on which you want to set a breakpoint, and double-click in the marker bar to add a breakpoint.

  The marker bar is along the left edge of the code editor.

  A breakpoint marker is added to the marker bar and to the list of breakpoints in the Breakpoints view of the Flash Debug perspective.

When the debugger encounters a breakpoint, the application is suspended, the Flash Debug perspective is displayed, and the line of code is marked with a breakpoint. The line of code is highlighted in the code editor. You then use the debugging commands to interact with the code. (See “Managing the debugging session in the Debug view” on page 135).

Remove a breakpoint in the code editor

❖ In the marker bar, double-click an existing breakpoint.

  The breakpoint is removed from the marker bar and the Breakpoints view of the Flash Debug perspective.

You manage breakpoints in the Breakpoints view. You can remove one or all breakpoints in the list or disable them and re-enable them at a later time (see “Managing breakpoints in the Breakpoints view” on page 134).

Setting Conditional Breakpoints

You can specify conditions for breakpoints to stop the debugger from executing when specific conditions are met. When you set a conditional breakpoint, you specify an ActionScript expression that is evaluated during the debugging session. You configure the conditional breakpoint to halt execution for any of the following conditions:

- The expression evaluates to true.
- The value of the expression changes.
- A specified Hit Count has been reached.

Setting a conditional breakpoint

1 From the context menu for a breakpoint, select Breakpoint Properties.

2 In the Breakpoint Properties dialog, specify any of the following:

  - Enabled

    Toggle to enable or disable the breakpoint.

  - Hit Count

    Select Hit Count to enable a counter for the breakpoint. Specify a number for the Hit Count.

If you specify both Hit Count and Enable Condition, the Hit Count is the number of times that the specified condition is met (either evaluates to true or the value of the condition changes).

If you specify Hit Count only, then Hit Count is the number of times the breakpoint has been reached.

- Enable Condition

  Select Enable Condition and enter an ActionScript expression to evaluate. See "Examples of expressions" on page 137 for information on types of expressions supported for evaluation.

  Note: *Flash Builder checks the syntax of the expression and notifies you of syntax errors. If you have an assignment operator in the expression, Flash Builder displays a warning.*

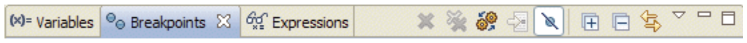
- Suspend when:

  Specify when to halt execution, either when the expression for the condition evaluates to true or the value of the expression changes.

Managing breakpoints in the Breakpoints view

In the Breakpoints view you manage breakpoints during a debugging session. You can remove, disable and enable, or skip them.

The following commands are available from the Breakpoints view toolbar (as shown left to right):



Button/Command	Description
Remove Selected Breakpoints	Removes the selected breakpoints.
Remove All Breakpoints	Removes all breakpoints.
Show Breakpoints Supported by Selected Target	Displays breakpoints that are applicable to the select debug target.
Go to File for Breakpoint	Opens the file (if it is not already open) that contains the breakpoint in the code editor and highlights the line of code on which the breakpoint was set. You can also simply double-click the breakpoint to display it in the code editor.
Skip All Breakpoints	Skips all breakpoints.
Expand All	Expands all breakpoints.
Collapse All	Collapses all breakpoints.
Link With Debug View	Links to Debug view.

Remove breakpoints in the Breakpoints view

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

❖ Select one or more breakpoints from the list of breakpoints, and then click Remove Selected Breakpoints.

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

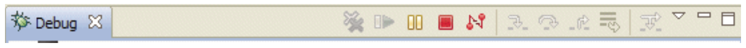
Remove all breakpoints from the Breakpoints view

❖ In the Breakpoints view, click Remove All Breakpoints.

Managing the debugging session in the Debug view

The Debug view is the control center of the Flash Debug perspective. You use it to control the execution of the application, to suspend, resume, or terminate the application, or to step into or over code.

The Debug view provides the following debugging commands, which are available from the Debug view toolbar (as shown left to right):



Button/Command	Description
Remove All Terminated Launches	Clears all terminated debugging sessions.
Resume	Resumes the suspended application.
Suspend	Suspends the application so that you can inspect, step into the code, and so on.
Terminate	Terminates the debugging session.
Disconnect	Disconnects the debugger when debugging remotely.
Step Into	Steps into the called function and stops at the first line of the function.
Step Over	Executes the current line of the function and then stops at the next line of the function.
Step Return	Continues execution until the current function has returned to its caller.
Drop to Frame	This command is not supported in Flash Builder.
Use Step Filters	This command is not supported in Flash Builder.

