AS2Doc Documentation

Edition: December 2004
Version: 1.0.5
http://www.as2doc.com

AS2Doc 1.0

Mirell Development
Heinrich-Barth-Str.1
D-20146 Hamburg
Germany
http://www.mirell.de

This document should provide information about the how to work with the various features AS2Doc has to offer.

It guides through the installation and configuration, explains user interface behavior, how to write comments within your code and gives help on the generation of various output formats AS2Doc can produce.

The appendix contains quick help on problems and frequently asked questions (FAQ).

1. Installation

This chapter explains how to install AS2Doc on your target computer and how to register it for initial use.

1.1 General Installation

To install AS2Doc:

1. Double-Click the AS2Doc setup installer
2. Follow the onscreen instructions. The installation program prompts you to enter the required information.
3. Once the installation program is finished you can start AS2Doc using the start menu and continue with the product registration.

The installation process is the same if you are upgrading to a new version unless it is explicitly noted that you have to uninstall a prior version within the installation program.

1.2 Product Registration

Upon ordering you have received a **Product Key** and a **Serial Key**.

If you start up AS2Doc for the first time you will have to enter the **Username** you have purchased your license with and both keys into the appropriate fields in the form.

Click on the “Register” button once finished to finally unlock AS2Doc and start working with your program.

AS2Doc can not be used before it is successfully unlocked.

1.3 System Requirements

Before installing AS2Doc, make sure your computer is equipped with the following hardware and software.

Microsoft® Windows

- Microsoft® Windows Millennium Edition, Windows NT® 4.0 with Service Pack 6, Windows 2000 with Service Pack 2, Windows XP Professional or Home Edition, Windows XP Tablet PC Edition
- 300 Mhz or higher
- 64 MB of RAM (128 MB recommended)
- Disk space to cover AS2Doc (around 1 MB) and generated Documentation
- Minimum 800x600 pixel resolution
- Internet Explorer 4 or higher
- MSXML Version 3 or higher (automatically installed with Internet Explorer)

2. Getting Started

This chapter intends to give a short and basic view upon the inner processes AS2Doc is using in order to produce your documentation to understand the various options you have on the way to your final documentation. A short tutorial introduces a quick “ready-to-go” example.

2.1 Introduction

AS2Doc is a generator for API (Application Programming Interface) documentation which is automatically derived from the source code of ActionScript 2.0 based classes (.as) and comments contained within.

AS2Doc offers a GUI (Graphical User Interface) to configure various options of the generation process but also supports automated execution using a CLI (Command Line Interface), too.

AS2Doc was inspired by the popular tool “JavaDoc ™” and contains many approaches similar to it especially in the way of commenting the source code.

2.2 How AS2Doc works

AS2Doc works in three steps:

Step 1:

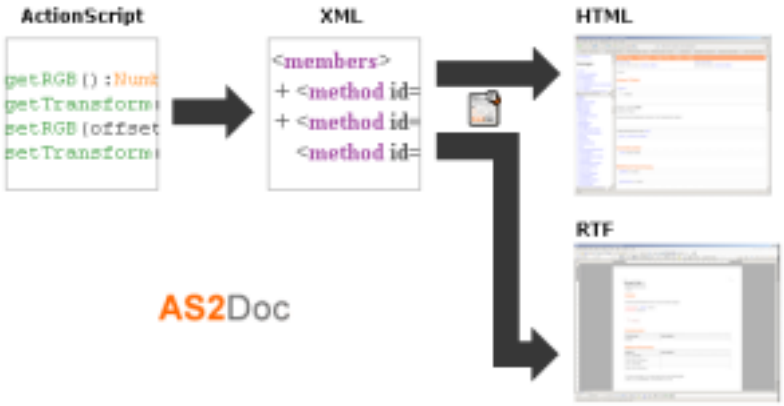
AS2Doc reads your source files from a specified source file or directory.
Internally a tree of all classes and corresponding associations is created.

Step 2:

AS2Doc creates a XML file based on your configuration you set for the generation process.

Step 3:

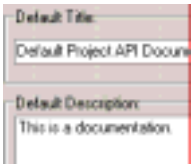
AS2Doc takes the XML file from Step 2 and transforms it using one of the various AS2Doc Styles (XSL Stylesheets) into the output format you have selected.



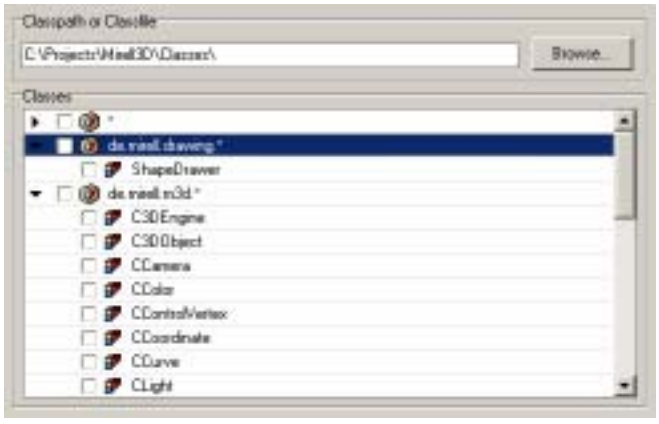
2.3 A simple tutorial

This simple tutorial will show you the simple steps to create HTML documentation for a directory which contains a couple of AS 2.0 based classes (.as).

1. Start AS2Doc (register the product if needed, *See 1.2 Product Registration*)
2. Enter a title for your project on the first screen and a small description. You can leave the default entries if you simply want to try the features at this moment.



3. Click on the “Source” tabulator to open up the source settings.
4. Click on the button “Browse” and select the directory containing your classes and click “OK” within the dialog.
5. AS2Doc will list all classes and interfaces it will find within any (.as) source code files in the directory you selected as well as it’s subdirectories.



6. You can (de)select individual classes or whole packages using the checkboxes next to a class entry or using the popup menu which appears if you click the right mouse button.
7. Click on the “Output” tabulator to configure the destination settings.

8. Click on the “Browse” button and select a directory where your generated documentation should be saved. Any prior generated files will be overwritten.
9. Make sure you select the “Default AS2Doc HTML Style” in the list as it will produce our final HTML documentation.
10. Click on the “Generate” button now. You will see AS2Doc switching to the “Log” page and display information about the generation process.
11. Once the process is finished you can press the “View” button to show the index page of the HTML Documentation. It will open your default browser for HTML files.

3. User Interface

This chapter will explain each element of the graphical user interface (GUI) witch can be used to configure documentation project settings, save/load projects and generate documentation.

The project files saved from within the GUI can be used from the command line interface as a parameter. (*See 4. Command Line*)

AS2Doc is divided into five pages and general control buttons at the bottom (About, View, Generate and Exit).

The following sections explain the individual elements of each page which can be accessed by clicking on the appropriate tabulator in the top of the program screen.

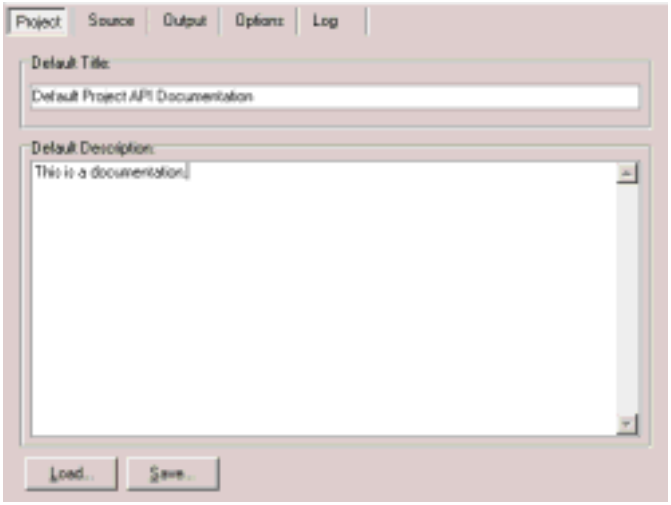
3.1 Project Settings

The project settings page allows you to define default project title and description and the ability to load an existing project from or save it on your computer.

The title and description will be used by the output styles if needed to produce title pages, chapter titles or similar display using the information you entered.

The description offers a small subset of HTML. (*See 5.1.2 Description HTML Subset*)

Mind that the usage of {@link …} has no context and fully qualified identifiers need to be used if you link to packages or classes. If you only target to output HTML documentation you can use full HTML.

