as2api Documentation

Table of Contents

About

as2api parses ActionScript 2 source code and generates HTML API documentation in the style of JavaDoc. It is open source, and runs on Windows, MacOSX and Linux.

Getting the software

There are three versions of the software available from the project homepage, http://www.badgers-in-foil.co.uk/projects/as2api/:

- A single-file, command-line executable for MacOSX
- A single-file, command-line exe for Windows
- The source ruby scripts, runnable from the command-line under Linux (or anywhere else that Ruby is available)

Command-line Reference

Name

as2api -- ActionScript 2 API reference documentation generator

`as2api` [--help] [--output-dir *`path`*] [--classpath *`path`*] [--title *`title-text`*] [--progress] [--encoding *`encoding-name`*] [--draw-diagrams] [--dot-exe *`executable-path-and-name`*] [--sources] *`package-spec`*...

Options

Arguments can be given in any order. *`package-spec`* is required.

`package-spec`

You must specify at least one package of types to be documented, and may specify multiple packages:

```
as2api org.example.utils com.yoyodyne.app
```

If you want to document all packages who's names share a common prefix, you can avoid listing each package individually by giving the package prefix with the suffix '.*':

```
as2api com.yoyodyne.gui.*
```

`--help`, `-h`,

Causes the program to exit immidiately after showing some brief usage instructions.

`--output-dir` *`path`*, `-d` *`path`*,

`path` specifies the location where the generated HTML files should be placed. If unspecified, the default location will be `apidocs`, within the current directory.

If *`path`* does not exist, it will be created.

`--classpath` *`path`*

The *`path`* is a list of directories in which as2api should search for source code. If no classpath is specified, as2api just searches within the current directory. Items within the list should be seperated by the platform-dependant path seperator:

Windows --classpath *`dir1`*;*`dir2`*

OSX, Unix, --classpath *`dir1`*:*`dir2`*

Note that if one of the directories in the classpath contains spaces, you will need to enclose the whole classpath in quotes:

```
--classpath "C:\Documents and Settings\dave\as;dir2"
```

`--title` *`title-text`*

Some text to be included in the titles of generated pages (e.g. the name of the software project / library).

`--progress`

Causes some extra progress information to be shown as the pro-

	gram runs
`--encoding` *`encoding-name`*	The encoding for the generated HTML files. Note that this must match the encoding of all input source files; no transcoding is performed. as2api cannot handle a mixture of file encodings in the set of source files to be proccessed.
`--draw-diagrams`	Enables the generation of inheritance diagrams on package overview pages. This requires the external **dot** tool from the graphviz package. If you have downloaded graphviz, but the tools are not available in the standard system PATH, you will need to specify the location of **dot** by adding `--dot-exe` to the command line.
`--dot-exe` *`executable-path-and-name`*	Explicitly names the **dot** executable to use, should this not be available in any of the standard locations on your system.
`--sources`	Enables the inclusion of a copy of the source code of each documented type in the generated HTML. The source will be converted to an HTML file with 'syntax highlighting'.

How to Write Documentation

Overview

ActionScript Comments

ActionScript 2 provides two kinds of *comment* marker, for denoting text which is not part of the executable program code. There is the C++ style single-line comment,

```
// everything on the line after the two slashes
// is ignored
```

and the C style multi-line comment,

```
/* text between start and end markers is
   ignored, and the comment may continue across
   multiple lines in the source code */
```

as2api Documentation Comments

as2api ignores the single-line comments, but it will read text from the multi-line comments if both the following conditions are true,

- The start-of-comment marker has two asterisks, not just one: "`/**`"
- The comment immediately precedes the definition of part of the public API of an ActionScript 2 class (e.g. just before the class definition itself)

Example 1. A Simple class definition with as2api documentation

```
/*
 * MyFirstClass.as
 *
 * Copyright (c) __MyEmployer__ 2005
 *
 * The contents of this comment are ignored
 */

import "otherpackage.MyOtherClass";

/**
 * Objects of this class are responsible for
 * serving as an example to others.
 */
class thispackage.MyFirstClass {

  /**
   * When called, this method puts into motion a
   * plan so cunning that words cannot describe it.
   */
  public function enactCunningPlan():Void {
      // TODO: implement cunning plan
  }
}
```

Writing Useful Documentation

Here are a few guidelines

Empathy!

Take a deep breath.

Take five (mental) paces back from your code.

Try to see the API from someone else's point of view. It will help if you imagine that this other person is lazy, but *not* stupid. They want to gain the maximum understanding with the minimum amount of reading.

