tekom EUROPE

Anne Tarnoruder

Standards and Guidelines for API Documentation

For Technical Writers, Software Developers, Information and Software Architects

Practical Guides

Bibliographic Information of the Deutsche Nationalbibliothek (The German Library)

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Publisher
tcworld GmbH, Rotebühlstraße 64, 70178 Stuttgart, Germany
Phone +49 711 65704-0, Fax +49 711 65704-99
E-mail info@tekom.de, www.tekom.de

ISBN 978-3-944449-82-1 Print
ISBN 978-3-944449-83-8 eBook PDF

Layout: Elisabeth Gräfe, www.fraugraefe.de

Contents

Preface 5

1 Introduction 6

2 Terms and Concepts 7

3 API Documentation Processes 9

3.1 API Naming Guidelines 10

3.2 Common API Naming Mistakes 12

3.3 API Reference Quality Checklist 12

4 Java, JavaScript and MS.NET API Reference Documentation 13

4.1 Documentation Comments 13
4.1.1 Description 14
4.1.2 Tags 16

4.2 Documentation Tags 17
4.2.1 Java and JavaScript Tags 17
4.2.1.1 @deprecated Tag 17
4.2.1.2 @example Tag 18
4.2.1.3 @link Tag 18
4.2.1.4 @namespace Tag 21
4.2.1.5 @param Tag 22
4.2.1.6 @return Tag 23
4.2.1.7 @see Tag 24
4.2.1.8 @since Tag 26
4.2.1.9 @throws Tag 26
4.2.1.10 @version Tag 27
4.2.1.11 Additional JavaScript Tags 27
4.2.1.12 Custom or Legacy Java Tags 28
4.2.2 Inline Java Tags 29
4.2.3 C and C++ Tags 30
4.2.3.1 \file Tag 30
4.2.3.2 \mainpage Tag 30
4.2.4 .NET Tags 31
4.2.4.1 <example> Tag 31
4.2.4.2 <exception> Tag 32
4.2.4.3 <include> Tag 33
4.2.4.4 <param> Tag 34
4.2.4.5 <remarks> Tag 34
4.2.4.6 <returns> Tag 35
4.2.4.7 <see> Tag 35
4.2.4.8 <summary> Tag 36
4.2.5 HTML Tags 36

4.3 Java API Documentation Templates 37
4.3.1 Overview Page 37
4.3.2 Package Pages 39
4.3.3 Interface and Class Template 40
4.3.4 Method Template 42
4.3.5 Enum Template 44
4.3.6 Constant Template 45

5 REST and OData API Reference Documentation 47

5.1 Auto-Generated REST and OData API Reference Documentation 49
5.1.1 Guidelines for Documenting APIs with the OpenAPI Specification 49
5.1.1.1 General Guidelines for Descriptions 49
5.1.1.2 Overview 50
5.1.1.3 Operations 51
5.1.1.4 Parameters 51
5.1.1.5 Responses 52
5.1.1.6 Definitions 53
5.1.1.7 Security 55
5.1.1.8 Tags 56
5.1.1.9 External Documentation 57

5.2 Manually Written REST and OData API Reference 57
5.2.1 Using REST API Templates 57
5.2.1.1 REST API Overview Template 58
5.2.1.2 REST API Method Template 59
5.2.2 Using OData API Templates 60
5.2.2.1 OData Service Overview Template 60
5.2.2.2 OData Resource Template 61
5.2.2.3 OData Operation Template 62

6 Writing Developer Guides 64

6.1 Integration of API Reference Documentation 64

6.2 Writing Guidelines 64

7 External Resources 66

Preface

In the rapidly expanding API economy, software vendors are expanding their offerings of development platforms, tools and APIs. Professional API documentation is a key facilitating factor in the adoption of these offerings. Various companies and tool vendors define and maintain their own rules and best practices for documenting APIs, but so far there is no comprehensive widely adopted industry standard in this area.

Furthermore, API documentation tends to fall between the cracks. It is often written by developers who don't have enough resources and professional writing skills, which leads to the lower quality of documentation. On the other hand, professional technical writers do not always have the special knowledge and skills required to handle these topics.

To address these gaps, a group of API documentation experts at SAP, led by Anne Tarnoruder, had been formed in 2014 to produce a company-wide set of standards, guidelines and best practices. These Standards and Guidelines (S&G) aim to reach a higher level of quality and usability of the APIs, and thus increase the customer satisfaction and acceptance of APIs.