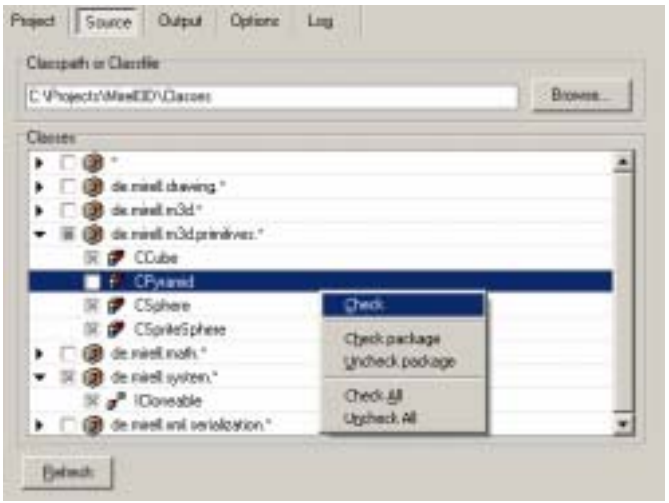


"Load" Button: Opens a dialog to select a valid project file (default .xml extension) to load. Any existing settings of the current project are removed.

"Save" Button: Opens a dialog to enter a filename to save the project file to.

3.2 Source Settings

The source page selects the classes and/or packages you want to appear in the generated documentation.



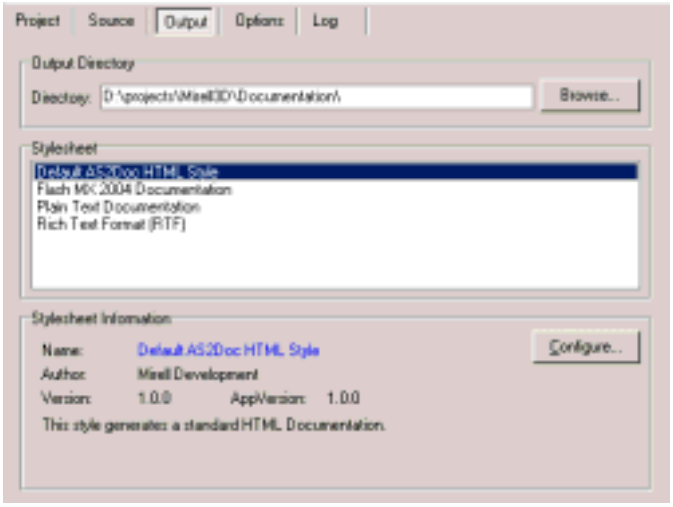
"Browse" Button: Opens a dialog to select a directory or a classfile (.as). After you have closed the dialog your selection will be searched for valid AS2.0 classes.

Class List: Once you have selected a valid class directory or classfile this component will list your classes and interfaces it has found listing the package and corresponding contained classes. By clicking the checkboxes next to an entry, you can define which elements should appear in the final documentation and which should be skipped. To ease selection you can press the right mouse button within the list to open a popup menu where you have advanced options for selection.

"Refresh" Button: Initiates a new search in the directory or classfile specified as the source. This is useful if your source is a developer directory and an updated of the classlist is required due to adding new classfiles or moving/renaming existing classfiles for example.

3.3 Output Settings

The output page controls where the generated documentation will be created and which AS2Doc Output Style will be used for generation.



"Browse" Button: Opens a dialog where you can select a directory to act as the target for the generated documentation.

"Stylesheet" List: Upon program startup, AS2Doc reads available Output Styles from a folder named "styles" within the AS2Doc working directory. The styles it finds are listed here. Click on a style in order to select it as the active output style for generation.

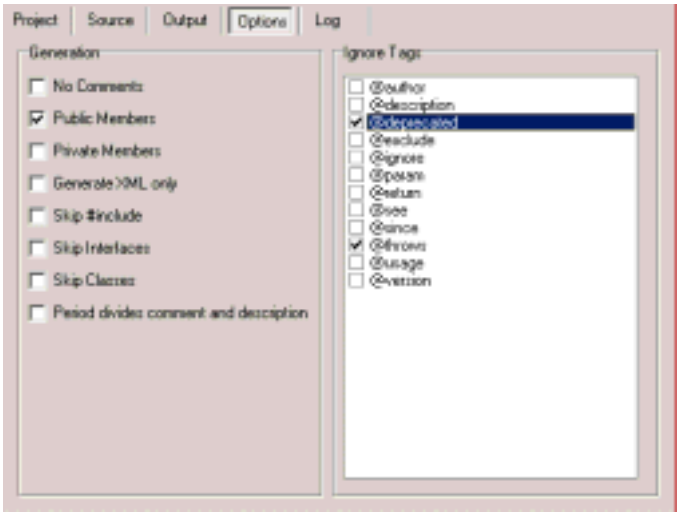
"Stylesheet Information": This area shows information and a clickable label to the style author's website if available and a short description of the style and output format.

Name	Name of the style
Author	Author of the style
Version	Version of the style
AppVersion	Version of AS2Doc the style was build for

"Configure" Button: Depending on the style you selected this button will be shown if the style supports and further configuration (like custom colors, text replacements, footers etc.). It opens a dialog where you can configure style dependent options which are saved within your project. These settings are not discarded if you select a different style.

3.4 Options

The options page allows configuring style independent settings of the generation process.



"Ignore Tags" List: Check one of the listed block comment tags which will be skipped and not read from the source files if encountered and completely discarded. Any actions they trigger (e.g.: @ignore, @exclude) are not processed.

"Generation" Checkboxes:

No Comments All comments including appropriate tags are not read from the source files.

Public Members The default checked option will generate all members with public class scope.

Private Members Includes members in private scope into the final documentation.

Generate XML only Once generated a XML file will be saved in the output directory which is usually used by the selected style.

Skip #include Does not resolves the #include compiler directive

Skip Interfaces Any interfaces encountered within the source are not generated to the final documentation.

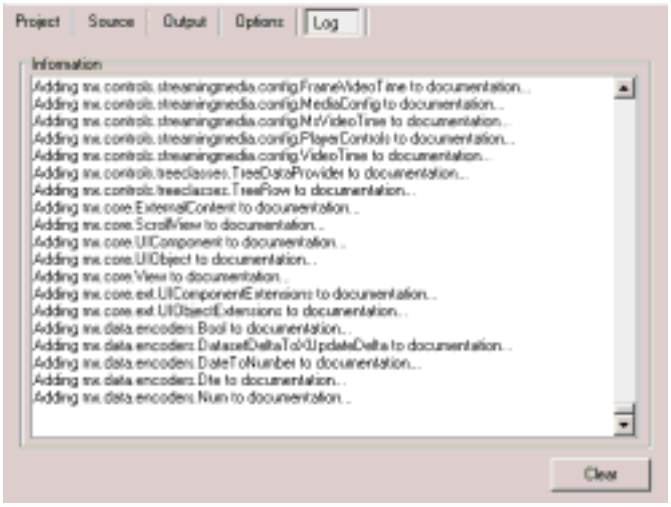
Skip Classes Any classes encountered within the source are not generated to the final documentation.

Period divides comment and description
AS2Doc is strict regarding the separation of a short and a long description of a code element. Usually any comment without a tag identifier in the form of @tag is regarded as a short description with a more detailed and long description within the @description tag. Since some projects we encountered used standard "JavaDoc ™" descriptions which define that the short and long description is divided by the first period followed by a whitespace character these descriptions would be combined to a very long short description in the output of AS2Doc.

To enable AS2Doc to automatically divide the descriptions according to this schema check this option.

3.5 Log View

The Log page contains a display for log messages generated on errors and during the generation of your documentation. It will list all files processed and the current status of the generation. A progress bar beneath the log display will show you the progress of the generation.



"Clear" Button: Clears any log entries within the log display.

3.6 About, View, Generate and Exit Buttons

"About" Button: Show information about the version of AS2Doc you are using

"View" Button: If you have successfully generated documentation this button will be enabled. Once clicked, it will open your default viewer for the documentation's main file if available.

"Generate" Button: Click the button to start the generation of the documentation. If AS2Doc is missing any settings you will be notified to add/configure these and press the button again afterwards.

"Exit" Button: Quits the application discarding any changes made to the project since last save,

4. Command Line

The command line interface allows control over all aspects AS2Doc offers using command line parameters. This way you can automate the generation in an editor of your choice without using the GUI or automate generation into multiple output formats or in any other scenario which makes use of command line parameters.

Command line parameters are separated by a space and marked by a preceding "-".

If you supply any command line parameter to AS2Doc it will not start in command line interface mode (*CLI*).

In order to get an overview of the parameters available open a command prompt and change to the directory AS2Doc was installed to (Example: "*cd c:\Program Files\Mirell\AS2Doc Pro*"), then start AS2Doc using: "*as2doc.exe –help*"

4.1 Arguments

General Usage: *as2doc.exe [project.xml] [Options]*

AS2Doc differs between three modes.

