Wireless Generation

Javadoc Code Commenting Standards for the Data Store ReST API

Draft 0.1

Created By: Julie Ross (US2484)

Last Modified: May 16, 2012

Revision History

The following table provides a change log of major revisions to this runbook.

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Name | Change Summary |
| 5/16/12 | 0.1 | J. Ross | Create doc |

Table of Contents

[Introduction 4](#_Toc324960797)

[Audience 4](#_Toc324960798)

[Roles and Responsibilities 4](#_Toc324960799)

[Scope 4](#_Toc324960800)

[What to Document in the REST API Java Source Code 5](#_Toc324960801)

[Resource/class description requirements 5](#_Toc324960802)

[How the REST “API DocGen Tool” processes resource descriptions 5](#_Toc324960803)

[How the Javadoc tool processes resource descriptions 5](#_Toc324960804)

[Some Simple Examples 7](#_Toc324960805)

[Example 1: Resource description doc comment 7](#_Toc324960806)

[**Example source code comment (incorrect)** 7](#_Toc324960807)

[**Example rendering of code comment (incorrect)** 7](#_Toc324960808)

[Example 1: What is wrong with this doc comment? 7](#_Toc324960809)

[**Example source code comment (rewritten)** 8](#_Toc324960810)

[QUESTIONS FROM JULIE 9](#_Toc324960811)

[General Structure of Java Doc Comments 10](#_Toc324960812)

[Main Description 10](#_Toc324960813)

[Tag Section 10](#_Toc324960814)

[First Sentence 10](#_Toc324960815)

[Writing Tips 11](#_Toc324960816)

# Introduction

This document provides a reference, guidelines, and examples that help programmers and technical writers produce API specification documentation from Java source code. Following these standards ensures that doc comments produce consistently-formatted API documentation, whether generated by Javadoc or by the internal reference-generation system.

The standards in this document have been agreed to by the Wolverine and Storm teams, but can easily be adapted to other teams’ standards, or revised as needed.

## Audience

The audience for this document includes users who directly modify Java API source code to create and/or edit doc comments.

This document assumes that readers are already familiar with the Javadoc tool, Javadoc-style doc comments, and HTML.

### Roles and Responsibilities

This document assumes the following roles and responsibilities of its readers:

* **Developers.** Responsible for writing API doc comments as part of the code-development process, and for keeping the comments consistent with changes to the API. Developers strive for “CAR”:
* Completeness
* Accuracy
* Relevance
* **Technical Communications Team writers.** Responsible for writing, technical editing, and copyediting the API doc comments to improve consistency, grammar, and adherence to standards.

All roles are responsible for generating, reviewing, and testing the generated output(s) after updates are made to API source comments.

## Scope

This document specifically addresses defect DE433, which is now US2484. See Rally, iteration 4.5, for more information.

This document describes what and how to document. This document does not describe how to generate the documentation.

# What to Document in the REST API Java Source Code

The Java source code for REST API resource classes must provide, at a minimum, the following types of content for each resource:

* A “complete” description of the resource (think “CAR”, as described previously)
* A description of each possible GET method for the resource.

The following subsections provide rationales for why the structure of Javadoc code comments for REST API resources is so important.

## Resource/class description requirements

We have dual requirements for the use of our Java code comments:

* For the purpose of the REST API auto-generated content, a one-paragraph description of the resource is required.
* For the purpose of the Java SDK, complete Javadoc documentation of the class is required.

To meet these requirements, developers must adhere to strict rules so that the description content can be single-sourced, or reused, for these dual purposes.

### How the “API DocGen Tool” processes resource and method descriptions for the REST API Reference

This is what you need to know about how the internally-developed tool chain (hereafter known as “API DocGen Tool”) generates the REST API Reference from Javadoc code comments:

* API DocGen Tool grabs all text from the opening /\*\* until the first @ tag in the header block comments. No matter how this text is formatted, the tool renders it as a single paragraph.
* The tool does not do Javadoc-type processing of the content. That is, if you include HTML elements such as <p /> to separate paragraphs (legal and recommended for formatting Javadoc-processed comments), those HTML elements will appear in the REST API reference “as-is”. So, please do not use HTML tags at this time.

Best Practice #1: Do not use HTML tags in the Main Description area of the class and method header block comments.

To learn how Javadoc doc comments are structured, see [General Structure of Java Doc Comments](#_General_Structure_of).

### How the Javadoc tool processes resource and method descriptions for Javadoc documentation

This is what you need to know about how the Javadoc tool parses Javadoc comments to extract the descriptions of Java classes, interfaces, methods, and so on:

* Javadoc grabs all text from the opening /\*\* until the end of the header block comment. Javadoc processes the code comments as HTML. Therefore, it is essential to use HTML tags so that the generated reference documentation formats as expected, such as white space between paragraphs and code examples.
* Javadoc interprets special tags that help developers document method parameters, return values, exceptions, and more, to produce a rich set of code documentation.