Using the Console view

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

The Console view provides the following commands, which are available from the Console view toolbar (as shown left to right):

Button/Command	Description
Terminate	Terminates the debugging session.
Remove Launch	Clears all launched debugging sessions.
Remove All Terminated Launches	Clears all terminated debugging sessions.
Clear Console	Clears all content from the Console view.
Scroll Lock	Prevents the Console view from scrolling.
Show Console When Standard Out Changes	Displays the Console when writing to standard out
Show Console When Standard Error Changes	Displays the Console when writing to standard error

Button/Command	Description
Pin Console	Prevents the console from refreshing its contents when another process is selected.
Display Selected Console	Displays the selected Consoles
Open Console	Opens new console and displays pop-up menu to select other console views.

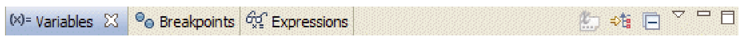
Managing variables in the Variables view

The Variables view displays the variables that the currently selected stack frame defines (in the Debug view). Simple variables (name and value) are displayed on a single line. The icons displayed with the variables provide visual cues about the type of variable.

Complex variables can be expanded to display their members. You use the Variables view to watch variables by adding them to the Expressions view and to modify the value of variables during the debugging session. You can also set watchpoints in the Variables view, as described in "Using Watchpoints" on page 139.

All superclass members are grouped in a separate tree node; by default you see only the members of the current class. This helps reduce excess numbers of variables that are visible at one time in Variables view.

The Variables view provides the following actions, which are available from the Variables view toolbar (as shown left to right):



Command	Description
Show Type Names	Displays the type names of variables.
Show Logical Structure	This command is not supported in Flash Builder.
Collapse All	Collapses the Variables view.

Change the value of a variable

1 Select the variable to modify.

2 Right-click (Control-click on Macintosh) to display the context menu and select Change Value.

3 Enter the new value and click OK.

The variable contains the new value.

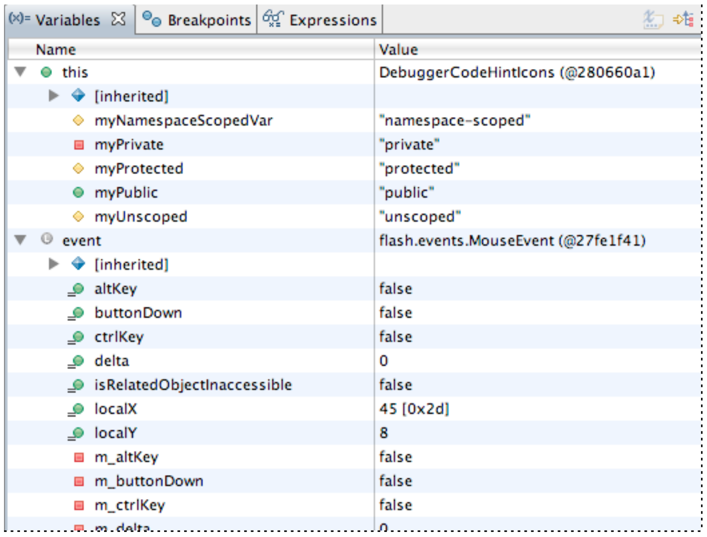
Modified variables are displayed in red.

Find variables

❖ To locate a variable or variable member in the Variables view, with the Variables view selected, begin entering the name of the variable you're looking for. You can also use the wildcard character (*) to search for words that occur anywhere within a variable name (for example, "*color").

Icons representing variables in the Variables view

The Variables view uses icons and overlays to provide visual cues to the types of variables that are being displayed. See "Icons representing language elements" on page 112 for a list of icons and their meaning.



Name	Value
this	DebuggerCodeHintIcons (@280660a1)
[inherited]	
myNamespaceScopedVar	"namespace-scoped"
myPrivate	"private"
myProtected	"protected"
myPublic	"public"
myUnscoped	"unscoped"
event	flash.events.MouseEvent (@27fe1f41)
[inherited]	
altKey	false
buttonDown	false
ctrlKey	false
delta	0
isRelatedObjectInaccessible	false
localX	45 [0x2d]
localY	8
m_altKey	false
m_buttonDown	false
m_ctrlKey	false
m_delta	0

The Variables view

Using the Expressions view

Use the Expressions view to watch variables you selected in the Variables view and to add and evaluate watch expressions while debugging your applications.