The S&G have since been used across SAP as a source of guidance and education for both writers and developers who produce API documentation.

The S&G cover both auto-generated and manually written API reference documentation, and apply to the major API languages and technologies, such as Java, JavaScript, MS.Net, REST and OData. These guidelines, based on the widely used industry standards for these languages and technologies, are more of a generic nature than specific to SAP and can be applied in any company.

Credits:

To Frederic Moitel and Ray Lefuel, the key contributors to the Java, JavaScript and MS.NET API Reference Documentation chapters.

To Michelle Kemp, Jack Schueler, Malca Sagal, Raylene Mehl, Nicole Goldman, Daniel Wroblewski, Vadim Tomnikov, Trevor Holdsworth, Steffen Lutter, Frederic Rousseau, G S, Sreedhara, and all other SAP colleagues who contributed by reviewing the materials, providing their feedback and support.

1 Introduction

The purpose of standards and guidelines (S&G) for API documentation is to help achieve consistency of content and style across various types and areas of the API documentation.

API Documentation Deliverables

API documentation deliverables fall into two complementary categories: API reference documentation and developer guides.

Documentation Deliverable	Description
API reference	Contains detailed reference information about all elements of APIs. Created and maintained by developers in software source code, reviewed by technical writers. Written in structured mode. Auto-generated from source code and integrated with the relevant developer guide or delivered separately. If auto-generation option is not available or not applicable, written manually by technical writers in the documentation system as part of developer guides.
Developer guide	Explains how to use the APIs. Contains concepts, diagrams, setup information, tutorials, tasks, code samples, and more. Created and maintained by technical writers in cooperation with developers in the documentation system. Written in freestyle mode. Delivered as part of the product documentation.

Scope and Target Audience

Standards and guidelines defined in this document:

- Apply mostly to API reference documentation and define the writing style and formatting rules for this documentation.
- Encompass major development languages and technologies, such as Java, JavaScript, Microsoft.NET, C/C++, REST and OData.
- Are targeted at technical writers and software developers who co-author API reference documentation either in source code or in the documentation system.

The following topics are out of scope of this document:

- API design principles and guidelines.
- Actual auto-generation and production of API documentation.

4 Java, JavaScript and MS.NET API Reference Documentation

Standards and guidelines discussed in this chapter apply to API reference documentation that is auto-generated from Java, JavaScript and Microsoft.NET source code.

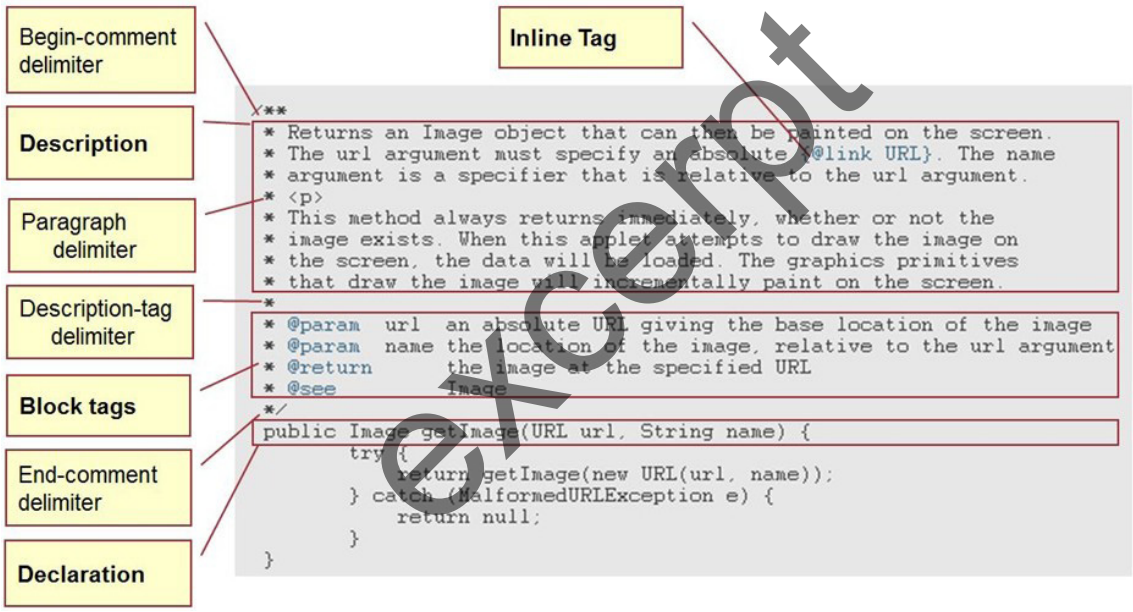
4.1 Documentation Comments

API reference documentation is generated from the documentation comments that are written in the API source code according to certain rules.