1. You supply a project file only (example: "*as2doc.exe myproject.xml*")
2. You supply no project file, only parameters
3. You supply both project file and parameters (example: "*as2doc.exe pro.xml –private -quiet*")

In the last case the parameters override any options set within the project.
In all cases the minimum requirements to start the generation is to specify a valid source and destination.

[Options] include:

-s <path|file>

Specify a classpath or a single classfile to read. If you specify a classpath, it is searched recursively for *.as classfiles.

-d <path>

Specify the directory to generate the documentation to. Files will be overwritten.
Ignored if you specify the *–xml* parameter since no documentation will be generated besides the XML file in that case.

-g <pkg|class>

By default, AS2Doc generates all classes found in the source specified by the *–s* parameter.
The *–g* parameter is useful if you want to generate specific packages and/or classes out of those found using the *–s* parameter. The list of packages and/or classes/interfaces in any combination should be separated by a space.

To mark a package and all classes within use "*my.package.**"
To mark a single class or interface supply the fully qualified identifier "*my.package.myclass*"

Examples: …"*-g com.as2doc.examples.* com.as2doc.AClass*"…

- xml <file>

Generates XML based documentation in a single file. The file is usually used by AS2Doc to further process it using an output style. Any destination specified within the project file or using *–d* is discarded.

-style <style.xml>

Specify the full path to the output style to be used for generation.
The default is the "*Default AS2Doc HTML Style*" if this option is omitted or nothing else was saved in a project file used. (Example from AS2Doc installation directory to call the RTF style: "*as2doc.exe myproject.xml –style styles/AS2Doc/RTF/style.xsl*")

This parameter can also be used in conjunction with the *–styleoptions* parameter to display all advanced options of the specified style.

-run

Automaticly spawn the default viewer once the documentation was generated. Same effect as the "View" button within the GUI.

-nopublic

By default all members in public scope will be generated. If you don't want this add the *–nopublic* option.

-private

Generates members in private scope if used.

-styleoptions

List all available advanced options of the used style which can be passed on the command line. It is the equivalent to the "Configure" Dialog in the GUI.

-quiet

Disable output of log messages and errors. AS2Doc will be started and generates no output on the command line.

-@<tag>

Disable parsing of <tag>. (Example to skip all @see block comment tags: "*as2doc.exe myproject.xml -@see*")

Add multiple *-@<tag>* parameters if you want to ignore multiple tags.

-nocomments

Discards reading any comments including comment tags from the source files.

-skipinclude

Does not process any *#include* compiler directives.

-skipclasses

Skips all classes found in the source path.

-skipinterfaces

Skips all interfaces found in the source path.

-descatperiod

The short description will end at the first period followed by a whitespace or tab character in the source file. This mimics "JavaDoc™"-like behavior.

-help

Display a short help and overview of the command line parameters of AS2Doc and the default AS2Doc output style.

4.2 Examples

Some examples making use of the command line interface of AS2Doc:

1. Generate a standard HTML documentation without specifying a project file:

as2doc.exe –s c:\myproject\classes –d c:\myproject\documentation\html\ -a:title "My Project API Documentation" –a:description "Description for my project"

2. Generate a RTF documentation from a standard project file specifying to only generate classes in the root package, ignoring the @example tag and overriding the title from the project file.

*as2doc.exe c:\myproject\myproject-doc.xml –g * -@example –a:s "My Project Root Classes Documentation"*

3. Generate a standard HTML documentation without frames, a custom css file, also private members based on an existing project file.

as2doc.exe c:\myproject\myproject-doc.xml –s:frames false –s:css c:\myproject\documentation\blue-skin.css –private

5. Documentation Comments

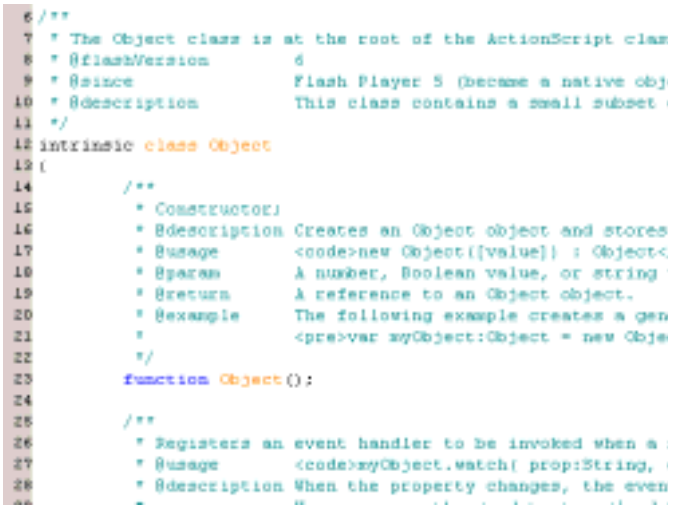
In order to be able to process your source files and recognize certain features AS2Doc will parse a classfile and its comments for “Documentation Comment Tags” of the form “@tagidentifier”.

Since the “JavaDoc ™” format has become a commonly used documentation standard and its existing specifications cover it very detailed the following sections will only cover the differences and additions offered by the comment system used by AS2Doc.

See the “JavaDoc Tool Home Page” for detailed informations:
http://java.sun.com/j2se/javadoc/reference/docs/

5.1 Commenting the Source Code (AS2Doc Additions)

You can include documentation comments ("doc comments") in the source code, ahead of declarations for any class, interface, method, constructor, field or property (methods specified using get/set keywords).



A doc comment consists of the characters between the characters `/**` that begin the comment and the characters `*/` that end it. Leading asterisks are allowed on each line and are described further below. The text in a comment can continue over multiple lines.

```
/**
 * This is the typical format of a simple documentation comment
 * that spans two lines.
 */
```

AS2Doc also allows the following variation:

```
// This is a simple comment spanning
// two lines aswell.
// @description   Tags are fully supported
```

and

```
/*
 This is also a valid comment.
 @description With this description tag being recognized either.
*/
```

You should conform to the first example provided to have well-formed comments. The alteration is available to cover some exotic cases.

First Sentence
AS2Doc recognizes the first sentence as the short description which should contain a summary sentence describing the following code element. It is not preceded by any comment tag "@<identifier>". The long description is specified using the "@description" tag.

```
/**
 * A short description of the code element.
 * @description More detailed long description of the element.
 */
```

5.1.2 Description HTML Subset

Not all output styles offer full HTML support.

They often (besides the HTML Style) only support a small subset of HTML. In order to maintain compatible upon the various output formats, you should limit your comments to the following set of HTML tags:

br, a, b, i, u, table, th, tr, td, code, pre, img

Special notes:

table please use `<table cellspacing="0" cellpadding="0" border="0">` in order for all styles to recognize tables.

th Make use of the *th* tag for table headings to be recognized by all styles correctly.

code Use this tag to mark a characters as program code, it differs from the pre tag by being inline with the surrounding text if it does not span multiple lines. Otherwise it has the same effect as the pre tag.

pre Use this tag to mark a block of program code. The program code within the pre tag is white-space aware and line breaks are preserved in the output. The program code marked by the pre tag is shown in a separated block and any preceding or following elements do not appear on the same line.

Line breaks are not recognized and need to be created by adding the `<br>` HTML tag for a break.

This is different from anything written between `<code>` and `<pre>` tags. Line breaks are preserved and copied the generated documentation. AS2Doc also adds correct indentation.

Some examples:

```
/**
 * @example This code appears in a separate block preserving indents:
 *          <code>
 *          var myObject = new Object();
 *          myObject.testFunc = function ()
 *          {
 *                // This will appear intended
 *                trace(this.toString());
 *          }
 *          </code>
 *          Within the regular comment you need to use <br>
 *          to force a manual line break as in HTML.<br>
 *          Any tag, written between &lt;code&gt; tags will be escaped:
 *          <code>
 *          // This inline code comment will show tags -> <b>, <i> and <u>
 *          <?xml version="1.0"?>
 *          </code>
 *          The usage of &lt;code&gt; differs if it does not span multiple
 *          lines in the comment <code>like this text</code> appears not
 *          as a block but still formatted as code and inline of the text.
 */
```

If you target to only use HTML as an output format, you can use full HTML within the description including any “style” attributes for richer display of your description text.

The comments also allow the use of inline comment tags {@link …} and {@docRoot}. (*See 5.2.2 Inline Tags*)

5.2 AS2Doc Tag Reference

AS2Doc parses special tags when they are embedded within a comment.

These tags enable you to generate a complete, well-formatted API from your source code.

The tags start with an "at" sign (@) and are case-sensitive “@<identifier>” -- they must be typed with the uppercase and lowercase letters as shown.

A tag must start at the beginning of a line (after any leading spaces and an optional asterisk) or it is treated as normal text.