Best Practice #2: Although you could use the power of Javadoc and HTML to create sophisticated documentation, do not use it in Main Descriptions that must be processed by API DocGen Tool.

To learn how Javadoc doc comments are structured, see [General Structure of Java Doc Comments](#_General_Structure_of).

# Some Simple Examples

Now that you understand some of the constraints and requirements, here are some simple examples of code comments that work and code comments that do not work.

## Example 1: Resource description doc comment

The following examples are based on the courses resource of the REST API.

### Example source code comment (incorrect)

This example shows a resource/class doc comment that is not correctly formatted. This code comment will not render properly with either API DocGen Tool or Javadoc.

/\*\*

 \* CourseResource

 \*

 \* This educational entity represents the organization of subject matter

\* and related learning experiences provided for the instruction of students

\* on a regular or systematic basis.

 \*

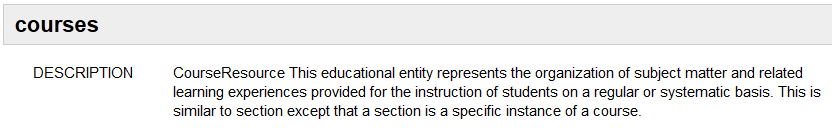
 \* This is similar to section except that a section is a specific instance

\* of a course.

 \*/

### Example rendering of code comment (incorrect)

The following figure shows how API DocGen Tool renders the preceding code comment.



### Example 1: What is wrong with this doc comment?

As you can see, all elements of the preceding code comment were combined into one large paragraph. Furthermore, it does not meet “CAR” standards.

To be more specific, consider the following elements of the code comment and how they affected the generated rendering:

* CourseResource – This is the actual Java class name, which is irrelevant in a REST API resource reference. Furthermore, including the Java class name in the doc comment is not consistent with Javadoc commenting standards, nor is it necessary in Javadocs documentation.

Solution: Do not include the name of the Java class in the main description.

* This educational entity … – This wording is incorrect style for the wording of a “first sentence”, as defined by Javadoc commenting standards.

Solution: Rewrite the first sentence according to the guidelines in this document. See [First Sentence](#_First_Sentence).

* The writing style could be improved to enhance clarity – (Note: this text was probably copied verbatim from the Ed-Fi documentation of entities.) Use generic terms. Shorten sentences and chunk information to improve readability.

Best Practice #3: Strive for “CAR”, that is, clarity, accuracy, and relevance. Tell users that they need to know, but present the information clearly. See [Writing Tips](#_Writing_Tips).

### Example source code comment (rewritten)

In the following code comment, API DocGen Tool and Javadoc would still combine all of the Main Description text into a single paragraph. However, this text meets the “CAR” requirements and code commenting standards.

/\*\*

 \* Represents the definition of a course. A course is an educational entity that defines

\* the organization of subject matter and related learning experiences. Courses are provided

 \* for the instruction of students on a regular or systematic basis. For detailed information,

\* see the schema for the Course entity.

 \*

 \* A section is an instance of a course that provides specific details such as when the

\* course was taught. For more information, see the sections resource.

 \*/

NOTE FROM JULIE: In the above example, I wanted to distinguish between a reference to an entity schema (“Course”) and a reference to a resource (“sections). I didn’t use the $$sections$$ convention because it’s kind of misleading due to the spelling—the resource name—but the link takes you to the entity schema, which really should appear as “Section”.

# QUESTIONS FROM JULIE

1. Which text comes from the Java source code? Current assumptions:

* DESCRIPTION – Comes from the Main Description of the resource’s Java class code comment. We only provide a DESCRIPTION for the base resource. We do not provide descriptions for the 2-, 3-, and 4-part URIs that are derived from the base resource.
* METHOD (GET) – Description of the method should always come from the method header block comment. However, many Java classes do not have method descriptions—just the name of the method, for example, most methods in the CourseResource class. And, it looks as if some Java classes use the @return tag of methods to document what should really be in the Main Description, for example, most methods in CourseResource.java.  
  One of my concern is that each class has a large number of getter methods. I don’t see how the developers could document all of these—in a meaningful way— in time for Alpha.  
  Solution A: Boilerplate the GETs text in the API DocGen Tool (no changes required in the Java source code—that is, the tool ignores the @return text ).  
  Solution B: Boilerplate the code comments in the Java source code. Although “vanilla” comments with no value added, this approach has the advantage of being faster to edit. Furthermore, Solution B establishes a placeholder for text that could be elaborated on later.  
  VERY IMPORTANT: If we choose Solution B, we should stop documenting this information in the @return tag. The information about what the method does (returns things) belongs in the Main Description of the getter method. The @return tag is supposed to document values and error codes, not the purpose of the method!!!
* METHOD (PUT, POST, DELETE) – Currently boilerplate text provided by the API DocGen Tool? If so, can we keep it that way? The reason is that few of these methods are documented properly. These methods, like the getters, are not using the Javadoc standard of describing the purpose of the method in the Main Description.