While debugging, you can inspect and modify the value of the variables that you selected to watch. You can also add watch expressions, which are code expressions that are evaluated whenever debugging is suspended. Watch expressions are useful for watching variables that may go out of scope when you step into a different function and are therefore not visible in the view.

The Expressions view provides the following commands, which are available from the Expressions view toolbar (as shown left to right):

Command	**Description**
Show Type Names	Shows the object types for items in the Expressions view.
Show Logical Structure	This command is not supported in Flash Builder.
Collapse All	Collapses all expressions in view.
Remove Selected Expressions	Removes the selected variable or watch expression.
Remove All Expressions	Removes all variables and watch expressions from the Expressions view.

You can also hover the mouse pointer over an expression or variable in the source editor to see the value of that expression or variable as a tooltip. You can add the expression to the Expressions view by right-clicking and selecting Watch from the menu.

Examples of expressions

The Flash Builder Debugger supports a wide range of simple and complex expressions. The following table lists examples of expressions that can be evaluated during a debugging session. This is not the complete list of expressions supported, but just a sampling of what you can do.

Examples of supported expressions

Expression	Description
myString.length	Returns the length of a string.
myString.indexOf('@')	Tracks the index of the '@' character.
"constant string".charAt(0)	Tracks the character at a specific position in a string. String constants are supported.
employees.employee.@name	employees is an XML variable. This type of expression is useful for debugging E4X applications.
x == null	Reserved words representing values in expressions.
user1 === user2	Most ActionScript operators are supported.
MyClass.myStaticFunc()	Functions resolved to a class.
this.myMemberFunc()	Functions resolved using the keyword `this`.
String.fromCharCode(33)	String is actually a function, not a class, and String.fromCharCode is actually a dynamic member of that function.
myStaticFunc()	Can be valuated only if myStaticFunc is visible from the current scope chain
myMemberFunc()	Can be valuated only if myMemberFunc is visible from the current scope chain.
Math.max(1,2,3)	Math functions are supported.
mystring.search(/myregex/i)	Regular expressions are supported.
["my", "literal", "array"]	Creation of arrays.
new MyClass()	Instantiation of classes.
"string" + 3	Correctly handles string plus Integer.
x >>> 2	Logical shift operations supported.
3.5 + 2	Performs arithmetic operations correctly.

Limitations of expression evaluation

There are some limitations to expression evaluation.

- Namespaces are not supported.
- Inline objects are not supported.
- The keyword `super` is not supported.
- Fully qualified class names are not supported.

  For example, you cannot evaluate mx.controls.Button.

  You can refer to the unqualified class name. For example, you can specify Button to refer to mx.controls.Button.

  If a class name is ambiguous (two classes with the same name in different packages,) then you cannot control which class will be evaluated. However, you can specify:

  ```
  getDefinitionByName("mx.controls.Button")
  ```

- Most E4X expressions can be evaluated, but E4X filter expressions are not supported.

  For example, you cannot evaluate `myxml.(@id=='3'))`.

- You cannot call functions that are defined as a variable.

Using Watchpoints

When debugging an application, you can set watchpoints on specific instances of variables to halt execution when the watched variable changes value. Because watchpoints are set on a specific instance of a variable, you cannot set the watchpoint in the code editor. Instead, you set a watchpoint from the Variables view during a debugging session.

When setting watchpoints, keep in mind the following:

- When a debugging session ends, all watchpoints are removed.
- You cannot set watchpoints on getters, but you can set them on the field of a getter.

  For example, you cannot set a watchpoint on `width`, but you can set a watchpoint on `_width`.
- You cannot set watchpoints on local variables, but you can set watchpoints on members of local variables, as illustrated in the following code fragment.

```
public class MyClass
{
   // These are fields of a class, so you can set a watchpoint on
   // 'memberInt', and on 'memberButton', and on 'memberButton._width':
   private var memberInt:int = 0;
   private var memberButton:Button = new Button();

   public function myFunction():void {
      // You CANNOT set a watchpoint on 'i', because it is local:
      var i:int = 0;

      // You CANNOT set a watchpoint on 'someButton', because it is local;
      // but you CAN set a watchpoint on 'someButton._width':
      var someButton:Button = new Button();

...
}
```

- Execution halts for a watchpoint when the original value of an object instance changes.