By convention, tags with the same name are grouped together.
For example, put all @see tags together.

A list of tags currently supported by AS2Doc:

@author
@deprecated
@since
@version
@param
@return
@throws
@see

{@link}
{@docRoot}

(AS2Doc adds additional tags besides the standard JavaDoc™ based tags)

@description
@example
@ignore
@exclude
@usage

5.2.1 Block Tags

Block Tags can be placed only in the tag section that follows the short description. Form: @tag

@author *name*

Specify the author of the commented code element. Multiple @author tags are allowed and will concatenate with a comma "," in the final output.

Example:
```
/**
 * @author  Mirell Development
 * @author  Second Author
 */
```

@deprecated *deprecated-text*

Add a comment indicating that this API should no longer be used (even though it may continue to work). The deprecated-text is moved ahead of the main description, summary pages and index, placing it in italics and preceding it with a bold warning: "Deprecated".

Example:
```
/**
 * @deprecated Since Flash 6, see {@link #bProperty}
 */
```

@usage *usage-text*

Specify the usage of the commented code element.

Example:
```
/**
 * @usage   <pre>myTest = new CTest(p1[, p2]);</pre>
 */
```

@since *since-text*

This tag specifies that this change or feature commented has existed since the software release specified by *since-text*. Multiple @author tags are allowed and will concatenate with a comma "," in the final output.

Example:
```
/**
 * @since   API 2.3
```

```
 * @since   AOP SDK 5.1
 */
```

@version *version-text*

This tag is intended to hold the current version number of the software that this code element is part of. Multiple @author tags are allowed and will concatenate with a comma “,” in the final output.

Example:

```
/**
 * @version API 2.5
 */
```

@ignore

This tag instructs the AS2Doc parser to ignore any comments and tags previously read for this code element and clear the comment buffer. Use this to avoid certain parts of a comment to be processed.

Example:

```
/**
 * This text might include some license information or other
 * data not intended to appear in the documentation or be used
 * as the short description of the following code element.
 * @description Tags are also discarded by the @ignore tag
 *
 * @ignore
 *
 * The preceding text is skipped and this is the short description.
 */
```

@exclude

If AS2Doc encounters an @exclude tag within a comment the following code element is excluded from the final documentation.

Note: You can reverse the effect and generate all elements marked with @exclude by configuring AS2Doc to ignore the @exclude tag.

@description *text*

This tag contains the detailed long description *text* of the following code element. Multiple @description tags are combined into one description in the final documentation.

Example:

```
/**
 * A short description of the code element.
 * @description   The long description of the code element spanning
 *                multiple lines and explaining the element in detail.
 */
```

@param *[identifier] description*

This tag is only valid in a doc comment of a method or constructor. This includes property write access methods (“*function set myProperty(Value: String) {}*”.

The *description* may span multiple lines. AS2Doc allows the *identifier* to be skipped if the following applies: a) @param tag order reflects method parameter order b) All @param tags do not contain the *identifier* part.

Example:
```
/**
 * A short description of the code element.
 * @param   Value The new state of the light
 */
public function set LightState(Value: Boolean) {}

/**
 * A short description of the code element.
 * @param   Number of options
 * @param   Default option value
 * @param   Resets options to string specified in <code>b</code>
 *          if <code>True</code>.
 */
public function setOptions(a: Number, b: String, c:Boolean) {}
```

@return *description*

The tag is only valid in a doc comment of a method or a getter method. ("*function get myProperty(): String*").

The *description* may span multiple lines. It should describe the return type and the permissible range of values returned by the method.

Example:
```
/**
 * @return  The combination of the current position
 *          and direction within a boundry of 0-100.
 */
```

@throws *class-name description*

The *class-name* is the name of the exception that may be thrown by the method. This tag is valid only in the doc comment for a method or constructor.

Multiple @throws tags can be used in a given doc comment for the same or different exceptions.

Example:
```
/**
 * @throws  IllegalArgumentError    Description of when this error
 *          occours.
 */
```

@see *reference*

Define a link or text entry that points to *reference*. A comment may contain any number of @see tags, which are all grouped under the same heading. The @see tag has three variations; the third form below is the most common.

For inserting an in-line link within a sentence to a package, class or member, see {@link}.

@see *“string”*

Add a text entry for string. No link is generated. The string is a book or other reference to information not available by URL.

Example:
```
/**
 * @see     “Book about this API”
 */
```

@see *label*

Adds a link as defined by *URL#value*. The *URL#value* is a relative or absolute URL.

Example:
```
/**
 * @see     <a href="index.html#overview">Class Overview</a>
 */
```

@see *package.class#member label*

Add a link, with visible text label, that points to the documentation for the specified name that is referenced. The *label* is optional; if omitted, the name appears instead as the visible text, suitably shortened

Use the label when you want the visible text to be different from the auto-generated visible text.

package.class#member is any valid program element name that is referenced, a package, class, interface, constructor, method, field or property name, except that the character ahead of the member name should be a hash character (#). The *class* represents any class or interface. The *member* represents any constructor, method, field or a property. If this name is in the documented classes, AS2Doc will automatically create a link to it.

label is optional text that is visible as the link's label. The *label* can contain whitespace. If label is omitted, then package.class.member will appear suitably shortened relative to the current class and package.

A space is the delimiter between *package.class#member* and *label.*

Example (see tag refers to watch method of class *Object*):
```
/**
 * @see     Object.watch
 */
```

Since ActionScript 2.0 does not support overloading this referencing differs from that used by the “JavaDoc ™”-tool. Following “*()*” parentheses should not be provided.

This *package.class#member* name can be either fully-qualified, such as “*mx.core.View#draw*“ or not, such as “*View#draw*” or “*#draw*”. If less than fully-qualified, AS2Doc searches for it the same way as during compilation:

- the current class or interface
- any enclosing classes and interfaces, searching closest first
- any superclasses and superinterfaces, searching closest first
- the current package

- any imported packages, classes and interfaces, searching in the order of the import statement

Several Examples show the resulting output link (context of a class named *mx.data.TestClass*):

```
/**
 * @see mx.core.View                 //    mx.core.View
 * @see mx.core.View The view class  //    The view class
 * @see TextField                    //    TextField
 * @see TextField#getDepth           //    TextField.getDepth()
 * @see mx.core.ScrollView#draw      //    mx.core.ScrollView.draw()
 * @see View.DEF_HEIGHT              //    mx.core.View.DEF_HEIGHT
 * @see TestClass#getValue           //    getValue()
 * @see #getValue                    //    getValue()
 * @see <a href="a.hrml">Home A</a>  //    Home A
 * @see "Book about an API"          //    "Book about an API"
 */
```

5.2.2 Inline Tags

Inline Tags can be placed anywhere within comments of block tags. Inline tags are denoted by curly braces: {@tag}

```
/**
 * @description   This comment uses an inline tag to link to a
 *                different {@link my.package.class class}.
 */
```

{@docRoot}

The tag represents the relative path to the generated document's (destination) root directory from any generated document.

It is useful when you want to include a file, such as a copyright page or company logo, which you want to reference from all generated documents. Linking to the copyright page from the bottom of each page is common.

The reason this tag is needed is because the generated docs are in hierarchical directories, as deep as the number of sub-packages. This expression (if default HTML Output Style is used):

```
<a href="{@docRoot}copyright.html">
```

would resolve to:

```
<a href="../../copyright.html">      // for mx/core/View.as
```

If a style consists of only a single document or multiple documents in the same directory the tag will be substituted correctly by AS2Doc.

Linking images or HTML pages from documentation generated using the HTML Output Style is obvious, other styles will translate these references in a Style respective/appropriate way. (The RTF style for example converts HTML <img> tags to external image references inside the RTF and links to HTML pages as hyperlinks within the document)

```
/**
 * @description   This comment uses an inline tag to show an image
 *                within the text which resides in the root of
 *                the documentation directory: <br>
```

```
 *                <img src="{@docRoot}mylogo.jpg" border="0" alt="">
 */
```

{@link *package.class#member label***}**

The tag inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class.

This tag is very similar to @see -- both require the same references and accept exactly the same syntax for *package.class#member* and *label*.

The main difference is that {@link} generates an in-line link rather than placing the link in the "See Also" section.

Also, the {@link} tag begins and ends with curly braces to separate it from the rest of the in-line text. If you need to use "*}*" inside the label, use the HTML entity notation `&#125;`

There is no limit to the number of {@link} tags allowed in a sentence.

For example, here is a comment that refers to the *unwatch* method of the class *Object*:

```
/**
 * @description   Use the {@link Object#unwatch} method.
 */
```

We plan to include support for additional inline tags in future updates.

