

# Chapter 1: Using ASDoc

ASDoc is a command-line tool that you can use to create API language reference documentation as HTML pages from the ActionScript classes and MXML files in your Adobe® Flex™ application. The Adobe Flex team uses the ASDoc tool to generate the *Adobe Flex Language Reference*.

## About the ASDoc tool

The ASDoc tool parses one or more ActionScript class definitions and MXML files, and generates API language reference documentation for all public and protected methods and properties. The ASDoc tool also recognizes many types of metadata, such as the [Bindable], [Event], [Style], and [Effect] metadata tags.

You specify a single class, multiple classes, an entire namespace, or any combination as inputs to the ASDoc tool.

ASDoc generates its output as a directory structure of HTML files that matches the package structure of the input class files. Also, ASDoc generates an index of all public and protected methods and properties. To view the ASDoc output, open the index.html file in the top-level directory of the output.

## Using ASDoc

To use ASDoc, run the `asdoc` command from the bin directory of your Flex installation. For example, from the bin directory, enter the following command to create output for the Flex Button class:

```
asdoc -source-path C:\flex\frameworks\projects\framework\src
      -doc-classes mx.controls.Button
      -main-title "Flex API Documentation"
      -window-title "Flex API Documentation"
      -output framework-asdoc
```

In this example, the source code for the Button class is in the directory `C:\flex\frameworks\projects\framework\src\mx\controls`. Use the `source-path` option to specify where the ASDoc tool looks for the source code, and the `doc-classes` option to specify the name of the class to process.

The ASDoc tool writes the output to `C:\flex\bin\framework-asdoc` directory, as defined by the `output` option.

The ASDoc tool supports many options for specifying the files to process. For example, instead of explicitly specifying the list of files to process, you can use the `doc-sources` option to specify a directory name. The following example runs the ASDoc tool on all files in the `C:\flex\frameworks\projects\framework\src\mx\controls` directory:

```
asdoc
      -doc-sources C:\a\flex\flex\sdk\frameworks\projects\framework\src\mx\controls
      -main-title "Flex API Documentation"
      -window-title "Flex API Documentation"
      -output framework-asdoc
```

To view the output, open the file `C:\flex\bin\framework-asdoc\index.html`. For more information on running the `asdoc` command, see “Using the ASDoc tool” on page 21.

## Creating ASDoc comments in ActionScript

A standard programming practice is to include comments in source code. The ASDoc tool recognizes a specific type of comment in your source code and copies that comment to the generated output.

### Writing an ASDoc comment

An ASDoc comment consists of the text between the characters `/**` that mark the beginning of the ASDoc comment, and the characters `*/` that mark the end of it. The text in a comment can continue onto multiple lines.

Use the following format for an ASDoc comment:

```
/**
 * Main comment text.
 *
 * @tag Tag text.
 */
```

As a best practice, prefix each line of an ASDoc comment with an asterisk (\*) character, followed by a single white space to make the comment more readable, and to ensure correct parsing of comments. When the ASDoc tool parses a comment, the leading asterisk and white-space characters on each line are discarded; blanks and tabs preceding the initial asterisk are also discarded.

The ASDoc comment in the previous example creates a single-paragraph description in the output. To add additional comment paragraphs, enclose each subsequent paragraph in HTML paragraph tags, `<p></p>`. You must close the `<p>` tag, in accordance with XHTML standards, as the following example shows:

```
/**
 * First paragraph of a multiparagraph description.
 *
 * <p>Second paragraph of the description.</p>
 */
```

All the classes that ship with Flex contain the ASDoc comments that appear in the *Adobe Flex Language Reference*. For example, view the `mx.controls.Button` class for examples of ASDoc comments.

### Placing ASDoc comments

Place an ASDoc comment immediately before the declaration for a class, interface, constructor, method, property, or metadata tag that you want to document, as the following example shows for the `myMethod()` method:

```
/**
 * This is the typical format of a simple
 * multiline (single paragraph) main description
 * for the myMethod() method, which is declared in
 * the ActionScript code below.
 * Notice the leading asterisks and single white space
 * following each asterisk.
 */
public function myMethod(param1:String, param2:Number):Boolean {}
```

The ASDoc tool ignores comments placed in the body of a method and recognizes only one comment per ActionScript statement.

A common mistake is to put an `import` statement, or other code line or metadata tag, between the ASDoc comment for a class and the `class` declaration. Because an ASDoc comment is associated with the next ActionScript statement in the file after the comment, this example associates the comment with the `import` statement, not the `class` declaration:

```
/**
 * This is the class comment for the class MyClass.
 */
import flash.display.*; // MISTAKE - Do not to put import statement here.
class MyClass {
}
```

## Formatting ASDoc comments

The main body of an ASDoc comment begins immediately after the starting characters, `/**`, and continues until the tag section, as the following example shows:

```
/**
 * Main comment text continues until the first tag.
 *
 * @tag Tag text.
 */
```

The first sentence of the main description of the ASDoc comment should contain a concise but complete description of the declared entity. The first sentence ends at the first period followed by a space, tab, or line terminator.

ASDoc uses the first sentence to populate the summary table at the top of the HTML page for the class. Each type of class element (method, property, event, effect, and style) has a separate summary table in the ASDoc output.

The tag section begins with the first ASDoc tag in the comment, defined by the first `@` character that begins a line, ignoring leading asterisks, white space, and the leading separator characters, `/**`. The main description cannot continue after the tag section begins.

The text following an ASDoc tag can span multiple lines. You can have any number of tags, where some tags can be repeated, such as the `@param` and `@see` tags, while others cannot.

The following example shows an ASDoc comment that includes a main description and a tag section. Notice the use of white space and leading asterisks to make the comment more readable:

```
/**
 * Typical format of a simple multiline comment.
 * This text describes the myMethod() method, which is declared below.
 *
 * @param param1 Describe param1 here.
 * @param param2 Describe param2 here.
 *
 * @return Describe return value here.
 *
 * @see someOtherMethod
 */
public function myMethod(param1:String, param2:Number):Boolean {}
```

For a complete list of the ASDoc tags, see “ASDoc tags” on page 15.

## Using the @private tag

By default, the ASDoc tool generates output for all public and protected elements in an ActionScript class, even if you omit the ASDoc comment for the element. The ASDoc tool ignores all elements defined as private.

To make ASDoc ignore a public or protected element, insert an ASDoc comment that contains the @private tag anywhere in the comment. The ASDoc comment can contain additional text along with the @private tag, which is also excluded from the output.