  This differs from using an expression in a conditional breakpoint to halt execution whenever a variable changes value.

Setting watchpoints

❖ In a debugging session, there are two ways to set a watchpoint:

- In the Variables view, open the context menu for a variable, and select Toggle Watchpoint
- From the Flash Builder Run menu, select Add Watchpoint.

  From the Add Watchpoint dialog, select the variable you want to watch.

The Variables view displays a “pencil icon” to indicate that a watchpoint has been set on that variable.

Note: *If you attempt to set a watchpoint on a getter, Flash Builder opens a dialog suggesting a valid variable for the watchpoint. If you delete the suggested variable, the dialog lists all valid variables for the object.*

Using Run to Line

Flash Builder provides the Run to Line command to break out of a loop during a debugging session.

While debugging, you might find that your code is executing a loop that repeats many times. To break out of this loop, use the Run to Line command, available from the Run menu.

FlexUnit test environment

The FlexUnit test environment allows you to generate and edit repeatable tests that can be run from scripts or directly within Flash Builder. Flash Builder supports both FlexUnit 4 and Flex Unit 1 open source frameworks.

From Flash Builder, you can do the following:

- Create unit test cases and unit test suites

  Flash Builder wizards guide you through the creation of test cases and test suites, generating stub code for the tests.

- Run the test cases and test suites

  You can run test cases and test suites various ways from within Flash Builder or outside the Flash Builder environment. The results of the tests are displayed in a test application. Flash Builder opens a Flex Unit Results View for analysis of the test run.

- Navigate to source code from the Flex Unit Results View

  In the Test Results panel, double-click a test to open the test implementation.

  The Test Failure Details panel lists the source and line number of the failure. If the listed source is in the current workspace, double-click the source to go directly to the failure.

Creating FlexUnit tests

You can create FlexUnit test case classes and test case suites for the following types of projects:

- Flex project
- ActionScript project
- Flex Library project
- AIR project

When creating a test case class, you can specify the following options:

- A `src` folder in a Flex project for the class
- A package for the class
- The classes to test
- Methods to test for each specified class

A FlexUnit test case suite is a series of tests based on previously created test case classes, specified methods in those classes, and other test case suites.

Creating a FlexUnit test case class

When you create a FlexUnit test case class, Flash Builder generates an ActionScript file for the test case class, which it places in a package for test cases.

The following procedure assumes that you have created a project in Flash Builder in which you want to create and run Flex Unit tests.

1. Select the Flex project and then from the context menu, select New > Test Case Class.

   If you select an ActionScript class file in project, then that class is automatically selected for the FlexUnit test case in the New Test Case Class wizard.

2. In the New Test Case Class wizard, specify whether to create a class in the FlexUnit 4 style or FlexUnit 1 style.
3. Specify a name for the test case class.
4. (Optional) Specify a source folder and package for the test case class, or accept the defaults.

   The default source folder is the `src` folder of the current project. The default package is `flexUnitTests`, which is at the top level of the default package structure for the project.

5. (Optional) Enable the Select Class to Test toggle, and browse to a specific class. Click Next.
6. (Optional) Select the methods in the selected class that you want to test.
7. Click Finish.

   Code the test case you created. Use the generated code stubs as a starting point.

Creating a FlexUnit test case suite

This procedure assumes that you have previously created test case classes.

1. Select the Flex project and then create a test case suite from the context menus by selecting New > Test Suite Class.
2. In the New Test Suite Class wizard, specify whether to create a class in the FlexUnit 4 style or FlexUnit 1 style.
3. Provide a name for the test suite.
4. Navigate in the test suites and test cases to select the classes and methods to include in the test suite. Click Finish.

Customizing default FlexUnit test case classes and test case suite classes

You can customize the default FlexUnit test case classes and test case suite classes that are created by Flash Builder. Flash Builder uses file templates to create the default versions of these files.

The file templates for FlexUnit are available from the Preferences window at Flash builder > File Templates > FlexUnit. There are separate templates for FlexUnit1 and FlexUnit4 test case classes and test suite classes.