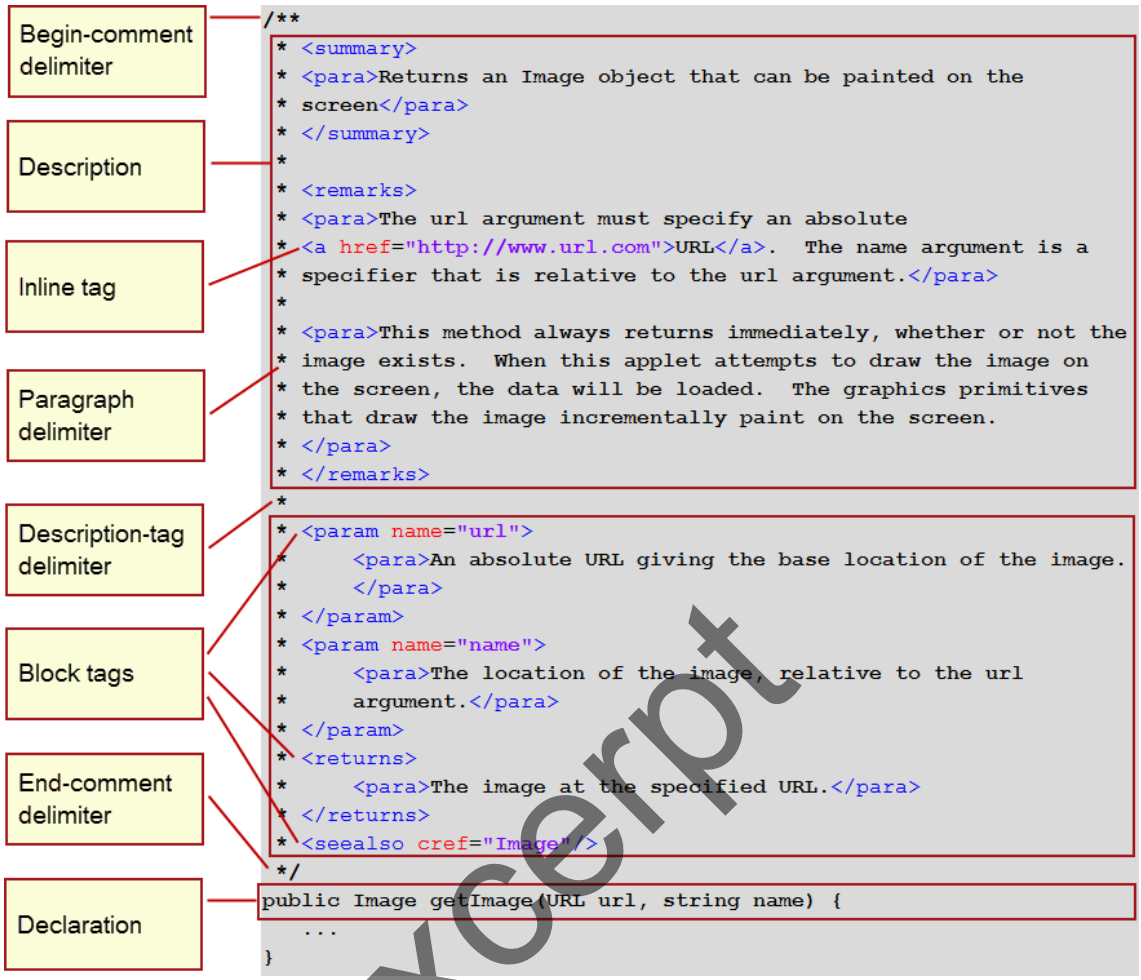
A documentation comment should precede the declaration statement of a namespace, class, interface, or class or interface element. A comment is made up of two parts: description and block tags, separated by delimiters.

The following figures show the structure and syntax of a documentation comment.

Java, JavaScript



.NET



For .NET APIs, it is possible to place documentation comments in an external XML file and then use the `<include>` tag to reference that file in the source code.

4.1.1 Description

Description is the first and mandatory part of a documentation comment for a class, interface, or class or interface element.

A description is usually made up of two parts:

- A mandatory summary sentence containing a short and exact description of the declared member.
- An optional detailed description that provides additional information about this element.