ASDoc generates output for all public classes in the list of input classes. You can specify to ignore an entire class by inserting an ASDoc comment that contains the @private tag before the class definition. The ASDoc comment can contain additional text along with the @private tag, which is also excluded from the output, as the following example shows:

```
/**
 * This class is omitted from the output.
 *
 * @private
 */
class MyClass {
}
```

## Excluding an inherited element

By default, the ASDoc tool copies information and a link for all elements inherited by a subclass from a superclass. In some cases, a subclass might not support an inherited element. You can use the [Exclude] metadata tag to cause ASDoc to omit the inherited element from the list of inherited elements.

The [Exclude] metadata tag has the following syntax:

```
[Exclude (name="elementName", kind="property|method|event|style|effect")]
```

For example, to exclude documentation on the click event in the MyButton subclass of the Button class, insert the following [Exclude] metadata tag in the MyButton.as file:

```
[Exclude (name="click", kind="event")]
```

## Using HTML tags

You can use selected HTML entities and HTML tags to define paragraphs, format text, create lists, and add anchors. For a list of the supported HTML tags, see “Summary of commonly used HTML elements” on page 19.

Write the text of an ASDoc comment in XHTML-compliant HTML. That means your HTML syntax has to conform to XML syntax rules. For example, close all HTML tags, such as <p> and <code> tags, by inserting the closing </p> or </code> tag.

The following example comment contains HTML tags to format the output:

```

/**
 * This is the typical format of a simple multiline comment
 * for the myMethod() method.
 *
 * <p>This is the second paragraph of the main description
 * of the <code>myMethod</code> method.
 * Notice that you do not use the paragraph tag in the
 * first paragraph of the description.</p>
 *
 * @param param1 Describe param1 here.
 * @param param2 Describe param2 here.
 *
 * @return A value of <code>>true</code> means this;
 * <code>>false</code> means that.
 *
 * @see someOtherMethod
 */
public function myMethod(param1:String, param2:Number):Boolean {}

```

## Using special characters

The ASDoc tool can fail if your source files contain non-UTF-8 characters such as curly quotes. If it does fail, the error messages it displays refers to a line number in the class. That message helps you track down the location of the special character.

ASDoc passes all HTML tags and tag entities in a comment to the output. Therefore, if you want to use special characters in a comment, enter them using HTML code equivalents. For example, to use a less-than (<) or greater-than (>) symbols in a comment, use `&lt;`; and `&gt;`; . To use the at-sign (@) in a comment, use `&#64;`; . Otherwise, these characters are interpreted as literal HTML characters in the output.

For a list of common HTML tags and their entity equivalents, see “Summary of commonly used HTML elements” on page 19.

Because asterisks (\*) are used to delimit comments, ASDoc does not support asterisks within a comment. To use an asterisk in an ASDoc comment, use the double tilde (~~).

## Hiding text in ASDoc comments

The ASDoc style sheet contains a class called `hide`, which you use to hide text in an ASDoc comment by setting the class attribute to `hide`. Hidden text does not appear in the ASDoc HTML output, but does appear in the generated HTML file. Therefore, do not use it for confidential information. The following example uses the `hide` class:

```

/**
 * Dispatched when the user presses the Button control.
 * If the <code>autoRepeat</code> property is <code>true</code>,
 * this event is dispatched repeatedly as long as the button stays down.
 *
 * <span class="hide">This text is hidden.</span>
 * @eventType mx.events.FlexEvent.BUTTON_DOWN
 */

```

## Rules for parsing ASDoc comments

The following rules summarize how ASDoc processes an ActionScript file:

- If an ASDoc comment precedes an ActionScript element, ASDoc copies the comment and code element to the output file.
- If an ActionScript element is not preceded by an ASDoc comment, ASDoc copies the code element to the output file with an empty description.
- If an ASDoc comment contains the `@private` ASDoc tag, the associated ActionScript element and the ASDoc comment are ignored.
- The comment text must precede any `@` tags, otherwise the comment text is interpreted as an argument to an `@` tag. The only exception is the `@private` tag, which can appear anywhere in an ASDoc comment.
- HTML tags, such as `<p></p>`, and `<ul></ul>`, in ASDoc comments are passed through to the output.
- HTML tags must use XML style conventions, which means there must be a beginning and ending tag. For example, close an `<li>` tag with a `</li>` tag.

## Documenting ActionScript elements

You can add ASDoc comments to class, property, method, and metadata elements to document ActionScript classes. For more information on documenting MXML files, see “Documenting MXML files” on page 11.

### Documenting classes

The ASDoc tool automatically includes all public classes in its output. Place the ASDoc comment for a class just before the `class` declaration, as the following example shows:

```
/**
 * The MyButton control is a commonly used rectangular button.
 * MyButton controls look like they can be pressed.
 * They can have a text label, an icon, or both on their face.
 */
public class MyButton extends UIComponent {
}
```

This comment appears at the top of the HTML page for the associated class.

To configure ASDoc to omit the class from the output, insert an `@private` tag anywhere in the ASDoc comment, as the following example shows:

```
/**
 * @private
 * The MyHiddenButton control is for internal use only.
 */
public class MyHiddenButton extends UIComponent {
}
```

### Documenting properties

The ASDoc tool automatically includes all public and protected properties in its output. You can document properties that are defined as variables or defined as setter and getter methods.

### Documenting properties defined as variables

Place the ASDoc comment for a public or protected property that is defined as a variable just before the `var` declaration, as the following example shows:

```
/**
 * The default label for MyButton.
 *
 * @default null
 */
public var myButtonLabel:String;
```

A best practice for a property is to include the `@default` tag to specify the default value of the property. The `@default` tag has the following format:

```
@default value
```

This tag generates the following text in the output for the property:

```
The default value is value.
```

For properties that have a calculated default value, or a complex description, omit the `@default` tag and describe the default value in text.

ActionScript lets you declare multiple properties in a single statement. However, this does not allow for unique documentation for each property. Such a statement can have only one ASDoc comment, which is copied for all properties in the statement. For example, the following documentation comment does not make sense when written as a single declaration and would be better handled as two declarations:

```
/**
 * The horizontal and vertical distances of point (x,y)
 */
public var x, y;// Avoid this
```

ASDoc generates the following documentation from the preceding code:

```
public var x
    The horizontal and vertical distances of point (x,y)

public var y
    The horizontal and vertical distances of point (x,y)
```

### Documenting properties defined by setter and getter methods

Properties defined by setter and getter methods are handled in a special way by the ASDoc tool because these elements are used as if they were properties rather than methods. Therefore, ASDoc creates a property definition for an item that is defined by a setter or a getter method.