See "Customizing File Templates" on page 120 for information on how to modify the default file templates.

Note: `FlexUnitCompilerApplication.mxml` *and* `FlexUnitApplication.mxml` *derive from the template for MXML Web Application or MXML Desktop Application. The template that is used depends on whether the Flex project is configured for a web application (runs in Adobe® Flash® Player) or a desktop application (runs in Adobe AIR®).*

More Help topics

Open source language reference for FlexUnit

Open source documentation for FlexUnit

Running FlexUnit tests

FlexUnit tests can be run from within Flash Builder or from outside Flash Builder using SWFs generated for the FlexUnit test. In either case, the results of the tests are displayed in the FlexUnit Results View.

You can also configure and save a FlexUnit test before running it.

By default, FlexUnit tests run in the Flash Debug perspective. You can launch tests from the Flash Development or Flash Profile perspectives, but Flash Builder switches to the Flash Debug perspective when running the test.

You can modify the default perspective for FlexUnit tests. Open the Preferences window, and navigate to Flash Builder > FlexUnit.

FlexUnit compiler application and FlexUnit application

When you create a FlexUnit test case, Flash Builder creates the following FlexUnit compiler application and a FlexUnit application:

- FlexUnitCompilerApplication.mxml
- FlexUnitApplication.mxml

Flash Builder uses these applications when compiling and running FlexUnit tests. Flash Builder places the applications, in the `src` directory of the project.

This application contains references to all FlexUnit test cases and test suites generated by Flash Builder. Flash Builder places all FlexUnit tests in the `<fx:Declarations>` tag of this application. You typically do not edit or modify this file directly.

Refresh the FlexUnit compiler application for the following circumstances:

- You manually add a test case.

  If you create a test case class without using the New Test Case wizard, refresh `FlexUnitCompilerApplication.mxml`. Place the new test case in the package with the other test cases.

- You rename a test case
- You delete a test case

Refresh `FlexUnitCompilerApplication.mxml`:

1. If the FlexUnit Results view is not open, select Windows > Other Views > FlexUnit Results. Click OK.
2. In the FlexUnit Results view, click the Refresh button.

Run a FlexUnit test within Flash Builder

You can run FlexUnit tests within Flash Builder on a project level or for individual test cases.

You typically run FlexUnit tests from the context menu for a project or from the context menus for an individual test case.

However, you can also run FlexUnit tests from the Flash Builder Run menu, the Flash Builder Run button, or from the Execute FlexUnit Test button in the FlexUnit Results view.

If you run tests from the Flash Builder Run menu, a Test Configuration dialog opens, allowing you to select which test classes and methods to run. Test cases for library projects cannot be run using the Flash Builder Run menu.

Flash Builder provides the following keyboard shortcuts to quickly launch FlexUnit tests:

- `Alt+Shift+A, F`

  Runs all FlexUnit tests in the project.

- `Alt+Shift+E, F`

  Runs the selected FlexUnit test.

Run FlexUnit tests from the current selection in the editor. (See "Configuring FlexUnit tests" on page 143.

1 Select a project and run the test:

  From the context menu for a project, select Execute FlexUnit Tests.

  From the Flash Builder Run menu or Run button, select Run > FlexUnit Tests.

2 (Flash Builder Run menu) In the Run FlexUnit Tests configuration dialog, select the test cases and methods to run in the test. Click OK to run the tests.

3 View the test results.

  Flash Builder generates a SWF file in the bin-debug folder of the project.

  An application opens displaying information about the test and indicates when the test is complete.

  The FlexUnit Results View panel opens displaying the results of the test. See "Viewing results of a FlexUnit test run" on page 144.

Run individual FlexUnit tests:

1 In the Project Explorer, navigate to the `flexUnitTest` package:

  From the context menu for a FlexUnit test file, select Execute FlexUnit Tests.

2 View the test results.

  Flash Builder generates a SWF file in the bin-debug folder of the project.

  An application opens displaying information about the test and indicates when the test is complete.

  The FlexUnit Results View panel opens displaying the results of the test. See "Viewing results of a FlexUnit test run" on page 144.

Run a FlexUnit test outside the Flash Builder environment

This procedure assumes that you have previously run a FlexUnit test within Flash Builder and that Flash Builder is running.