Guidelines

- In the summary sentence, omit clauses like "This class" or "This method". For an element that represents an action, start directly with a verb in the third-person form: adds, allocates, constructs, converts, deallocates, destroys, gets, provides, reads, removes, represents, returns, sets, saves and so on. For example:
  - › `Adds a new customer`
  - › `Provides read and write access to employee data`
  - › `Retrieves a Role object`

- For an element that represents an object rather than an action, use a noun phrase. For example:
  - `Base class for navigation`
  - `Alias of a backend system`
- Write the detailed description only to provide additional information that does not repeat the self-explanatory API name or the summary sentence.
- Avoid implementation details and dependencies unless they are important for usage.
- To avoid line wrapping, make sure each line of the description has fewer than 80 characters.
- In the output, the line breaks in a description are ignored, and it appears as a continuous text. To format descriptions, use HTML tags.
- To offset language keywords, API names, and code examples in a description, use the `<code>` tag.

Syntax

Java and JavaScript

Only the summary sentence, terminated by the first period, appears in the summary section of a generated reference. Everything after the first period is cut off, so make sure that the summary sentence can stand on its own.

Java

```
/**
 * This is the summary sentence.
 * <p>
 * This is the detailed description. Note that you can have multiple
 * sentences in the detailed description.
 * </p>
 **/
```

.NET

The summary sentence and detailed description are enclosed by the dedicated tags, `<summary>` and `<remarks>`. To format descriptions, use the `<para>` tag.

Code Syntax

.NET

```
<summary>
   <para>This is the summary sentence.</para>
</summary>
<remarks>
   <para>This is the detailed description.</para>
   <para>Note that you can have multiple sentences in the detailed
description.</para>
</remarks>
```

Related Information

4.2.1 Java and JavaScript Tags [page 17]
4.2.4 .NET Tags [page 31]
4.2.5 HTML Tags [page 36]
7 External Resources [page 66]

4.1.2 Tags

Documentation comments include special tags that are used by generation tools for generating structured API reference documentation.

Generation tools support tags of the following types:

- Block tags that can be placed only in the tag section below the description
- Inline tags that can be placed anywhere in documentation comments
- HTML tags that can be used for formatting of documentation comments

Syntax

Java and JavaScript

- Block tags are separated from the description by an empty line
- Block and inline tags have the following syntax: `@tagname comment`
- Inline tags are denoted by curly brackets: `{@tagname comment}`
- HTML tags have the standard HTML syntax: `<tag></tag>`

The following example shows the use of block and inline tags in a Java/JavaScript documentation comment:

```
/** Returns an Image object that can be painted on the screen.
 * The url parameter must specify an absolute {@link URL}.
 *
 * @param url An absolute URL of the image
 * @returns The image at the specified URL
 * @see Image
 */
public Image getImage(URL url);
```

.NET

All parts of a documentation comment, including descriptions, are enclosed in XML tags: `<tag name>comment</tag name>`.

The following example shows the use of XML tags in a .NET documentation comment:

```
/// <summary>
///  <para>Returns an Image object that can be painted on the screen.</para>
/// </summary>
/// <remarks>
///  <para>The url parameter must specify an absolute <a href="http://www.
url.com">URL</a>.
///  </remarks>
/// <param name="url">An absolute URL of the image</param>
/// <returns>
///  <para>The image at the specified URL</para>
/// </returns>
/// <seealso cref="Image"/>
public Image getImage(URL url);
```

Guidelines

- In a documentation comment, certain tags, such as `<param>`, can appear multiple times, while others, such as `<return>`, can appear only once.
- Certain tags are mandatory, while others are optional.
- Group multiple block tags of the same type together.
- Keep a consistent order of tags across all your API reference. For example:

```
param
return
```

```
throws
see
since
deprecated
```

- Tag comments are case-sensitive.
- Begin a block tag comment with upper case.
- Do not end a tag comment with a period if it is only one sentence or phrase.
- Do not repeat a tag comment that is generated automatically. Always try to add value to your comment.

Related Information

4.2.1 Java and JavaScript Tags [page 17]
4.2.4 .NET Tags [page 31]
4.2.1.12 Custom or Legacy Java Tags [page 28]
4.2.5 HTML Tags [page 36]

4.2 Documentation Tags

Reference of the documentation tags that are supported by the common API documentation generators.

4.2.1 Java and JavaScript Tags

A reference of the Java and JavaScript block and inline tags that can be used in documentation comments. The recommended order of tags inside a method comment block is as follows:

```
/**
 * @param
 * @return
 * @throws
 * @see
 * @since
 * @deprecated
 * @version
 */
```


Note
This section lists the most commonly used tags. For a complete reference of the available Java and JavaScript tags, see 7 External Resources [page 66].