If you define a setter method and a getter method, insert a single ASDoc comment before the getter, and mark the setter as `@private`. Adobe recommends this practice because usually the getter comes first in the ActionScript file, as the following example shows:

```
/**
 * Indicates whether or not the text field is enabled.
 */
public function get html():Boolean {};

/**
 * @private
 */
public function set html(value:Boolean):void {};
```

The following rules define how ASDoc handles properties defined by setter and getter methods:

- If you precede a setter or getter method with an ASDoc comment, the comment is included in the output.
- If you define both a setter and a getter method, only a single ASDoc comment is needed – either before the setter or before the getter.
- If you define a setter method and a getter method, insert a single ASDoc comment before the getter, and mark the setter as `@private`.
- You do not have to define the setter method and getter method in any particular order, and they do not have to be consecutive in the source-code file.
- If you define just a getter method, the property is marked as read-only.
- If you define just a setter method, the property is marked as write only.
- If you define both a public setter and public getter method in a class, and you want to hide them by using the `@private` tag, they both must be marked `@private`.
- If you have only one public setter or getter method in a class, and it is marked `@private`, ASDoc applies normal `@private` rules and omits it from the output.
- A subclass always inherits its visible superclass setter and getter method definitions.

## Documenting methods

The ASDoc tool automatically includes all public and protected methods in its output. Place the ASDoc comment for a public or protected method just before the `function` declaration, as the following example shows:

```
/**
 * This is the typical format of a simple multiline documentation comment
 * for the myMethod() method.
 *
 * <p>This is the second paragraph of the main description
 * of the <code>myMethod</code> method.
 * Notice that you do not use the paragraph tag in the
 * first paragraph of the description.</p>
 *
 * @param param1 Describe param1 here.
 * @param param2 Describe param2 here.
 *
 * @return A value of <code>true</code> means this;
 * <code>false</code> means that.
 *
 * @see someOtherMethod
 */
public function myMethod(param1:String, param2:Number):Boolean {}
```

If the method takes an argument, include an `@param` tag for each argument to describe the argument. The order of the `@param` tags in the ASDoc comment must match the order of the arguments to the method. The `@param` tag has the following syntax:

```
@param paramName description
```

Where *paramName* is the name of the argument and *description* is a description of the argument.

If the method returns a value, use the `@return` tag to describe the return value. The `@return` tag has the following syntax:

```
@return description
```

Where *description* describes the return value.

## Documenting metadata

Flex uses metadata tags to define certain characteristics of a class. ASDoc recognizes some of these metadata tags to generate output. The metadata tags recognized by ASDoc include the following:

- [Bindable]
- [DefaultProperty]
- [Deprecated]
- [Effect]
- [Event]
- [Exclude]
- [Style]

Some metadata tags take an ASDoc comment, such as [Effect] and [Event]. Other metadata tags, such as [Bindable] and [DefaultProperty] do not. If you put an ASDoc comment before a metadata tag that does not accept a comment, the comment is passed through to the next element in the class. If the next element in the class already has an ASDoc comment, the comment on the metadata tag is ignored.

For more information on the [Exclude] tag, see “Excluding an inherited element” on page 4.

For more information on using these metadata tags in an application, see Metadata Tags in Custom Components.

### Documenting bindable properties

A bindable property is any property that can be used as the source of a data binding expression. To mark a property as bindable, you insert the [Bindable] metadata tag before the property definition, or before the class definition to make all properties defined within the class bindable. The [Bindable] metadata tag does not take an ASDoc comment.

When a property is defined as bindable, ASDoc automatically adds the following line to the output for the property:

```
This property can be used as the source for data binding.
```

For more information on the [Bindable] metadata tag, see Metadata Tags in Custom Components.

### Documenting default properties

The [DefaultProperty] metadata tag defines the name of the default property of the component when you use the component in an MXML file. The [DefaultProperty] metadata tag does not take an ASDoc comment.

When ASDoc encounters the [DefaultProperty] metadata tag, it automatically adds a line to the class description that specifies the default property. For example, see the List control in *Adobe Flex Language Reference*.

For more information on the [DefaultProperty] metadata tag, see Metadata Tags in Custom Components.

### Documenting deprecated properties

A class or class elements marked as deprecated is one which is considered obsolete, and whose use is discouraged. While the class or class element still works, its use can generate compiler warnings.

Insert the [Deprecated] metadata tag before a property, method, or class definition to mark that element as deprecated. The [Deprecated] metadata tag does not take an ASDoc comment.

The following example uses the [Deprecated] metadata tag to mark the dataProvider property as obsolete:

```
[Deprecated(replacement="MenuBarItem.data")]
public function set dataProvider(value:Object):void
{}
```

When ASDoc encounters the `[Deprecated]` metadata tag, it adds a line to the output marking the item as deprecated, and inserts a link to the replacement item.

The mxmcl command-line compiler supports the `show-deprecation-warnings` compiler option, which, when `true`, configures the compiler to issue deprecation warnings when your application uses deprecated elements. The default value is `true`.

For more information on the `[Deprecated]` metadata tag, see [Deprecated metadata tag](#).

## Documenting skin states and skin parts

### Documenting effects, events, and styles

You use metadata tags to add information about effects, events, and styles in a class definition. The `[Effect]`, `[Event]`, and `[Style]` metadata tags typically appear at the top of the class definition file. To document the metadata tags, insert an ASDoc comment before the metadata tag, as the following example shows:

```
/**
 * Defines the name style.
 */
[Style "name"]
```

For events and effects, the metadata tag includes the name of the event class associated with the event or effect. The following example shows an event definition from the `Flex mx.controls.Button` class:

```
/**
 * Dispatched when the user presses the Button control.
 * If the <code>autoRepeat</code> property is <code>>true</code>,
 * this event is dispatched repeatedly as long as the button stays down.
 *
 * @eventType mx.events.FlexEvent.BUTTON_DOWN
 */
[Event(name="buttonDown", type="mx.events.FlexEvent")]
```

In the ASDoc comment for the `mx.events.FlexEvent.BUTTON_DOWN` constant, you insert a table that defines the values of the `bubbles`, `cancelable`, `target`, and `currentTarget` properties of the `Event` class, and any additional properties added by a subclass of `Event`. At the end of the ASDoc comment, you insert the `@eventType` tag so that ASDoc can find the comment, as the following example shows:

```
/**
 * The FlexEvent.BUTTON_DOWN constant defines the value of the
 * <code>type</code> property of the event object
 * for a <code>buttonDown</code> event.
 *
 * <p>The properties of the event object have the following values:</p>
 * <table class=innertable>
 * <tr><th>Property</th><th>Value</th></tr>
 * ...
 * </table>
 *
 * @eventType buttonDown
 */
public static const BUTTON_DOWN:String = "buttonDown"
```

The ASDoc tool does several things for this event:

- In the output for the `mx.controls.Button` class, ASDoc creates a link to the event class specified by the `type` argument of the `[Event]` metadata tag.
- ASDoc copies the description of the `mx.events.FlexEvent.BUTTON_DOWN` constant to the description of the `buttonDown` event in the `Button` class.

For a complete example, see the `mx.controls.Button` and `mx.events.FlexEvent` classes.

For more information on the `[Effect]`, `[Event]`, and `[Style]` metadata tags, see [Metadata Tags in Custom Components](#).