1 Copy the generated SWF file for a test from the bin-debug folder in your project to a folder outside your development environment.

  You can copy the automatically generated SWF file or a SWF file from a FlexUnit test that you have previously saved and configured.

2 Run the copy of the SWF file.

  Flash Player opens displaying information about the test and indicates when the test is complete.

  The FlexUnit Results View panel opens in Flash Builder displaying the results of the test.

Configuring FlexUnit tests

1 Open the Run FlexUnit Tests configuration dialog:

  You can open the Run FlexUnit Tests configuration dialog from the Run menu or from the FlexUnit Results view.

  - Select a project. From the Flash Builder menu, select Run > Run > Execute FlexUnit Test.
  - From the FlexUnit Results view, select the Execute FlexUnit Tests button.

    If the FlexUnit Results view is not open, select Window > Other Views > Flash Builder > FlexUnit Results.

2 In the Test Configuration dialog, select the test cases and methods to save as a test configuration.

Note: The Test Configuration dialog is not available when you run a test from the context menu in the Package Explorer.

3 (Optional) Select Load to import previous configurations saved as XML files.

4 Click Save.

Flash Builder writes an XML file and an MXML file in the `.FlexUnitSettings` folder of your project.

You can use the XML file in build scripts to execute the test.

You can generate a SWF file from the MXML file. This SWF file can be used for testing outside the Flash Builder environment. Typically, you copy the generated MXML file to the `src` folder in your project to generate the SWF file.

Viewing results of a FlexUnit test run

The FlexUnit Results view displays results of a FlexUnit test, detailing any failures. You can navigate through the results, filter the display, write the results to a file, and load the results from a file.

You can also rerun tests, cancel a running test, and clear the results from the view.

If the FlexUnit Results view is not open, select Window > Other Views > Flash Builder > FlexUnit Results.

Test Results Panel

This panel lists all tests within the test run, indicating whether the test passed or failed.

Double-click a test in the list to go to the test in the ActionScript editor.

Test Failure Details Panel

Select a test in the Test Results panel to display failure details.

Each failure detail lists the source file and method, including the line number of the failure.

If the source file is local to the working space, double-click the listing to go to the failure in the ActionScript editor.

FlexUnit Results view menu

From the FlexUnit Results view menu, you can do the following:

- Filter the results displayed

  Hide the Test Failure Details panel.

  Display only test runs that failed.

- Scroll through the test runs displayed in the Test Results panel.
- Cancel a running test.
- Save the results of a test run or the configuration of a test run.
- Load results previously saved to a file.
- Clear the results from the panel.
- Rerun the current test. You can choose from:
  - Run all tests.
  - Run failures only.
  - Run the selected test.

- Refresh the FlexUnit configuration.

  If you modify a test or add or remove tests, click refresh to load the new FlexUnit configuration.

- Configure and run FlexUnit tests.

  Use the Execute FlexUnit Tests button to configure and run FlexUnit tests.

Monitoring applications that access data services

The Network Monitor is a useful tool for monitoring and debugging applications that access data services. The Network Monitor allows you to examine the data that flows between an application and a data service. It also examines XML, AMF, and JSON data, which are sent using SOAP, AMF, HTTP, and HTTPS protocols.

The Network Monitor is active in the Flash Development and Flash Debug perspectives.

Enabling network monitoring

You enable the Network Monitor for individual Flex projects. The monitor state (enabled or disabled) applies to all applications within that project. You cannot enable or disable the Network Monitor on an individual application basis.

By default the Network Monitor is not enabled. You enable the Network Monitor by selecting the Enable Monitor icon in the Network Monitor toolbar.

Enabling the Network Monitor

This procedure assumes you are in the Flex Development or Flex Debug perspective.

1. If the Network Monitor view is not open, from the Flash Builder menu select Windows > Other Views > Flash Builder > Network Monitor.
2. If the Network Monitor is not enabled, in the Network Monitor toolbar click the Enable Network Monitor button.

   This button is a toggle for enabling or disabling the Network Monitor.

Monitoring remote services

To monitor your application, run either the development or debug version of the application with the Network Monitor enabled.

In general, the Network Monitor captures and stores all event data until you either quit the application or explicitly clear the data. The events are displayed in chronological order.