Describe the Interface, not the Implementation

Classes are useful because they can hide the complicated details of how things are achieved behind a simple, black-box interface.

If part of a class's implementation can be changed, and no code making use of the class would notice, then this is an implementation detail, not part of the API which must be documented.

Avoid Repeating the Code in English

This kind of annotation is not useful:

```
/**
 * Set the name property
 * @param name the name to set
 */
public function setName(name:String):Void {
  // ...
}
```

Method and class names are an integral part of the API documentation, and should already convey a useful overview of their own purpose.

Syntax Overview

The javadoc-style syntax supported by as2api allows the contents of javadoc comments to contain special javadoc 'tags' to add extra meaning, and XHTML tags if extra formatting (e.g. tabulated information) is required.

If the first non-whitespace characters on a line within a comment are one or more asterisks (the '*' character), then everything upto and including the initial asterisks on the line will be stripped before it reaches the documentation. This allows you to use the common comment formatting idiom whereby the left-hand margin of the comment is marked with a column of asterisks.

Example 2. Initial Asterisks on Comment Lines

```
/**
 * This is an
 * example
 */
```

The documentation will include the text “This is an example”; the '*' characters before the words *This* and *example* will disappear.

Javadoc tags appear in two forms, *block* tags, such as `@throws`, and *inline* tags, such as `{@link}`.

Tags that are allowed in some contexts may not be allowed in others. For instance `@throws` is allowed to appear in the documentation of methods, but not in the documentation of classes. See the Syntax Reference below for the full details.

Block Tags

Documenation-comment is split into 'blocks'. The first block is always a general description of the API element to which the comment is attached. No special tag is required to denote this description block. Further blocks are added to the comment with block-tags, which must always appear at the beginning of a new line.

Example 3. Valid Block-Tag Usage

```
/**
 * So then this is the
 * description.
 * @throws Error in all cases
 */
```

Example 4. Invalid Block-Tag Usage

```
/**
 * So then this is the
```

```
 * description.  @throws Error in all cases
 */
```

Example 5. Block-Tag With Newlines

```
/**
 * So then this is the
 * description.
 * @throws Error in all cases
 *
 * text appearing here is still part of the 'throws'
 * block
 */
```

Inline Tags

Inline tags can appear within the middle of a line of comment text (unlike block-tags, which must always appear at the start of a line). Inline tags are always surrounded by curly braces (the '{' and '}' characters) so that as2api can tell them appart from 'normal' text.

XHTML Tags

Example 6. XHTML Markup in Comments

```
/**
 * The allowed values are:
 * <ul>
 *   <li>0.1</li>
 *   <li>0.01</li>
 *   <li>-6.66</li>
 * </ul>
 */
```

Syntax Reference

Supported JavaDoc Features

Feature	Description
Type Description	Describe a class or interface
Member Description	Describe a field/method of a class or interface
`@param`	Describe a method parameter
`@return`	Describe the value returned by a method
`@see`	Partially supported. Link to additional information.
`@throws`, `@exception`	Describe an exception thrown by a method
`{@link}`	Link to the documentation of another type or type-member

Feature	Description
{@code}	A code example (automatically escapes HTML special characters, like '&'

Unsupported JavaDoc Features

Feature	Commentary
@author	
{@docRoot}	
@depricated	
{@inheritDoc}	
{@linkPlain}	
{@literal}	
@serial	
@serialData	
@serialField	
@since	
{@value}	
@version	

Unsupported ActionScript Features

Feature	Commentary
#include "*file-name*"	includes are ignored
[*attributes*]	attributes on types/members are not documented

Class/Interface Description

A doc-comment immediately before a class or interface is taken to be a description of that class. The text here will be placed at the top of the page documenting the class's public fields and methods.

The first sentence of the class description will also be included in the package-level index of classes and interfaces. It should therefore try to give a brief overview of the class's purpose

Example 7. Class Description

```
/**
 * An immutable, type-safe wrapper around the
 * String value of the user identifier.
 * The constructor will raise an exception if
 * the given value is not of the correct format,
 * asserting that the GUI validation code has
 * done its job, and preventing invalid data
 * being sent to the backend.
 */
class thispackage.UserId {
```

```
    // ...
}
```

May contain. @see

Member Description

A doc-comment immediately before a method or field is used as the description of that member. The text here will be added to this members section in the type's API reference.