Related Information

4.2.5 HTML Tags [page 36]
7 External Resources [page 66]

4.2.1.1 @deprecated Tag

Description

A mandatory block tag indicating that the class, interface, or class or interface element is deprecated.

Applies to: Java, JavaScript.

Syntax

```
@deprecated As of version NN, replaced by {@link class_name}
```

Writing Guidelines

Do not leave the tag comment empty. Provide the following information:

- "As of" followed by the API version in which the class, interface or class or interface element was deprecated.
- "replaced by" followed by a link (`@link` tag) to the replacing class or element.
- Optionally, provide additional information about the deprecation, but avoid mentioning any specific time frames for phaseout.
- Start the comment with a capital letter.

Example

```
/**
 * Returns the SELECT clause of a SQL query.
 *
 * @return A <code>String</code> that contains the SELECT clause
 * @deprecated As of version 2.4, replaced by {@link NewClass#getSelect()}
 * <p>{@link #getNewClass()} allows you to retrieve a {@link NewClass} ob-
ject.</p>
 */
String getSelect();
```

4.2.1.2 @example Tag

Description

An optional JavaScript block tag that inserts an example into a comment block.

Applies to: JavaScript.

Syntax

```
@example title
...
```

Writing Guidelines

- Optionally, add a title to the example.
- Add empty lines and indent code lines to make the comment easy to read.
- This tag can be used multiple times in the same comment block.

4.2.1.3 @link Tag

Description

An optional inline tag that inserts a hyperlink pointing to the documentation of the specified package, class, interface, element, or to external packages, such as Java Platform SE APIs, or to any URL.

Examples

With a title

```
/**
 * ...
 * @example To create a client
 * var client = new $.net.http.Client();
 *
 */
```

Without a title

```
/**
 * ...
 * @example
 * // creates client
 * var client = new $.net.http.Client();
 *
 * // where and what to send
 * var dest = $.net.http.readDestination("testApp", "myDestination");
 * var request = new $.net.http.Request($.net.http.GET, "/"); // new
Request(METHOD, PATH)
 *
 */
```

With indentations

```
/**
 * ...
 * @example
 * var id = myjob.schedules.add({
 *   description: "Added at runtime, run every 10 minutes",
 *   xscron: "* * * * * *\/10 0",
 *   parameter: {
 *     a: "c"
 *   }
 * });
```

Can be used in any part of a comment block and in any tag.

Applies to: Java, JavaScript.

Syntax

Java

```
{@link #constant label}
{@link #constructor(Type, Type,...) label}
{@link #constructor(Type id, Type id,...) label}
{@link #method(Type, Type,...) label}
{@link #method(Type id, Type id,...) label}
{@link class label}
{@link class#constant label}
{@link class#constructor(Type, Type,...) label}
{@link class#constructor(Type id, Type id,...) label}
{@link class#method(Type, Type,...) label}
{@link class#method(Type id, Type id,...) label}
{@link package.class label}
{@link package.class#constant label}
{@link package.class#Constructor(Type, Type,...) label}
{@link package.class#Constructor(Type id, Type id,...) label}
{@link package.class#method(Type, Type,...) label}
{@link package.class#method(Type id, Type id,...) label}
{@link package label}
```

JavaScript

```
{@link pathOrURL}
[label]{@link pathOrURL}
{@link pathOrURL|label}
{@link pathOrURL label}
```

Writing Guidelines

- Even though the number of `{@link}` tags allowed in a comment is unlimited, use this tag sparingly to keep the comment easy to read. We recommend using this tag where it adds value and only for the first occurrence of each API name in the doc comment.
- Method parameter types are optional.
- Optionally, specify a label to display in place of the URL.
- In Java, use the HTML tag `<a href="...">link</a>` to link to external web sites.
- In JavaScript, if you enter free text in a `@see` tag, add `@link` to create a hyperlink.