5.3 Example

Class / Interface example:

```
/**
 * A class representing a window on the screen.
 * @description   For example:
 *                <code>
 *                Window win = new Window(parent);
 *                win.show();
 *                </code>
 *
 * @author        Hei Ho
 * @version       2.4
 * @see           de.mirell.math.CQuaternion
 * @see           de.mirell.as2.lang.Object
 */
class Window extends BaseWindow {
   ...
}
```

Field example:

```
/**
 * The X-coordinate of the component.
 *
 * @see #getLocation
 */
public var x: Number = 1263732;
```

Property example:

```
/**
 * Sets the X-coordinate of the component.
 * @param   Value New x-coordinate from 0-100 pixels
 * @see #x
 */
public function set xPosition(Value: Number) {
   ...
}
```

Constructor and Method example:

```
/**
 * Returns the character at the specified index.
 * @description   An index ranges from <code>0</code>
 *                to <code>length() - 1</code>.
 *
 * @param         index  the index of the desired character.
 * @return        the desired character.
 * @throws        StringIndexOutOfRangeException
 *                if the index is not in the range <code>0</code>
 *                to <code>length()-1</code>.
 * @see           String#toString
 */
public function charAt(index: Number): Number {
   ...
}
```

6. AS2Doc Styles

AS2Doc uses output styles to generate your final documentation. These styles are nothing else than XSL Stylesheets. You should be familiar with XSL if you want to write custom output styles.

You can learn more about XSL at http://www.w3.org/xslt

AS2Doc uses them to process the generated XML Documentation and output your documentation in the specific output style.

The styles must reside in a subdirectory of the as2doc executable called “styles”.
If you browse your installation directory you should note several default styles bundled with the AS2Doc distribution.

The following sections will explain the structure and behavior of these styles to allow you to develop your own custom output formats and give an overview of each bundled style of the AS2Doc distribution.

6.1 Creating custom Output Styles

An AS2Doc Output Style typically consists of several XSL Stylesheets and optional resource files needed by the style.

The files must reside within a subdirectory of the as2doc executable called “styles” followed by a directory named after the vendor/developer and any further subdirectories like the output format for example. (Default AS2Doc Style: “*styles/AS2Doc/HTML/*”

The “*styles*” subdirectory is searched recursively by AS2Doc on startup for files named “*style.xsl*” which marks the main information and output pipeline file for an AS2Doc Style.

This is the only file AS2Doc will recognize and the minimum to allow a style to appear in the AS2Doc styles list or be used on the command line interface.

In order to understand the idea behind the pipeline file mind the process of generating documentation:

Read Source Code >> Generate a XML Documentation >> Transform the XML Documentation using a given Output Style >> Final Document(s)

The “*style.xsl*” output file contains two main sections and is a valid XML document with special tags explained in the following style reference and the option to use XSL transformations within to alter this information.

The first section contains the name, version, AS2Doc version the style was build for, a description, author information, author homepage and a list of style parameters which can be configured from AS2Doc.

The second section is a list of `<file>` tags specifying the “generation pipeline”. Each `<file>` tag represents a file copy action or further XSL Translation. This way, further XSL Stylesheets or resource files (images, css etc.) can be used to create files in the documentation output directory.

AS2Doc reads the “*style.xsl*” file and translates it with the XML Documentation initially created. (So the `<file>` action list can be altered using XSL depending on the documentation AND on the parameters passed to the “*style.xsl*” template.)

The resulting file is stored temporary and acts as a list of copy and translation operations AS2Doc will execute in the order of appearance.

You can easily create custom copies of the default styles and customize them to your needs by copying them into a new directory within the “*styles*” folder and start editing instead of starting from scratch.

The following section explains the structure of the “*style.xsl*” file.

In future versions AS2Doc will introduce an “*as2doc*” XML namespace to have advanced functionality not available with regular XSL including binary file handling and more.

6.1.1 Style Reference

This is the minimum XML Structure required for a valid “*style.xsl*” pipeline file:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="no" media-type="text/xml" omit-xml-
declaration="no"/>
<xsl:template match="/">
<generation>
  <information>
      <author>Company/Author</author>
```

```
      <version>1.0.0</version>
      <appVersion>1.0.0</appVersion>
      <name>Format Documentation</name>
      <description>Generates Format documentation.</description>
      <url>http://www.as2doc.com</url>
  </information>
</generation>
</xsl:template>

</xsl:stylesheet>
```

The file is a typical XSL Stylesheet with the XML output method.

It contains one root node <generation>.
The root node can contain nodes with only one <information> and one or more <file> nodes.

Since the file is a valid XSL Stylesheet you can include various transformations in the context of the raw XML Documentation. (Generate <file> nodes for each class or package, copy resource images etc.)

1) <information>

The tag contains a collection of tags which supply a data about the output style. The available tags are:

<author>	The name of the author or company creating this style.
<version>	The version number of the current style.
<appVersion>	The version number of AS2Doc this style was build with. (To detect incompatible style formats between major new or older versions)
<name>	The short name of the style.
<description>	[optional] A short description of the style (max 400 characters)
<url>	[optional] A hyperlink to the author's style website.

Style Parameters

AS2Doc supports customization of the style's behavior using "*Style Parameters*". They are defined in the node named <parameters> within the <information> node and are added to the resulting XML Documentation with values supplied by the user.

The node contains one or more <parameter> nodes which define the type of the parameter which can be configured by a user from the command line or the graphical user interface.

Stylesheet parameters are passed on the command-line using *-s:<name> <parameter>*
In the GUI, parameters can be edited using the style configuration dialog.

```
<parameter name="identifier" module="line|color|bool|text|filename"
description="Parameter description"/>
```

The required attributes are:

name	Defines the parameter identifier
description	A short (max 40 chars) description of the parameter
module	Defines the "type" of the parameter. See table below for allowed types.

The default value is specified within the tag context.

The intermediate xml documentation contains these parameters to enable the stylesheet to lookup these in order to generate dynamic content.

Available standard module attribute values allowed are:

"*line*" Simple one-line text edit component
"*color*" A standard color dialog to pick a color from ("000000") (not implemented, currently same effect as "*line*")
"*bool*" A true/false checkbox
"*text*" A memo to enter a couple of text lines
"*filename*" A standard file selection dialog with custom information:

```
<title>Browse for a textfile...</title>
<filename></filename>
<filter>Actionscript 2.0 Class (*.as)|*.as|All Files (*.*)|*.*</filter>
<defaultext>*.as</defaultext>
```

`<title>` Title of the file selection dialog
`<filename>` Default Filename
`<filter>` A list defining file types in the dialog separated in the form of

"*description|Filter[description|Filter[…]]*"

`<defaultext>` The default selected filter of those defined in <filter>

Example of parameter nodes:

```
<parameters>
  <parameter name="noframes" module="bool" description="Omit Frameset in
documentation">false</parameter>
  <parameter name="navbg1" module="line" description="color" />
  <parameter name="footer" module="text" description="Footer Text" />
  <parameter name="css" module="filename" description="Custom CSS File">
      <title>Browse for css file...</title>
      <filename/>
      <filter>Cascading Style Sheet (*.css)|*.css|All Files (*.*)|*.*</filter>
      <defaultext>*.css</defaultext>
  </parameter>
</parameters>
```

Example command-line call:

as2doc ... -s:footer "Generated on" -s:css c:\styles.css -s:noframes true -s:navbg1 009900

2) `<file>`

The tag defines a certain action AS2Doc will execute according to its attributes which vary depending on two cases described below.

A) Copy a file to the target documentation

```
<file source="as2doc.css" dest="as2doc.css" basedir="style"/>
```

This node instructs AS2Doc to copy a file or directory specified by the attribute "*source*" to the file or directory specified by the attribute "*dest*". The attribute "*basedir*" has the following possible options:

style	The source is relative to the style's root folder (location of "*style.xsl*")
source	The source is relative to the root classpath or directory of the documented class
output	The source is relative to the root of the target output directory

Wildcards are also supported. Example to copy any JPG images from a "yourstyle/resources/" directory to a target directory "*images*" in the output folder you could use:

```
<file source="resources/*.jpg" dest="images/" basedir="style"/>
```

B) Translate XML Documentation with given XSL Style to generate a file in the target output directory

```
<file style="index.xsl" dest="index.html" [type=”indexfile”]/>
```

This node instructs AS2Doc to use the XSL Stylesheet "*index.xsl*" in the root of the style directory (location of the "*style.xsl*" file) and translate it with the XML Documentation and store the resulting document in the file named by the attribute "*dest*" in the output directory. The optional attribute "*type*" with the value "*indexfile*" marks the generated file as the "*main index*" file for the generated documentation format. This file is automatically opened if a user clicks the View Button within the AS2Doc GUI or uses the command line "*-run*" parameter.\

AS2Doc processes the nodes in document order. You can customize the set of `<file>` nodes dynamically to your needs using XSL Transformations.

Additionally you can pass dynamic XSL Stylesheet Parameters to each stylesheet to be called using a `<param>` node.

The following example will supply an "*index.xsl*" stylesheet with two XSL Stylesheet Parameters