## Documenting MXML files

An MXML file contains several types of elements, including MXML code, ActionScript code in `<Script>` blocks, and metadata tags. The ASDoc tool supports all these element types so that you can generate ASDoc content for MXML files just as you can for ActionScript classes.

MXML files correspond to ActionScript classes where the superclass corresponds to the first tag in the MXML file. For an application file, that tag is the `<Application>` tag and therefore an MXML application file appears in the ASDoc output as a subclass of the `Application` class.

## Documenting MXML elements

Use the following syntax to specify an ASDoc comment in an MXML file for an element defined in MXML:

```
<!-- asdoc comment -->
```

The comment must contain three dashes following the opening `<!` characters, and end with two dashes before the closing `>` character, as the following example shows:

```
<?xml version="1.0"?>
<!-- asdoc\MyVBox.mxml -->
<!--
    The class level comment for the component.
    This tag supports all ASDoc tags,
    and does not require a CDATA block.
-->
<VBox xmlns="http://ns.adobe.com/mxml/2009">

    <!--
        Comment for button
    -->
    <Button id="myButton" label="This button has a comment"/>
</VBox>
```

In this example, the first comment is a standard XML comment ignored by ASDoc. The second comment precedes the root tag of the component and uses the three dashes to identify it as an ASDoc comment. An ASDoc comment on the root tag is equivalent to the ASDoc comment before an ActionScript class definition. Therefore, the comment appears at the top of the output ASDoc HTML file.

A leading dash at the beginning of each comment line, and any whitespace characters before the dash, are ignored, as the following example shows:

```
<!---
  - Comment for my class
  - which is implemented as mxml
-->
```

If you copy a comment from an ActionScript file that uses the `/**`, `*`, and `*/` characters, those characters are also ignored, as the following example shows:

```
<!---
  /**
   * Comment for my class
   * which is implemented as mxml
  */
-->
<!---
  * Comment for my class
  * which is implemented as mxml
-->
```

All MXML elements in the file correspond to public properties of the component. The comment before the `Button` control defines the ASDoc comment for the public property named `myButton` of type `mx.controls.Button`.

You can use any ASDoc tags in these comments, including the `@see`, `@copy`, `@param`, `@return`, and other ASDoc comments.

The ASDoc command-line tool only processes elements of an MXML file that contain an `id` attribute. If the MXML element has an `id` attribute but no comment, the element appears in the ASDoc output with a blank comment. An MXML element with no `id` attribute is ignored, even if it is preceded by an ASDoc comment, as the following example shows:

```
<?xml version="1.0"?>
<!-- asdoc\MyVBoxID.mxml -->
<!---
  The class level comment for the component.
  This tag supports all ASDoc tags,
  and does not require a CDATA block.

  @see mx.container.VBox
-->
<VBox xmlns="http://ns.adobe.com/mxml/2009">

  <!---
    Comment for first button appears in the output.
  -->
  <Button id="myButton" label="This button has a comment"/>

  <Button id="myButton2"
    label="Has id but no comment so appears in output"/>

  <!---
    Comment for button with no id is ignored by ASDoc.
  -->
  <Button label="This button has no id"/>
</VBox>
```

Comments before `Definition`, `Library`, and `Private` tags are ignored. Also comments inside a private block are ignored.

## Documenting ActionScript in <Script> blocks

Insert ASDoc comments for ActionScript code in the <Script> block by using the same syntax as you use in an ActionScript file. The only requirement is that the ASDoc comments must be within a CDATA block, as the following example shows:

```
<?xml version="1.0"?>
<!-- asdoc\MyVBoxComplex.mxml -->
<!--
    The class level comment for the component.
    This tag supports all ASDoc tags,
    and does not require a CDATA block.
-->
<VBox xmlns="http://ns.adobe.com/mxml/2009">
    <!--
        Comment for language element - this comment will be ignored.
    -->
    <Script>
        <![CDATA[
            import flash.events.MouseEvent;

            /**
             * For a method in an <Script> block,
             * same rules as in an AS file.
             *
             * @param eventObj The event object.
             */
            public function handleClickEvent(eventObj:MouseEvent):void {
                dispatchEvent(eventObj);
            }

            /**
             * For a property in an <Script> block,
             * same rules as in an AS file.
             */
            public var myString:String = new String();

        ]]>
    </Script>

    <!--
        Comment for first button appears in the output.
    -->
    <Button id="myButton" label="This button has a comment"
        click="handleClickEvent(event);"/>

    <Button id="myButton2"
        label="Has id but no comment so appears in output"/>

    <!--
        Comment for button with no id is ignored by ASDoc.
    -->
    <Button label="This button has no id"/>
</VBox>
```

## Documenting metadata tags

You can insert ASDoc comments for metadata tags in `<Metadata>` blocks in an MXML file. For metadata tags, the ASDoc comments use the same syntax as you use in an ActionScript file. The only requirement is that the ASDoc comments must be within a CDATA block, as the following example shows:

```
<?xml version="1.0"?>
<!-- asdoc\MyVBoxMetaData.mxml -->
<!--
    The class level comment for the component.
    This tag supports all ASDoc tags,
    and does not require a CDATA block.
-->
<VBox xmlns="http://ns.adobe.com/mxml/2009">
    <!--
        Comment for language element - this comment will be ignored.
    -->
    <Script>
        <![CDATA[
            import flash.events.MouseEvent;

            /**
             * For a method in an <Script> block,
             * same rules as in an AS file.
             *
             * @param eventObj The event object.
             */
            public function handleClickEvent(eventObj:MouseEvent):void {
                dispatchEvent(eventObj);
            }

            /**
             * For a property in an <Script> block,
             * same rules as in an AS file.
             */
            public var myString:String = new String();

        ]]>
    </Script>

    <Metadata>
        <![CDATA[
            /**
             * Defines the default style of selected text.
             */
            [Style(name="textSelectedColor", type="Number", format="Color", inherit="yes")]

            /**
```

```

    * The component dispatches the darken event
    * when the darken property changes.
    *
    * @eventType flash.events.Event
    */
    [Event(name="darken", type="flash.events.Event")]

    /**
     * Played when the component darkens.
     */
    [Effect(name="darkenEffect", event="darken")]
    ]]>
</Metadata>

<!--
    Comment for first button appears in the output.
-->
<Button id="myButton" label="This button has a comment"
    click="handleClickEvent(event);"/>
</VBox>

```

## ASDoc tags

The following table lists the ASDoc tags:

ASDoc tag	Description	Example
@copy reference	<p>Copies an ASDoc comment from the referenced location. The main description, @param, and @return content is copied; other tags are not copied.</p> <p>You typically use the @copy tag to copy information from a source class or interface not in the inheritance list of the destination class. If the source class or interface is in the inheritance list, use the @inheritDoc tag instead.</p> <p>You can add content to the ASDoc comment before the @copy tag.</p> <p>Specify the location by using the same syntax as you do for the @see tag. For more information, see "Using the @see tag" on page 18.</p>	<pre>@copy #stop @copy MovieClip#stop</pre>
@default value	<p>Specifies the default value for a property, style, or effect. The ASDoc tool automatically creates a sentence in the following form when it encounters an @default tag:</p> <p>The default value is value.</p>	@default 0xCCCCCC
@eventType package.class.CONSTANT@eventType String	<p>Use the first form in a comment for an [Event] metadata tag. It specifies the constant that defines the value of the Event.type property of the event object associated with the event. The ASDoc tool copies the description of the event constant to the referencing class.</p> <p>Use the second form in the comment for the constant definition. It specifies the name of the event associated with the constant. If the tag is omitted, ASDoc cannot copy the constant's comment to a referencing class.</p>	See
@example exampleText	<p>Defines a code example in an ASDoc comment. By preceding the code example with this tag, ASDoc applies style properties, generates a heading, and puts the code example in the correct location.</p> <p>Enclose the code in &lt;listing version="3.0"&gt;&lt;/listing&gt; tags.</p> <p>Whitespace formatting is preserved and the code is displayed in a gray, horizontally scrolling box.</p> <p>If the code inside the &lt;listing&gt; tags uses literal "&lt;", "&gt;", or "&amp;" characters, convert them to the HTML character-code equivalent.</p>	<pre>@example The following code sets the volume level for your sound:  &lt;listing version="3.0"&gt;  var mySound:Sound = new Sound(); mySound.setVolume(VOL_HIGH);  &lt;/listing&gt;</pre>
@exampleText string	<p>Use this tag in an ASDoc comment in an external example file that is referenced by the @includeExample tag. The ASDoc comment must precede the first line of the example, or follow the last line of the example.</p> <p>External example files support one comment before and one comment after example code.</p>	<pre>/**  * This text does not appear  * in the output.  * @exampleText But this does.  */</pre>

ASDoc tag	Description	Example
<p><code>@includeExample</code> <i>textFile</i></p>	<p>Imports an example text file into the ASDoc output. ASDoc searches for the example file based on the package name of the class and the directory specified by the <code>-examples-path</code> option to the ASDoc tool.</p> <p>For example, you use the <code>examples-path</code> option to set the directory to <code>c:\examples</code>. To add an example for the <code>mx.controls.Button</code> class, place it in the <code>mx\controls\directory</code> under <code>c:\examples</code>, meaning the <code>c:\examples\mx\controls</code> directory.</p> <p>You can further qualify the location of the file by specifying a path to the <code>@includeExample</code> tag. For example, you specify the <code>@includeExample</code> as shown below:</p> <pre>@includeExample buttonExample/ButtonExample.xml</pre> <p>ASDoc looks for an example in the directory <code>c:\examples\mx\controls\buttonExample</code>.</p> <p>If you insert this tag in the comment for a class, the example appears at the end of the output HTML file. If you insert it in the ASDoc comment for a class element, the example appears in the detailed description of the element.</p>	<pre>@includeExample ButtonExample.xml</pre>
<p><code>@inheritDoc</code></p>	<p>Copies the comment from the superclass into the subclass, or from an interface implemented by the subclass. Use this tag in the comment of an overridden method or property. You cannot use this tag with comments on metadata tags.</p> <p>The main ASDoc comment, <code>@param</code>, and <code>@return</code> content are copied; other tags are not. You can add content to the comment before the <code>@inheritDoc</code> tag.</p> <p>When you include this tag, ASdoc uses the following search order:</p> <ol style="list-style-type: none"> <li>1. Interfaces implemented by the current class, in alphabetical order of the package and class name, and all their base-interfaces.</li> <li>2. Immediate superclass of current class.</li> <li>3. Interfaces of immediate superclass and all their base-interfaces.</li> <li>4. Repeat steps 2 and 3 until the Object class is reached.</li> </ol> <p>You can also use the <code>@copy</code> tag, but the <code>@copy</code> tag is for copying information from a source class or interface that is not in the inheritance chain of the subclass.</p>	<pre>@inheritDoc</pre>
<p><code>@internal</code> <i>text</i></p>	<p>Hides the text attached to the tag in the generated output. The hidden text can be used for internal comments.</p>	<pre>@internal Please do not publicize the undocumented use of the third parameter in this method.</pre>

ASDoc tag	Description	Example
<code>@param paramName description</code>	Adds a descriptive comment to a method parameter. The <code>paramName</code> argument must match a parameter definition in the method signature.	<code>@param fileName</code> The name of the file to load.
<code>@private</code>	Exclude the element from the generated output.  To omit an entire class, put the <code>@private</code> tag in the ASDoc comment for the class; to omit a single class element, put the <code>@private</code> tag in the ASDoc comment for the element.	<code>@private</code>
<code>@return description</code>	Adds a Returns section to a method description with the specified text. ASDoc automatically determines the data type of the return value.	<code>@return</code> The translated message.
<code>@see reference [displayText]</code>	Adds a See Also heading with a link to a class element. For more information, see “Using the <code>@see</code> tag” on page 18.  Do not include HTML formatting characters in the arguments to the <code>@see</code> tag.	<code>@see flash.display.MovieClip</code>
<code>@since text</code>	Adds a Since section to a class or element.	<code>@since</code> January 12, 2009
<code>@throws package.class.className description</code>	Documents an error that a method can throw.	<code>@throws SecurityError</code> Local untrusted SWFs may not communicate with the Internet.

## Using the @see tag

The `@see` tag lets you create cross-references to elements within a class; to elements in other classes in the same package; and to other packages. You can also cross-reference URLs outside ASDoc. The `@see` tag has the following syntax:

```
@see reference [displayText]
```

where *reference* specifies the destination of the link, and *displayText* optionally specifies the link text. The location of the destination of the `@see` tag is determined by the prefix to the *reference* attribute:

- `#` ASDoc looks in the same class for the link destination.
- `ClassName` ASDoc looks in a class in the same package for the link destination.
- `PackageName` ASDoc looks in a different package for the link destination.
- `global` ASDoc looks in the Top Level package for the link destination.
- `effect` ASDoc looks for an effect property for the link destination.
- `event` ASDoc looks for an event property for the link destination.
- `style` ASDoc looks for a style property for the link destination.

**Note:** You cannot insert HTML tags in *reference*. However, you can add an HTML link without using the `@see` tag by inserting the HTML code in the ASDoc comment.