Examples

Java

```
 /**
* Returns a business item from its full path.
*
* @param rootFolder the root item folder, retrieved with
*   {@link com.sap.sl.sdk.authoring.businesslayer.
BusinessLayer#getRootFolder()}
* @param path the item full path
* @param failed if <code>true</code>, an exception is thrown (item not found)
* @return A <code>BlItem</code> object
* @see #getBlItemPath(BlItem)
* @since 14.1.5
*/
BlItem getBlItem(RootFolder rootFolder, String path, boolean failed);
```

JavaScript

```
/**
* Sets a string parameter used for CHAR, VARCHAR column types.
* ...
* This function does not convert data; to improve performance, it stores data
directly in the database.
* Note that special characters (in Unicode SMP or SIP) can cause the read op-
eration to fail.
* For more information see {@link http://en.wikipedia.org/wiki/
Plane_%28Unicode%29| Plane (Unicode)}.
* ...
* @param {integer} columnIndex The index of the parameter in the statement
starting from 1
* @param {string} value The string value to be set for this parameter
* @throws Throws an error on invalid parameters.
* @throws {SQLException}
*/
$.db.CallableStatement.prototype.setString = function(columnIndex, value){};
```

Related Information

4.2.1.7 @see Tag [page 24]
4.2.2 Inline Java Tags [page 29]
4.2.5 HTML Tags [page 36]

4.2.1.4 @namespace Tag

Description

A JavaScript block tag indicating that an object is being used as a namespace to keep a collection of classes or properties and methods under a single global name. Use this tag to describe the functionality provided by the namespace.

Applies to: Java, JavaScript.

Syntax

Syntax 1

```
/**
* @namespace description
*/
var S = { ...
};
```

Syntax 2

```
/** Description
* @namespace [{type}] [name]
*/
```

Writing Guidelines

- Use syntax 1 if the comment block is followed by code that defines the namespace directly.
- Syntax 2 is more suitable when the namespace is defined indirectly. The namespace type and name are optional. However, if the name is specified in the comment, it should match the name in the code, otherwise documentation generation results can be unpredictable.

Examples

Syntax 1

```
/**
 * @namespace Defines namespace for UI library
 */
var ui = {
    ...
};
```

Syntax 2

```
/**
 * Defines namespace for UI library
 * @namespace sap.ui
 */
jQuery.sap.define('sap.ui'); // defines the "sap" namespace indirectly if it
does not already exist
```

Related Information

7 External Resources [page 66]

4.2.1.5 @param Tag

Description

A block tag that adds a parameter entry to the Parameters section of a method or function description. Mandatory in method and constructor descriptions.

Add a tag for every parameter, even when the description is obvious.

Applies to: Java, JavaScript.

Syntax

Java

```
@param parameter-name parameter-description
```

JavaScript

```
@param parameter-name
@param {parameter-type} parameter-name
@param {parameter-type} parameter-name parameter-description
@param {parameter-type} parameter-name=default-value parameter-description
@param {parameter-type} [optional-parameter-name] parameter-description
@param {parameter-type1|parameter-type2} parameter-name parameter-description
```

Writing Guidelines

- Use a noun phrase to describe the value represented by the parameter.
- For Java, start the first word of description with lower case, with no hyphen before it. For JavaScript, start the first word with upper case.

> **Note**
> This is true for the common generation tools, such as Javadoc, JSDoc and Doxygen, and depends on the way this tag is rendered in the output. For example, in generated Javadocs the description follows the parameter name on the same line, separated by a hyphen. If your API documentation is generated with a different tool, check the output to see the tag's rendering.

- Do not end with a period.
- Describe the default values and/or specific possible values that have importance.
- To mark a parameter as optional, enclose the name in square brackets.
- To describe multiple types of a parameter, use the pipe character (|) as separator.
- To specify an arbitrary type, use an asterisk in curly brackets ({*}).
- For multiple parameters, include a separate tag for each parameter.
- The order of tags should match the order of parameters in the method signature

Examples

Java

```
/**
 * ...
 * @param term the search term
 * @param exact indicates whether to search for the exact term (default is
true)
 * @param sortBy the field by which to sort
 * ...
 */
public string search(String term, Boolean exact, string sortBy);
```

JavaScript with a simple type parameter

```
/**
 * ...
 * @param {string} statement The SQL statement to prepare
 * ...
 */
$.db.Connection.prototype.prepareStatement = function(statement){};
```

JavaScript with an object parameter

```
/**
 * ...
 * @param {Object} obj The report
 * @param {integer} obj.id The selected report's ID
 * @param {integer} obj.pageNumber The selected page number
 * ...
 */
WebiApplication.loadReport = function(obj) {
   parent.callImplementation(arguments, "loadReport");
};
```


4.2.1.6 @return Tag

Description

A mandatory block tag that adds a description of the return value of a method or function to the Returns section.

Applies to: Java, JavaScript.

Syntax

Java

```
@return description
```