Starting a monitoring session

1. Run either a development or debug version of the application that accesses remote services.
2. For each access to a remote service, the Network Monitor lists the following:
   - Time of the request
   - Requesting service
   - The operation and URL if applicable
   - The time of the response
   - Elapsed time

3 Select a column header to sort the returned data according to the values in that column.

Click the column again to invert the order of the data.

4 Select the request and parameter tabs at the bottom of the monitor to view the details about the request operation.

The actual data sent in the request, as well as other information about the request, can be viewed from these tabs.

5 Select the response and result tabs at the bottom of the monitor to view the details about the response.

The actual data sent in the response, as well as other information about the response, can be viewed from these tabs.

6 Double-click an entry to go to the source code for that operation.

The Flash Builder source editor opens with the relevant line of source code highlighted.

***Note:** For most events, the Network Monitor can correlate an event with the Flex source code. For some events that are triggered outside the scope of the Network Monitor, the monitor cannot find the Flex source code.*

7 Click the Save button on the Network Monitor toolbar to write all captured information to an XML file.

***Note:** Use the generated XML file to study the data offline. You cannot import the data from this file back into the Network Monitor.*

8 Click the Clear icon in the Network Monitor toolbar to remove all captured information from the monitor.

Suspending a monitoring session

You can suspend and resume network monitoring. Suspending and resuming a session applies to all applications in the Flex project. For example, you cannot suspend one application in the project and continue monitoring another.

1 Click the Suspend button in the Network Monitor toolbar to suspend the monitoring of a session.

2 Click the Resume button in the toolbar to continue monitoring the session.

Stopping a monitoring session

To stop monitoring a session, you disable the Network Monitor.

1 (Optional) Close the Network Monitor.

***Note:** Simply closing the Network Monitor does not stop the monitoring session. Monitoring is still active, even if the Network Monitor is closed.*

2 Click the Enable Network Monitor button.

This button is a toggle for enabling or disabling the Network Monitor.

***Note:** Disabling the Network Monitor applies to all applications in the Flex project.*

Support for HTTPS protocol

The Network Monitor supports monitoring HTTPS calls to a server certified by a certificate authority (CA) or that has a self-signed certificate.

To monitor calls over the HTTPS protocol, modify the default preference for the Network Monitor to ignore SSL security checks. Open the Preferences dialog and navigate to Flash Builder > Network Monitor.

Viewing Network Monitor data

The leftmost panel of the Network Monitor provides information about the source of the data. It displays the following information:

- Source URL for the data service

- The type of service displayed

  For example RemoteService, HTTPService, or WebService.

- The request time, response time, and elapsed time for the data request
- The name of the operation called from the data service.

The Network Monitor has two tabs for viewing data, allowing you to view the request data and response data.

For each request and response, you can view the data in Tree View, Raw View, or Hex view. Select the corresponding icon for each view to change how the Network Monitor displays the data.

- Tree View

  Shows the XML, JSON, and AMF data in a tree structure format. This is the default view for data.

- Raw View

  Shows the actual data that is transferred.

- Hex View

  Shows the data in hexadecimal format. Hex view is useful when debugging binary data sent over a network.

By default, the Network Monitor clears all recorded data with every launch of an application. However, you can change the default behavior and retain all monitor data. Retained monitor data includes the data from all applications in all projects. Open the Preference dialog and navigate to Flash Builder > Network Monitor. Deselect Clear Entries on Start.

Saving Network Monitor data

You can save Network Monitor data to an XML. Click the Save button in the Network Monitor view to save Network Monitor data.

Monitoring multiple applications

You can monitor multiple applications simultaneously. There are two scenarios for monitoring multiple applications:

- Monitoring multiple applications in the same project

  You can only have one Network Monitor per Flex project. When monitoring multiple applications in the same project, events from all applications appear in the monitor according to the time the event occurred.

  You cannot filter events in the monitor according to specific applications.

- Monitoring multiple applications in different projects

  You can open a Network Monitor for each active Flex project. Each Network Monitor is independent of the other monitor, displaying only the events for its specific project.

  Suspending or disabling a Network Monitor in one project does not apply to monitors in other projects.

Limitations of the Network Monitor

Be aware of the following limitations when monitoring network data:

- The Network Monitor does not support applications that were created using pure ActionScript and Library projects.
- The Network Monitor does not support the Real Time Messaging Protocol (RTMP). For example, you cannot monitor streaming video.