
as2api Documentation

Table of Contents

About	1
Getting the software	1
Installing and running the software	1
MacOSX	1
Windows	1
Command-line	1
How to Write Documentation Comments for as2api	2
Overview	2
Writing Useful Documentation	3
Syntax Reference	3

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 pre-compiled GUI app for MacOSX
- A pre-compiled GUI exe for Windows
- The source ruby scripts, runnable from the command-line under Linux (or anywhere else that Ruby is available)

Installing and running the software

MacOSX

TODO

Windows

TODO

Command-line

TODO

How to Write Documentation Comments for as2api

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 Reference

TODO

Supported Javadoc Features

Feature	Description
Type Description	Describe a class or interface
Method Description	Describe a 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	Describe an exception thrown by a method

Feature	Description
{@link}	Link to the documentation of another type or type-member
{@code}	A code example (automatically escapes HTML special characters, like '&')

Unsupported JavaDoc Features

Feature	Commentary
@author	
{@docRoot}	
@depricated	
@exception	@throws is supported, but this synonym is not, yet
{@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 2. 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

Method Description

TODO

Example 3. Method 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() {  
    // ...  
}
```

May contain. @param, @return, @see

Method Parameters

TODO

Example 4. 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).  
 */  
public function each(callback:Function, index:Boolean):Void {  
    // ...  
}
```

Method Return Values

TODO

Example 5. Method Return Values

```
/**
 * Get the value corresponding to the given key
 *
 * @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 6. 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 7. 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 8. 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

TODO

Example 9. 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 {
    // ...
}
```