The following table shows several examples of the `@see` tag:

Example	Result
@see "Just a label"	Text string
@see http://www.cnn.com	External website
@see package-detail.html	Local HTML file
@see Array	Top-level class
@see AccessibilityProperties	Class in same package
@see flash.display.TextField	Class in different package
@see Array#length	Property in top level class
@see flash.ui.ContextMenu#customItems	Property in class in different package
@see mx.containers.DividedBox#style:dividerAffordance	Style property in class in different package
@see #updateProperties()	Method in same class as @see tag
@see Array#pop()	Method in top-level class
@see flash.ui.ContextMenu#clone()	Method in class in different package
@see global#Boolean()	Package method in Top Level (global)
@see flash.util.#clearInterval()	Package method in flash.util
@see mx.controls.Button#style:horizontalGap	Style property in Button class.
@see mx.containers.Accordion#event:change	Event in Accordion class.
@see mx.core.UIComponent#effect:creationCompleteEffect	Effect property in UIComponent class.

## Summary of commonly used HTML elements

Write the text of an ASDoc comment in XHTML-compliant HTML. That means your HTML syntax has to conform to XML syntax rules. For example, close all HTML tags, such as `<p>` and `<code>` tags, by inserting the closing `</p>` or `</code>` tag.

The following table lists commonly used HTML tags and character codes within ASDoc comments:

Tag or Code	Description
<p>	<p>Starts a new paragraph. You must close &lt;p&gt; tags.</p> <p>Do not use &lt;p&gt; for the first paragraph of a doc comment (the paragraph after the opening /**) or the first paragraph associated with a tag. Use the &lt;p&gt; tag for subsequent; for example:</p> <pre>/**  * The first sentence of a main description.  *  * &lt;p&gt;This line starts a new paragraph.&lt;/p&gt;  *  * &lt;p&gt;This line starts a third paragraph.&lt;/p&gt;  */</pre> <p>ASDoc ignores white space in comments. To add white space for readability in the AS file, do not use the &lt;p&gt; tag but add blank lines.</p>
class="hide"	Hides text. Use this tag if you want to add documentation to the source file but do not want it to appear in the output.
<listing>	<p>Indicates a program listing (sample code).</p> <p>Use this tag to enclose code snippets that you format as separate paragraphs, in monospace font, and in a gray background to distinguish the code from surrounding text. Close all &lt;listing&gt; tags.</p>
<pre>	<p>Formats text in monospace font, such as a description of an algorithm or a formula. Do not use &lt;br/&gt; tags at end of line.</p> <p>Use &lt;listing&gt; tag for code snippets.</p>
 	<p>Adds a line break. You must close this tag.</p> <p>Comments for most tags are automatically formatted; you do not generally have to add line breaks. To create additional white space, add a new paragraph instead.</p> <p>This tag might not be supported in the future, so use it only if necessary.</p>
<ul>, <li>	Creates a list. You must close these tags.
<table><th><tr><td>	<p>Creates a table. For basic tables that conform to ASDoc style, set the class attribute to <code>innertable</code>. Avoid setting any special attributes. Avoid nesting structural items, such as lists, within tables.</p> <p>ASDoc uses a standard CSS stylesheet that has definitions for the &lt;table&gt;, &lt;th&gt;, &lt;tr&gt; and &lt;td&gt; tags. You must close these tags.</p> <p>Use &lt;th&gt; for header cells instead of &lt;td&gt;, so the headers get formatted correctly.</p>
<img>	<p>Inserts an image. To create the correct amount of space around an image, enclose the image reference in &lt;p&gt;&lt;/p&gt; tags. Captions are optional; if you use a caption, make it boldface. You must close the &lt;img&gt; tag by ending it with /&gt;, as the following example shows:</p> <pre>&lt;img src = "../..images/matrix.jpg" /&gt;</pre>
<code>	Applies monospace formatting. You must close this tag.
<strong>	Applies bold text formatting. You must close this tag.
<em>	Applies italic formatting. You must close this tag.
&lt;	Less-than operator (<). Ensure that you include the final semicolon (;).
&gt;	Greater-than operator (>). Ensure that you include the final semicolon (;).
&#38;	Ampersand (&). Do not use &amp;. Ensure that you include the final semicolon (;).

Tag or Code	Description
&#42;	Do not use a literal "*" character in the body of a comment; instead, insert the HTML character code &#42;
&#x2014;	Em dash. Ensure that you include the final semicolon (;).
&#x99;	Trademark symbol (™) that is not registered. This character is superscript by default, so do not enclose it in <sup> tags. Ensure that you include the final semicolon (;).
&#xA0;	Nonbreaking space. Ensure that you include the final semicolon (;).
&#xAE;	Registered trademark symbol (®). Enclose this character in <sup> tags to make it superscript. Ensure that you include the final semicolon (;).
&#xB0;	Degree symbol. Ensure that you include the final semicolon (;).
@	Do not use an @ sign in an ASDoc comment; instead, insert the HTML character code: &#64;

## Using the ASDoc tool

The ASDoc tool, `asdoc`, is in the `flex\bin` directory of a Flex installation. To run the ASDoc tool, make sure that you either first change to the bin directory, or you add the bin directory to your system path.

To see a list of the command-line options available to the ASDoc tool, use the `-help list` option, as the following example shows:

```
asdoc -help list
```

By default, the ASDoc tool compiles its input files against the library SWC files in the `flex\frameworks\libs` directory in your Flex installation. If you must add additional SWC files to compile your code, you can add them by using the `library-path` option to specify the directory containing the SWC files:

```
asdoc ... -library-path+=C:\myLibs
```

You can also use a Flex Ant task to run the ASDoc tool. For more information, see [Using Flex Ant Tasks](#).

## Defining the input files to the ASDoc tool

Use the following options to specify the list of classes processed by the `asdoc` command: `doc-sources`, `doc-classes`, and `doc-namespaces`. The `doc-classes` and `doc-namespaces` options require you to specify the `source-path` option to specify the root directory of your files.

**Note:** The examples below assume that you installed Flex in the `c:\flex` directory on your machine. All the Flex source code is then available in the `C:\flex\frameworks\projects` directory.

The most basic example is to specify the path to a single class by using the `doc-sources` option, as the following example shows:

```
asdoc -doc-sources C:\flex\frameworks\projects\framework\src\mx\controls\Button.as
      -output framework-asdoc
```

This example generates ASDoc content for the Flex Button control shipped with Flex, and writes the output to the `flex\bin\framework-asdoc` directory.

You can specify multiple input class files, separated by spaces, as the following example shows:

```
asdoc -doc-sources C:\flex\frameworks\projects\framework\src\mx\controls\Button.as
      C:\flex\frameworks\projects\framework\src\mx\controls\ButtonBar.as
      -output framework-asdoc
```

The two previous examples use the `output` option to the directory that contains the ASDoc output. You can specify an absolute or relative path. In the previous example, the output directory is named `framework-output` relative to the current working directory. To view the output of your ASDoc build, open the `index.html` file in the output directory.