1. I think we need to distinguish between a reference to an entity schema (for example, “Course”) and a reference to a resource (for example, “sections”). The $$sections$$ convention is misleading due to the spelling—the resource name—but the link takes you to the entity schema, which really should appear as “Section”. Is there time to fix this?  
   Also, I don’t think it’s necessary to have these links appear in every method description, do you? Shouldn’t it be enough to only appear in the description of the main resource?

# General Structure of Java Doc Comments

The following table describes the structure and purpose of Javadoc code comments.

| Comment Area | Explanation |
| --- | --- |
| **Main Description** | Begins with the first sentence after /\*\* and ends with the first block tag, such as @param or @return.   * Tip: The Main Description can contain as many sentences as needed. |
| **Tag Section** | Begins with the first block tag, such as @param or @return.   * Tip: The Main Description must precede the Tag Section.   Required tags:   * @param * @return * @throws * @deprecated   Also recommended:   * @see * @since (use to highlight major changes in public-facing APIs) |
| First Sentence Required for:   * Interface * Class * Field * Method | An accurate, clear, complete, and concise description that defines what the entity is or does.   * Tip: Javadoc copies the first sentence to generated summaries and tables, as the following example shows.   The first sentence answers one or more of these questions:   * What is it? * What does it do? * What does it provide? * What does it contain?   Guidelines for writing a first sentence:   * Use a phrase, not a complete sentence. * Do not include the name of the class in the first sentence. * Capitalize the first word. * End the phrase with a period (the period tells Javadoc where the first sentence ends). * Provide more information than what is obvious by the name of the entity. |

# Writing Tips

The following table provides writing practices that enhance clarity, consistency, and readability.

| Writing Tip | Explanation | |
| --- | --- | --- |
| Chunk information into paragraphs | For the purposes of this document, you can separate content into paragraphs, but they will render as a single paragraph.  At this time, do not use HTML tags to break text into paragraphs. | |
| Strive for clarity and brevity | Guidelines:   * Use phrases instead of complete sentences, especially for the first sentence of the Main Description. * For example, don’t use “entity” when describing a “resource” or “resource representation”. * Begin a method description with a verb, for example: * (GET – collection) Returns the requested collection of resource representations. * (GET – one or more UUIDs) Returns the specified resource representation(s). * (POST) Creates a new resource using the given resource data. * (PUT) Updates the specified resource using the given resource data. * (DELETE) Deletes the specified resource. * (3-part URI using ASSOCIATION) Returns the requested collection of association resources that are associated with the specified resource. * (4-part URI collection) Returns the requested collection of resources that are associated with the specified resource. | |
| Speak in the 3rd-person declarative | Describe what the entity is or what the entity does. You are not instructing the reader to do something.  Start with a verb by completing this sentence: “This method ...” | |
| **Use active voice** | If you use the verb to be + the past participle of a verb, you might be using passive voice. If you can avoid passive voice, consider rewriting in active voice.  Passive voice (Avoid):   * the name to be assigned to this thing   Active voice (Preferred):   * the String to set the name to | |
| **Avoid using Latin** | This is an internationalization issue.  Suggestions for replacements:   * etc. 🡪 and so forth * e.g. 🡪 for example * i.e. 🡪 that is * aka 🡪 and so forth * viz 🡪 in other words | |
| **Avoid future tense** | State what something does, not what it will do.  Future tense (Avoid):   * It will not remove the objects. * The results will be limited to ...   Present tense (Preferred):   * It does not remove the objects. * The results are limited to ... | |
| **End a sentence with a preposition (if it sounds right)** | It is acceptable to end a sentence with a preposition when doing otherwise sounds clumsy.  (Avoid):   * the name of the action for which the permission is checked * the string to which the name will be set   (Preferred):   * the name of the action to check permission for * the string to set the name to | |
| **Capitalization** | To avoid ambiguity, follow the company standards for capitalization of resources (lowerCamelCase) and model entities (UpperCamelCase).   * Tip: If you think such an ambiguity exists, seek the advice of a developer or of a technical writer. | |
| **Common Pitfalls** | Use this ... | Instead of this ... |
| cannot | can not |
| all of the *<things>* | all the *<things>* |
| the *<thing>* that is uniquely identified by... | the *<thing>* uniquely identified by... |
| the *<things>* that have ... | the *<things>* which have ... |
| defines a *<thing>*, which represents ... | defines a *<thing>* which represents ... |