```
<file style=”index.xsl” dest=”index.html”>
      <param name=”classes”>true</param>
      <param name=”interfaces”>true</param>
</file>
```

You can alter the `<param>` tags using XSL transformations dynamically in order to change passed parameters.

To use the passed XSL Parameters, the stylesheet "*index.xsl*" has to have the following additions as top-level elements:

```
...
<xsl:param name=”classes”/>
<xsl:param name=”interfaces”/>
...
```

It would now be possible to access the contents of the classes parameter within an XPath expression using *$classes* or *$interfaces*. It could trigger if only classes, interfaces or both are output with our style.

Have a look at the standard HTML Output Style pipeline document for a working example in the "*styles/AS2Doc/HTML/styles.xsl*" file. It makes use of all the features explained.

Note: This has nothing in common with the <parameters> section within the <information> node.

6.1.2 A simple style tutorial

This simple tutorial should show you in an example how to create a style which generates a simple plain text list of all classes and interfaces documented.

1. Create a subdirectory in the "*styles*" folder: "*styles/examples/classlist/*"
2. Create a "*styles.xsl*" file in the above directory with the following contents:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
indent="no"
media-type="text/xml"
omit-xml-declaration="no"/>
<xsl:template match="/">
<generation>
      <information>
            <author>Your Name</author>
            <version>1.0.0</version>
            <appVersion>1.0.0</appVersion>
            <name>Classlist Textfile</name>
            <description>This style generates a simple text file which contains a
list of all classes and interfaces.</description>
      </information>
      <file style="classlist.xsl" dest="classlist.txt" type="indexfile"/>
</generation>
</xsl:template>
</xsl:stylesheet>
```

The "*style.xsl*" acts as the main pipeline file for our output. According to the reference in the preceding sections you can see that it contains basic information about our style and defines on file for generation using the <file> node.

The file it generates will be named "*classlist.txt*" and the attribute "*type*" indicates that it will be the index of our documentation output style thus be opened if the user clicks on the GUI View Button or uses the "*-run*" command line parameter along with our style.

The <file> node also specifies an attribute "*style*" and points to a XSL Stylesheet to use in order to generate the "classlist.txt" output file. This means that AS2Doc will take the XML Documentation and transform it using our "*classlist.xsl*" file and store the result in the "*classlist.txt*" file.

3. Within the directory of this style place a second file named "*classlist.xsl*" with the following contents:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"
encoding="ASCII"
media-type="text/plain"/>

<xsl:template match="/">
<xsl:for-each select="/documentation/package">
```

```
<xsl:for-each select="./classes/*[self::class|self::interface]">
<xsl:if test="./@ns!=''"><xsl:value-of select="./@ns"/>.</xsl:if><xsl:value-of
select="./@id"/><xsl:value-of select="'&#x0A;'"/>
</xsl:for-each>
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

The raw XML Documentation is being transformed with this stylesheet as we defined this operation within the "*style.xsl*" file's `<file>` node.

You see that the output method is "*text*", the specified encoding "*ASCII*" and the media-type "*text/plain*" as we intend to generate a plain text list of classes and interfaces.

The main template part contains `<xsl:for-each>` loops in order to process each class and interface within the XML Documentation. It then outputs a namespace "*./@ns*" with a dot "*.*" following if the namespace is not empty (would be the "root package" in that case so no "." required) and finally outputs the identifier of the class or interface and a "linefeed" character defined with "*
*".

4. The style is ready for use. Start AS2Doc now. If you have made any mistakes you should get an error report on the log page immediately, otherwise you can now find your style listed on the "*output*" page.

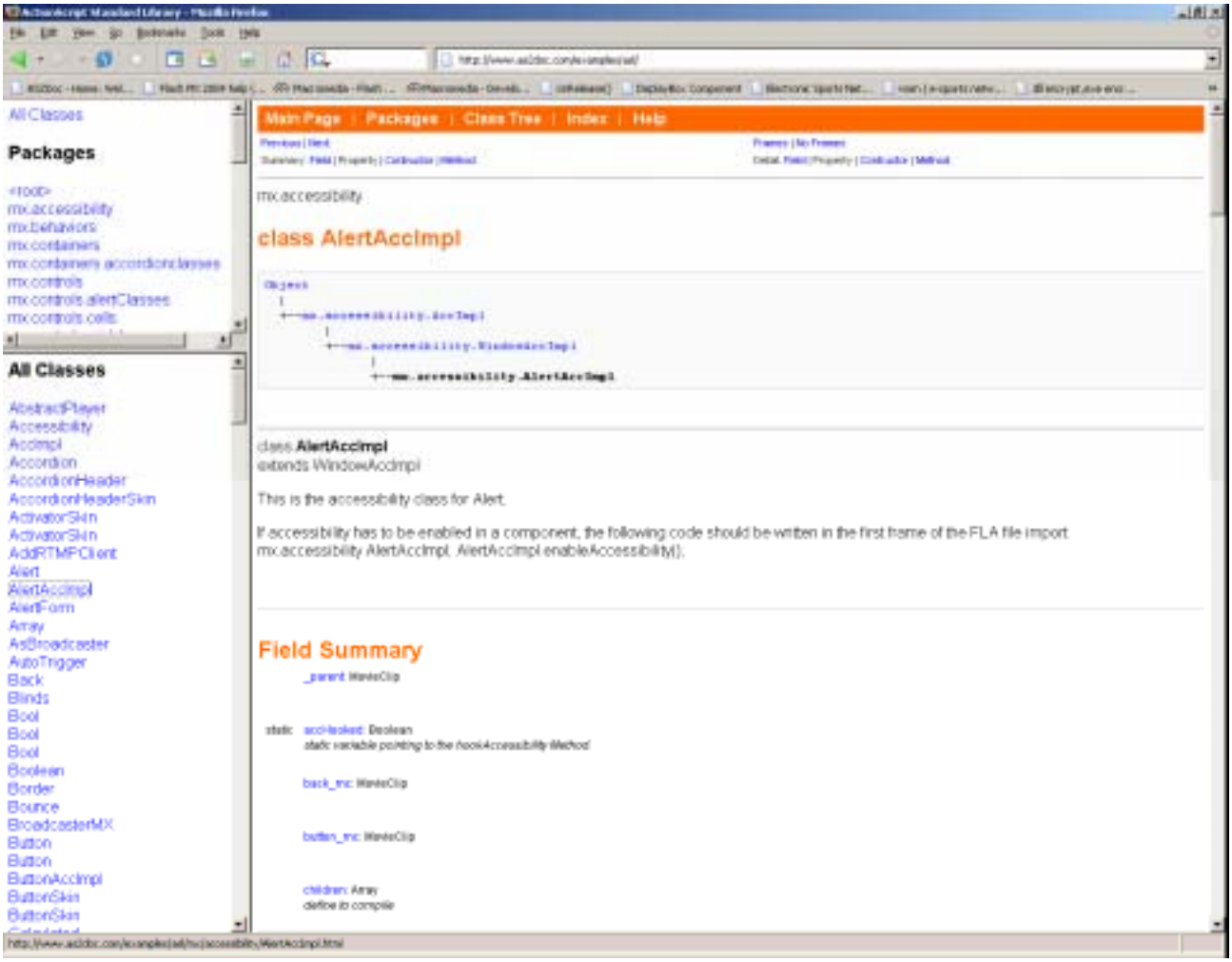
5. Select a source and a target output directory and test your style.

There are many possible scenarios you can solve and output formats you can create by creating custom styles.

See the bundled *default styles* for more examples.

6.2 Standard HTML Style

The standard HTML Style generates a documentation intended for Web-usage and contains a framed view based on the popular “JavaDoc ™” HTML documentation standard.



6.2.1 Introduction

If you use any files referenced within your doc comments put them in a subdirectory named “*doc_files*” in the package directory of the documented class.

Example image link:

```
/**
 * @description   A picture:<br>
 *                <img src=”doc_files/screenshot1.jpg” border=”0” alt=”Example”>
 */
```

If this comment would be used within a class “*de.mirell.m3d.CObject*”, you would have to put the picture into the “*de\mirell\m3d\CObject\doc_files*” directory within your source tree.

The output style automatically copies the contents of the “*doc_files*” directory thus leading to the picture appearing in the final HTML documentation correctly referenced.

This should work with any media files you use. (Flash, Images, further HTML, PDF etc.)