The `doc-sources` option takes a directory as an argument, as well as a file list. If you specify a directory, the ASDoc tool generates output for all files in the specified directory and any subdirectories. Use this option to build ASDoc output for all the files in the `C:\flex\frameworks\projects\framework\src\mx\controls` directory, as the following example shows:

```
asdoc -doc-sources C:\flex\frameworks\projects\framework\src\mx\controls
      -output framework-asdoc
```

You can specify multiple directories, separated by spaces.

Use the `doc-classes` option to specify the package and class name of a file to process. Use the `doc-classes` option with the `source-path` option, as the following example shows:

```
asdoc -source-path C:\flex\frameworks\projects\framework\src
      -doc-classes mx.controls.Button mx.controls.ButtonBar
      -output framework-asdoc
```

The `source-path` option adds directories to the source path. The ASDoc tool searches directories in the source path for the files to process. The value to the `doc-classes` option is a space delimited list of input files that use dot notation to specify the package name of each class.

You can combine the `doc-sources` and `doc-classes` options to specify the input to the ASDoc tool, as the following example shows:

```
asdoc -source-path C:\flex\frameworks\projects\framework\src
      -doc-classes mx.controls.Button mx.controls.ButtonBar
      -doc-sources C:\flex\frameworks\projects\framework\src\mx\validators
      -output framework-asdoc
```

In this example, you compile all the validator classes as well as the `Button` and `ButtonBar` components.

## Compiling dependent files

When performing a build, the ASDoc tool compiles the input files and also attempts to compile any dependent files referenced by the input files. For example, you specify class `A` as an input by using the `doc-classes` option. If class `A` imports class `B`, both class `A` and class `B` are compiled and included in the ASDoc output.

The following example specifies only the `mx.controls.Button` class as input:

```
asdoc -source-path C:\flex\frameworks\projects\framework\src
      -doc-classes mx.controls.Button
      -output framework-asdoc
```

The output of the build includes the `mx.controls.Button` class, plus any class referenced by the `Button` class, and any classes referenced by classes referenced by `Button`. The compiler uses the `source-path` option to locate these dependent classes.

If you set the `exclude-dependencies` option to `true`, dependent classes found when compiling the input classes are not documented. The default value is `false`, which means any classes that would normally be compiled along with the specified classes are documented.

The following example generates ASDoc content only for the `Button` class:

```

asdoc
  -source-path C:\flex\frameworks\projects\framework\src
  -doc-classes mx.controls.Button
  -output framework-asdoc
  -exclude-dependencies=true

```

Setting the `exclude-dependencies` option to `true` improves the performance of the ASDoc tool because you do not have to build output for all dependent classes.

**Note:** You cannot use `exclude-dependencies` with input class specified by the `doc-sources` option.

## Using a manifest file as input

If your source code is packaged for distribution as a SWC file, you can use a manifest file to define the content of the SWC file. You can use a manifest file as input to the ASDoc tool to specify the input file list as the following example shows:

```

asdoc -source-path C:\flex\frameworks\projects\framework\src
      -doc-classes FrameworkClasses
      -namespace http://www.adobe.com/2006/mxml
          C:\flex\frameworks\projects\framework\manifest.xml
      -doc-namespaces http://www.adobe.com/2006/mxml
      -output framework-asdoc

```

The preceding command line generates ASDoc content for all classes in the Flex framework.swc file. Notice that you specify the FrameworkClasses class file as input using the `doc-classes` option, and the manifest file by using the `doc-namespace` option. Most Flex SWC files are represented by a class file and a manifest file. Therefore, to build ASDoc for the SWC file, you specify both as input.

For more information on manifest files, see About manifest files.

## Excluding classes

All the classes specified by the `doc-classes`, `doc-sources`, and `doc-namespaces` options are documented, with the following exceptions:

- If you specified the class by using the `exclude-classes` option, the class is not documented. You must specify the package name of the files to omit, such as `mx.controls.Button`, separated by spaces.
- If the ASDoc comment for the class contains the `@private` tag, the class is not documented.
- If the class is found in a SWC, the class is not documented.

In the following example, you generate output for all classes in the current directory and its subdirectories, except for the two classes `comps\PageWidget` and `comps\ScreenWidget.as`:

```

asdoc -source-path . -doc-sources . -exclude-classes comps.PageWidget comps.ScreenWidget

```

The excluded classes are still compiled along with all the other input classes; only their content in the output is suppressed.

Use the `exclude-sources` option to exclude a file from being input to the compilation. This option is different from the `exclude-classes` option which you use to exclude a class from the output. However, the `exclude-classes` option still compiles the specified class. When you specify a class by using the `exclude-sources` option, the class is not even compiled.

**Note:** You can only exclude classes added by the `doc-sources` option, not classes added by the `doc-namespaces` or `doc-classes` options.

For example, the Flex mx/core package contains several include files that are not stand-alone classes, but files included by other classes. If you specify `doc-sources` for mx/core, you get compiler errors because the compiler tries to process all the files in the directory. Instead, you can use the `exclude-source` option to specify the full path to the files to ignore from the compilation, as the following example shows. Specify multiple paths separated by spaces, as the following example shows:

```
asdoc
  -doc-sources C:\a\flex\flex\sdk\frameworks\projects\framework\src\mx\core
  -exclude-sources C:\a\flex\flex\sdk\frameworks\projects\framework\src\mx\core\Version.as
```

## Excluding a namespace

To exclude an entire namespace from ASDoc, edit the `ASDoc_Config_Base.xml` file in the `asdoc\templates` directory of your Flex installation.

**Note:** Do not edit any other settings in the `ASDoc_Config_Base.xml` file. Set all other options from the `asdoc` command line.

For each namespace that you want to exclude, add a new `<namespace>` tag to the `<namespaces>` tag in the `ASDoc_Config_Base.xml` file, as the following example shows:

```
<namespaces hideAll="false">
...
  <namespace hide="true">my_custom_namespace</namespace>
</namespaces>
```

## Handling ASDoc errors

The ASDoc tool compiles all the input source files to generate its output. Therefore, your source code must be valid to generate ASDoc output. The ASDoc tool writes any compilation, such as syntax errors, errors to the console window.

Errors in ASDoc comments are not compilation errors but they do cancel the ASDoc build. For example, you can cause an error in an ASDoc comment by omitting a closing `</code>` or `</p>` tag. The source code still compiles, but ASDoc fails because it cannot generate the output.

The following example shows the error message for an ASDoc comment that omits a closing `</code>` tag:

```
[Fatal Error] :10:5: The element type "code" must be terminated by the matching end-tag
"</code>".
Encountered not well-formed text. Please see C:\flex\bin\framework-asdoc\validation_errors.log
for details.
```

The error message describes the condition that caused the error, and specifies to view the `validation_errors.log` file for more information. The `validation_errors.log` file contains additional information that describes the error and the location of the error in the input file.