Example 8. Member Description

```
/**
 * Clears all data that has previously been
 * collected in this object, and returns it to
 * its initial state.  Subclasses are expected
 * to override this method to clear any state of
 * their own, and use super() ensure that data
 * defined by this class is cleared too.
 */
public function clear() {
    // ...
}
```

When implicit fields are created (using `function get foo()` or `function set foo()` methods), an entry will be created for this in the 'Fields' section of the resulting documentation (not in the 'Methods' section). If both `get` and `set` methods are defined for a field, as2api will 'pick' the doc-comment from one of them to use as documentation for the resulting field. If only one of `get` or `set` are defined, the documentation will note that the field is read-only or write-only, as appropriate.

May contain. @param, @return, @see

Method Parameters

Within the doc-comment describing a method, `@param` tags may be used to give a description for each of the method's parameters. Immidiately following the `@param` must be the name of the parameter to be described, with any text following this name being used as the description.

Arguments named in `@param` tags must match the names of the actual method arguments, or they will be skipped. The special parameter name '...' may be used if the method can accept a variable number of paramters (by inspecting it's `arguments` array.

Example 9. Method Parameters

```
/**
 * For each value contained by this object, invoke
 * the given callback function, passing the value
 * as an argument.
 *
 * @param callback callback function to be invoked
 *        for each object.  The function must accept
 *        one parameter (or two, if index is true).
 * @param index if true, each invocation of the
```

```
 *        callback function will be passed an
 *        additional second parameter giving the
 *        index of the current value (starting with
 *        0 for the first value, 1 for the second,
 *        and so on).
 * @param ... any additional arguments to be passed
 *        to the callback function.
 */
public function each(callback:Function,
                     index:Boolean):Void {
    // ...
}
```

Method Return Values

A description of the value returned by a method, where the method description itself does not give this information succinctly enough.

Example 10. Method Return Values

```
/**
 * Get the value corresponding to the given key
 * by searching in the local cache first, then
 * in the database on the LAN, and finally
 * consulting a number of LDAP servers from
 * around the wider internet.
 *
 * @return either the value for the given key, or
 *         null if there is no such mapping.
 */
public function get(key:String):String {
    // ...
}
```

Links to Related Information

TODO

Example 11. Class Related information

```
/**
 * See-also tags will be copied to the output HTML,
 * but are not actually turned into links to other
 * classes/methods as they should be.
 *
 * @see "some stuff"
 * @see some.OtherClass
 * @see some.OtherClass#field_name
 * @see some.OtherClass#method_name()
 */
class somepackage.ThisClass {
    // ...
}
```

Exceptions Thrown by a Method

TODO

Example 12. Exceptions Thrown by a Method

```
/**
 * Causes this message to be sent
 *
 * @throws MessagingException if there is a failure
 *         in the underlying communications medium
 */
public function send():Void {
    // ...
}
```

Linking to Other Parts of the API Documentation

TODO

Example 13. Linking to Other Parts of the API

```
/**
 * Causes this message to be sent to the
 * {@link OtherClass} instance, where it will be
 * dealt with by {@link OtherClass#recieve()}.  We
 * remember the destination in our {@link #reciever}
 * field.
 */
public function send():Void {
    // ...
}
```

Giving Code Examples

The code tag is provided to allow example ActionScript source code to be included in the documentation. The advantage of `{@code}` over simply using the HTML `<code>` tag is that you don't need to worry about 'escaping' characters (like '&', which are special in HTML) within the example code.

In addition, as2api will apply 'syntax highlighting' to the text within `{@code}`, which makes it look pretty. The disadvantage of this is that you can expect to see problems if you make certain classes of syntax error in the contents of the tag (e.g. failing to close a string literal).

Example 14. Example Code

```
/**
 * Generates an ending XML tag. For example, if
 * passed {@code "p"}, the result will be
 * {@code "</p>"}
 */
```

```
public function endTag(name:String):String {
    // ...
}
```

`{@code}` may span multiple lines, and when this happens, the resulting documentation will include an HTML `<pre>` around the code so that its formatting is preserved.

Larger ActionScript code examples may need to include braces (the '{' and '}' characters), however the end of the `{@code}` will be marked by the closing brace. Nested braces *are* allowed within `{@code}`, but each opening brace, *must* have a matching closing brace.

Example 15. Complex, multi-line code

```
/**
 * Returns the result of calling startTag(), the
 * given bodyGenerator function and endTag(),
 * concatenated together.  For example, one might
 * pass an anonymous function:
 * {@code
 * out.element("head", function() {
 *   return "some dynamicaly generated content";
 * });
 * }
 */
public function element(name:String,
                    bodyGenerator:Function):String {
    // ...
}
```