Generated File Structure:

index.html	Initial page that sets up HTML frames (if noframes is true, title page)
packages-summary.html	Lists all packages
packages-tree.html	Lists class hierarchy for all packages
packages-frame.html	Lists all packages, used in upper-left frame
allclasses-frame.html	Lists all classes for all packages, used in lower-left frame
help-doc.html	Lists user help for how these pages are organized
index-all.html	Default index for the documentation
as2doc.css	CSS Stylesheet for pages (overwritten with file from css parameter)
as2doc.png	The AS2Doc logo for the footer (overridden by footer parameter)
some	
package	
AClass.html	Page for AClass class
AIntf.html	Page for AIntf interface
package-summary.html	Lists classes with short description summaries for this package
package-frame.html	Lists classes in this package, used in lower left-hand frame
package-tree.html	Lists class hierarchy for this package
doc-files	Directory holding image and example files

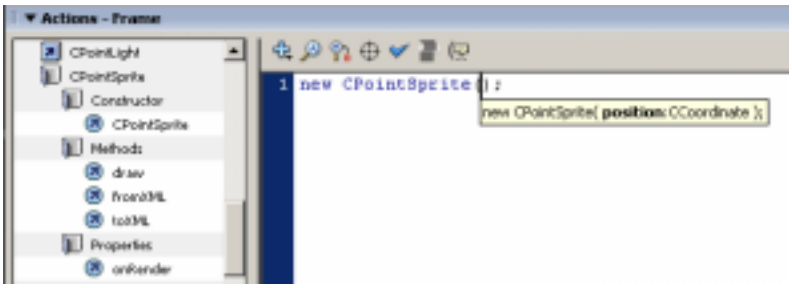
6.2.2 Configuration

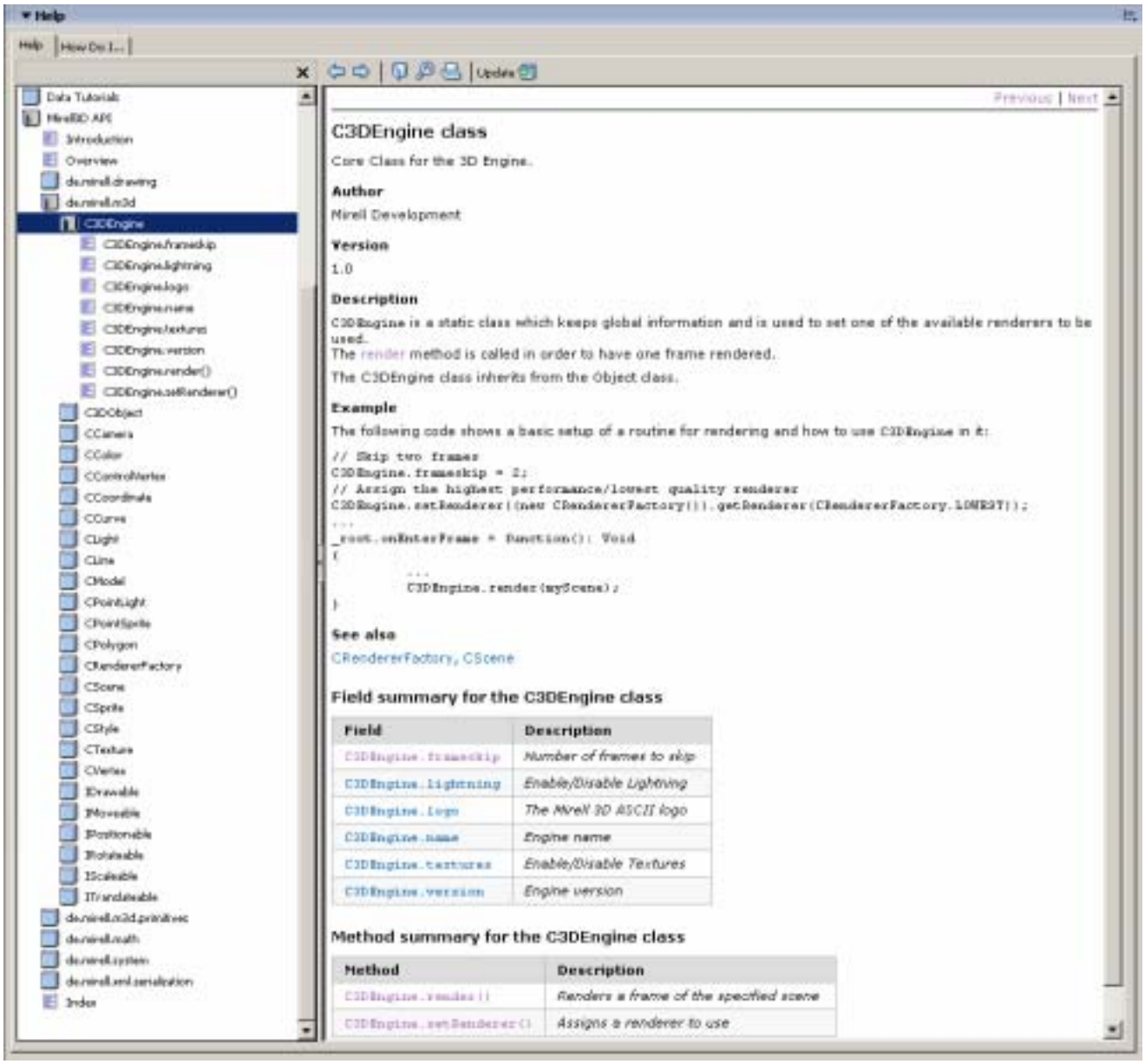
The Output Style offers the following options for configuration which can be accessed either using the GUI style configuration dialog or the *–s:<name> <parameter>* command line parameters:

noframes	Omits generation of the "*-frame.html" files and removes frame navigation overall
notree	Omits all class hierarchy pages in the final documentation
nohelp	Omits generation of the help page
noindex	Omits generation of the index page
nonav	Omits generation of the top navigation
footer	Define your custom footer to be displayed at the bottom of each page
css	The full filename to a valid CSS file to be used instead of the default one

6.3 Flash MX 2004 Style

The Flash MX 2004 Output Style generates Flash IDE standard documentation files assembling the existing look and feel of original documentation included in Flash as close as possible to integrate perfectly into the concept of Flash Help.





6.3.1 Introduction

Generated File Structure:

myproject.mxi	MXI Template - contains all files generated in `<files>` node
ActionsPanel	
CustomActions	
myproject.xml	CustomActions XML file named after your project title
HelpPanel	
Help	
help_search_index.xml	Search index for internal Flash IDE search
help_toc.xml	Table of Contents XML for the Flash IDE Help Navigation
help_index.html	Frameset showing an index
help_index_main.html	Bottom-frame index for the documentation
help_index_nav.html	Top-frame letter navigation for the index frameset
overview.html	Lists all packages
introduction.html	Contains the main title page with the project description
id#####.html	Random filenames representing class and member pages

6.3.2 Configuration

The Output Style offers the following options for configuration which can be accessed either using the GUI style configuration dialog or the *–s:<name> <parameter>* command line parameters:

nomxi	Omit generation of the .mxi template file
nocustomactions	Omit generation of the CustomActions file
nohelp	Omit generation of the HelpPanel files
nosource	Omit automated addition of file tags within the .mxi template for classes
helpfooter	Define a custom footer for the HelpPanel Help
mxi.title	The title to appear in the .mxi template (Omit for project title to be added)
mxi.version	The mxi release version
mxi.type	The mxi type, default is “Flash Component” (Check MXI Reference)
mxi.author	The author of the mxi package
mxi.description	Description of the mxi package (Omit for project description to be added)
mxi.ui-access	Description of how to access contents of the distribution

6.3.3 CustomActions

The target directory will contain an XML file in “*ActionsPanel\CustomActions*” which represents the CustomActions within the Flash IDE. The file is used to display a folder reference of all classes which can be clicked on to add constructs to the code editor, support for colored syntax highlighting, codehints to show parameter structure of methods and correct syntax within the IDE.

6.3.4 HelpPanel Documentation

HelpPanel Documentation is the newest addition in the IDE Help, it adds a folder browseable context-sensitive help partly based on HTML. It is the current standard of help generated for Flash based components and projects.

The output style automatically generates “*usage*” information for all elements without @usage tags.

6.3.5 MXI Template

Macromedia Products support the installation of extensions using a tool called Extension Manager. The distribution of extensions for Flash is made of MXP files which combined with the Extension Manager easily (un)install extensions.

Since these extensions can contain CustomActions, HelpPanel Documentation and ActionScript classes and more, a MXI description file (based on XML) is used by the Extension Manager to compile distributable MXP archives for users.