Use the `lenient` option to configure the ASDoc tool to generate output even when it encounters an error in an ASDoc comment. When specified, the `lenient` option causes the tool to omit the incorrect ASDoc comment from the output, but to complete the build. The following example uses the `lenient` option:

```
asdoc -doc-sources C:\flex\frameworks\projects\framework\src\mx\controls\Button.as
  -lenient
  -output framework-asdoc
```

With the `lenient` option, you see the same error message, and the ASDoc tool writes the same information to the `validation_errors.log` file. But, the tool generates the output, as the following message shows:

```
[Fatal Error] :10:5: The element type "code" must be terminated by the matching end-tag
"</code>".
Encountered not well-formed text. Please see C:\flex\bin\framework-asdoc\validation_errors.log
for details.
Documentation was created in C:\flex\bin\framework-asdoc\
```

## Using a configuration file with the ASDoc tool

Depending on the number of input files, the input specification to the ASDoc tool can get complex. To simplify it, you can use a configuration file with the ASDoc tool. The configuration file is an XML file that lets you specify the command-line arguments to the tool in a file. You then run the tool as show below:

```
asdoc -load-config+=configFileName.xml -output framework-asdoc
```

By default, the ASDoc tool uses the `flex\frameworks\flex-config.xml` configuration file. This configuration file contains many settings, including the location of the Flex SWC files required to compile the input files. You can want to look at this file for an example of the types of information that you can write to a configuration file.

The previous example uses the `load-config` option to specify the name of the configuration file. Notice that this option uses the `+=` syntax to add `configFileName.xml` to the list of configuration files so that the tool still includes the `flex\frameworks\flex-config.xml` configuration file. If you use the `=` syntax to specify the name of the configuration file, you must also define much of the information in the `flex\frameworks\flex-config.xml` configuration file.

The following configuration file creates the same output as the example in the section “Using a manifest file as input” on page 23 to generates ASDoc content for all classes in the Flex `framework.swc` file:

```
<?xml version="1.0"?>
<flex-config xmlns="http://www.adobe.com/2006/flex-config">

  <compiler>
    <source-path>
      <path-element>C:\flex\frameworks\projects\framework\src</path-element>
    </source-path>
    <namespaces>
      <namespace>
        <uri>http://www.adobe.com/2006/mxml</uri>
        <manifest>C:\flex\frameworks\projects\framework\manifest.xml</manifest>
      </namespace>
    </namespaces>
  </compiler>

  <doc-classes>
    <class>FrameworkClasses</class>
  </doc-classes>

  <doc-namespaces>
    <uri>http://www.adobe.com/2006/mxml</uri>
  </doc-namespaces>
</flex-config>
```

For more information on configuration files, see [About configuration files](#).

## Options to the asdoc tool

The `asdoc` tool works the same way as the `mxmlc` compiler and takes all the same command-line options. For more information on `mxmlc`, see [Using the Flex Compilers](#).

The following table lists the command-line options specific to the `asdoc` tool:

Option	Description
<code>-doc-classes path-element [...]</code>	A list of classes to document. These classes must be in the source path. This is the default option.  This option works the same way as does the <code>-include-classes</code> option for the <code>compc</code> component compiler. For more information, see <i>Using compc</i> , the component compiler.
<code>-doc-namespaces uri manifest</code>	A list of URIs to document. The classes must be in the source path.  You must include a URI and the location of the manifest file that defines the contents of this namespace.  This option works the same way as does the <code>-include-namespaces</code> option for the <code>compc</code> component compiler. For more information, see <i>Using compc</i> , the component compiler.
<code>-doc-sources path-element [...]</code>	A list of files to document. If a directory name is in the list, it is recursively searched.  This option works the same way as does the <code>-include-sources</code> option for the <code>compc</code> component compiler. For more information, see <i>Using compc</i> , the component compiler.
<code>-examples-path path-element</code>	Specifies the location of the include examples used by the <code>@includeExample</code> tag.
<code>-exclude-classes string</code>	A list of classes not documented. You must specify individual class names. Alternatively, if the ASDoc comment for the class contains the <code>@private</code> tag, is not documented.
<code>-exclude-dependencies true false</code>	Whether all dependencies found by the compiler are documented. If <code>true</code> , the dependencies of the input classes are not documented.  The default value is <code>false</code> .
<code>-exclude-sources path-element [...]</code>	Exclude a file from compilation. This option is different from the <code>-exclude-classes</code> option which you use to exclude a class from the output. However, the <code>-exclude-classes</code> option still compiles the specified class.  You can only exclude classes added by the <code>doc-sources</code> option, not classes added by the <code>doc-namespaces</code> or <code>doc-classes</code> options.
<code>-footer string</code>	The text that appears at the bottom of the HTML pages in the output documentation.
<code>-left-frameset-width int</code>	An integer that changes the width of the left frameset of the documentation. You can change this size to accommodate the length of your package names.  The default value is 210 pixels.
<code>-lenient</code>	Ignore XHTML errors (such as a missing <code>&lt;/p&gt;</code> tag) and produce the ASDoc output. All errors are written to the <code>validation_errors.log</code> file.
<code>-main-title "string"</code>	The text that appears at the top of the HTML pages in the output documentation.  The default value is "API Documentation".
<code>-output string</code>	The output directory for the generated documentation. The default value is <code>"asdoc-output"</code> .
<code>-package name"description"</code>	The descriptions to use when describing a package in the documentation. You can specify more than one <code>package</code> option.  The following example adds two package descriptions to the output:  <pre>asdoc -doc-sources my_dir -output myDoc -package com.my.business "Contains business classes and interfaces" -package com.my.commands "Contains command base classes and interfaces"</pre>

Option	Description
<code>-strict=false true</code>	Disable strict compilation mode. By default, classes that do not define constructors, or contain methods that do not define return values cause compiler failures. Set <code>strict</code> to <code>false</code> to override this default and continue compilation.
<code>-templates-path <i>string</i></code>	The path to the ASDoc template directory. The default is the <code>asdoc/templates</code> directory in the ASDoc installation directory. This directory contains all the HTML, CSS, XSL, and image files used for generating the output.
<code>-window-title "<i>string</i>"</code>	The text that appears in the browser window in the output documentation.  The default value is "API Documentation".

The `asdoc` command also recognizes the following options from the `compc` component compiler:

- `-source-path`
- `-library-path`
- `-namespace`
- `-load-config`
- `-actionscript-file-encoding`
- `-help`
- `-advanced`
- `-benchmark`
- `-strict`
- `-warnings`

For more information, see [Using mxmhc](#), the application compiler. All other application compiler options are accepted but ignored so that you can use the same command-lines and configuration files for the ASDoc tool that you can use for `mxmhc` and `compc`.