To ease adding the generated HelpPanel HTML files, which consist of random characters, to a MXI description for compiling a MXP archive, the style also generates the appropriate MXI file containing all files generated. You can also enable to generate appropriate XML nodes for the documented source code files. This would enable you to instantly compile a MXP distribution containing your ActionScript classes with full documentation for the Flash IDE and all its features!

Please mind that due to the specifications the MXI format specifies filenames to have a maximum length of 30 characters. We have seen classes with longer filenames, which still work within your Flash IDE but would not compile to a MXP archive and cause an error. The style therefore automatically uncomments appropriate `<file>` nodes if any files exceed 30 characters within your source file list.

You can find more information about the Extension Manager here:
http://www.macromedia.com/exchange/em_download/

The reference for the MXI format can be downloaded here:
http://www.macromedia.com/go/em_file_format/

6.4 RTF Style

The RTF style generates a document according to the Rich-Text-Format 1.8 Specifications of Microsoft® targeted for printing. It can further be used in editors able to read RTF files and customized. Various features are used including formatting styles which enable easy change of the look and feel of the document, dynamic page numbering and referencing, document hyperlinks, dynamic table of contents and more.

Classes and Interfaces are not just linked for browsing within the document but essentially provide page references for the reader of a printed version.

The RTF style uses features available in the Microsoft® Word Software like "*Fields*", once opened you will have to press *CTRL+A* (Select all) followed by *F9* (Update Fields) when the document has loaded completely.


de mirell.m3d

C3DEngine

Core Class for the 3D Engine.

```
class C3DEngine
extends Object
```

```
Object
  +--de.mirell.m3d.C3DEngine
```

Author:
Mirell Development

Version:
1.0

Description:
C3DEngine is a static class which keeps global information and is used to set one of the available renderers to be used.
The render (IUPDATE!) method is called in order to have one frame rendered.

Example:
The following code shows a basic setup of a routine for rendering and how to use C3DEngine in it:

```
// Skip two frames
C3DEngine.frameskip = 2;
// Assign the highest performance/lowest quality renderer
C3DEngine.setRenderer(new
CRendererFactory()).getRenderer(CRendererFactory.LOWEST);
...
_root.onEnterFrame = function(): Void
{
  ...
  C3DEngine.render(myScene);
}
```

6.4.1 Introduction

Generated File Structure:

myproject.rtf	The RTF document named after the project title

Internal File Structure:

Title Page	A title page containing the author, organization and project title
Table of Contents	A dynamicly created table of contents for the documentation
Introduction	A page with the project description
Package Overview	Lists all classes within a package

Class Page	Class detail page with member summary
Member Page	Member detail page

The last three pages are generated repeatedly depending on the amount of packages, classes and members of the class.

6.4.2 Configuration

The Output Style offers the following options for configuration which can be accessed either using the GUI style configuration dialog or the *–s:<name> <parameter>* command line parameters:

author	Author of the document (appears in the document summary and title page)
organization	Company/Organization (appears in the document summary and title page)
titletext	A custom text that appears on the title page
notitle	Omits generation of the title page
notoc	Omits generation of the table of contents pages
nointro	Omits generation of the introduction chapter containing the description
nopackage	Omits generation of package summary pages listing all classes within package
nosummary	Omits generation of member summary sections on class pages

7. Appendix

The appendix contains the Frequently Asked Questions (FAQ) and common troubleshooting information.

7.1 Frequently Asked Questions (FAQ)

What is AS2Doc?
AS2Doc is a documentation generator. It reads your ActionScript 2 based sourcecode and analyzes your comments and code. Information gathered from that process is then used to output a detailed and powerful documentation for your code.

Can I use AS2Doc with parameters from the command-line?
Yes. You can even supply a saved project information file so there is no need to add more options. Open AS2Doc once to open the user interface, configure your project settings, save them and use: "as2doc.exe myproject.xml to let AS2Doc generate the documentation on your settings automaticly.

Can I create my own documentation styles?
Simply, yes. Alongside with the standard styles we also provide you with a "template XSL style" which you can use to build your own style.
Look inside the subdirectory "styles" in your AS2Doc installation directory. The file to start with is called "styles.xsl" and is well-documented.

Also check out our Styles Section at http://www.as2doc.com/styles/ for new styles to download!

Why can I only select one class file (no classpath)?
This is a limitation of the trial edition. The full version enables you to select a classpath or single class to be used for generation. Using the GUI, you can configure which classes should be generated in the final output.

When I open the RTF Documentation "!UPDATE!" appears all over the document?
Due to a limitation of the RTF Format, fields are not automatically updated when the document is opened. In order to update fields in Word, use CTRL+A (Select all) followed by F9 (Update Fields).

Does AS2Doc support Chinese or Japanese characters?
Yes, AS2Doc supports all characters available for the input encoding UTF-8 as used for Flash ActionScript sources. The output styles have special processing built-in to enable encoding support within the output formats.

The class comments are missing from the generated docs, but are in the source code. I only get the doc comments for the methods, fields and constructors.
If the class-level doc comments are missing, a common mistake is putting the import statement between the doc comment and the class definition in the source file. There must not be any statements between the class doc comment and the class declaration.

```
/**
 * This is the class comment for the class Whatever.
 */

import de.mirell.*;   // MISTAKE - Important not to put statements here

public class Whatever {
}
```

Where can I find the latest version of this documentation?
On the AS2Doc website at http://www.as2doc.com in the support section.

7.2 Program Warnings

"Output directory does not exists. Attempting to create."
If the target directory or any subdirectories you specified do not exist, AS2Doc attempts to create these.

"No styles found in /styles/"
On startup, AS2Doc seeks for any styles within the "styles" subdirectory. Make sure you have the as2doc.exe within the correct installation folder and your bundled styles in the folder of the executable.

"No class/interface definition found in FILE.as!"
While searching for classes, AS2Doc recursively looks for .as files within each directory. It will try to open and parse a valid ActionScript 2.0 class from the file. Since .as files also contain ActionScript 1.0 source code or other data, it reports this warning. If the file reported IS a valid ActionScript 2.0 file you should check it for any syntactical errors.

"No <information> tag found in XSL Stylesheet."
Each output style AS2Doc recognizes upon startup has to contain an <information> tag. Missing tag leads to not displaying the style within the GUI. If this error is reported for a custom style you created, make sure to check your "style.xsl" file for problems.

"Skipping unknown parameter module: NAME"
This warning appear if your custom style has specified a <parameter> node with a "module" attribute value that is not one of the list. (*See 6.1.1 Style Reference*)

"A <file/> pipeline node missing instruction attributes (style or source) was encountered."
A valid <file> node has to either have a "style" or "source" attribute as defined by the reference within your "style.xsl" file.
(*See 6.1.1 Style Reference*)

"A stylesheet <param/> tag was missing the "name" attribute."
A <file> node within the "style.xsl" pipeline file of an output style contains a <param> tag without the "name" attribute.
(*See 6.1.1 Style Reference*)

"Style not found at FILENAME"
Displayed if you supply a custom style filename for generation which is not found by AS2Doc. The default HTML style is used instead.

7.4 Program Error Messages

"No class/interface was selected for generation."
You should check if you have selected any classes on the "*source*" page. You select elements using the checkbox or the right-click popup menu.

"Error loading XML Project document."

The error occurs if loading of a project file fails. This can have various reasons, most likely due to manually edited XML mistakes. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error loading XSL Stylesheet document."
A problem occurred during the loading of an XSL Stylesheet. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error loading XSL Stylesheet Information from project file."
A problem occurred during the reading the `<information>` node of an output style "*style.xsl*" file. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error loading source XML document. * Contact Mirell Development ***"**
If this error occurs AS2Doc has generated XML documentation which is not valid XML.
This error can only appear in special cases and is mostly caused by wrong/irregular doc commenting. Please contact support@as2doc.com and send us a saved copy of your project's XML file or the raw XML Documentation.

"Error loading XSL Generation Pipeline Stylesheet document."
AS2Doc could not read the "style.xsl" file due to errors in it. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error transforming XSL Generation Pipeline."
AS2Doc could not generate the initial generation pipeline using the source XML Documentation and an output style's "*style.xsl*" file. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error in generated Pipeline XML document."
AS2Doc encountered a problem within the XML file resulting from the transformation of the XML Documentation and an output style's "*style.xsl*" file. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error in included XSL Stylesheet."
A problem was located within a XSL Stylesheet provided by the current output style. The error should contain further information to solve and locate the problem and most likely line and column number.

"Error transforming XSL."
This error is displayed if AS2Doc is unable to transform a result document using a style specified within the attribute tag "*style*" of a `<file>` node. The error should contain further information to solve and locate the problem and most likely line and column number.

8. Annotations