Wonder Javadoc

> This page is work in progress.

This page presents thoughts and suggestions for creating and maintaining Javadocs in Wonder and should evolve to some sort of Javadoc coding standard.

What to document

Java classes should always be documented and have at least a short description in that a reader can quickly identify the purpose of the whole class.

All public and protected methods should be fully defined with Javadoc. Exception may be when a method is a simple getter or setter where a description would not add any information though it does no harm to do so. Package and private methods do not have to be documented, but may benefit from it. Methods that override another method should not have Javadocs except if their description would differ from the original one and include important additional information.

Comment Style

All Javadoc comments should be of the form

```
/**
 * Javadoc comment
 */
public …
```

If the Javadoc is short enough to keep on one line you can use the compressed form

```
/** compressed Javadoc comment */
public …
```

> Do not end a Javadoc comment with '**/' and do not start non-Javadoc comments with '/**'.

HTML usage

Use simple HTML tags if you need to add styling to the Javadoc. Don't use XHTML as that will only add unnecessary complexity and the Javadoc parser is expecting simple HTML anyways.

Paragraphs

Use single *p* tags between paragraphs to structure longer texts.

```
/**
 * A paragraph.
 * <p>
 * Another paragraph that runs on
 * multiple lines.
 */
public …
```

Lists

When inserting lists of options or enumerations use *li* tags

```
/**
 * A paragraph.
 * <p><ul>
 * <li>Alpha
 * <li>Beta
 * </ul><p>
 * Another paragraph
 */
public …
```

For correct paragraph formatting enclose the list with paragraph tags.

Code within comments

If you want to include Java code examples in Javadoc you have to wrap them in *pre* and *code* tags:

```
/**
 * A simple class that provides a resource-efficient WebObjects-friendly
ExecutorService for a
 * single application. Access the shared instance with:
 * <pre><code>
 * ExecutorService service = ERXExecutorService.executorService();
 * </code></pre>
 */
public class ERXExecutorService {
```

If it is only an inline code expression you should only use a *code* tag:

```
To enable super mode set the property <code>er.extensions.mode.super</code>
accordingly.
```

For the keywords *null*, *true* and *false* you should **not** use *code* tags as those keywords are very common in Javadoc and thus this would add much unnecessary code to Javadoc comments.

Mandatory Javadoc tags

There are some tags that should always be included if appropriate.

Class documentation

A class Javadoc should describe the purpose and if possible the usage of the class.

A class using generic type parameters should also have @param tags with a parameter name of <*T*> where *T* is the type parameter name.

Method documentation

A method Javadoc should always have a description. If the method takes any number of parameters you must add a @param tag for every parameter followed with the parameter name and a very short description. That description should be of the form of a phrase not a sentence and thus should not end with with a dot. Those param lines should be separated from the main description by an empty line and be ordered in the same order as they appear in the method declaration.

If a method returns a value you must add a @return tag after the last @param tag with a short description. Here too use a phrase instead of whole sentences. You should add a hint if the method could be returning null.

If the method throws exceptions you have to document them too with a @throws tag for every thrown type.

A method using generic type parameters should also have @param tags with a parameter name of <*T*> where *T* is the type parameter name.

An example of a well documented method:

```
/**
 * Returns the one object that matches the qualifier in the given array (or
 * null if there is no match).
 *
 * @param <T> the type of the objects
 * @param array the array to filter
 * @param qualifier the qualifier to filter with
 * @return one matching object or null
 * @throws IllegalStateException if more than one object matched
 */
public static <T> T one(NSArray<T> array, EOQualifier qualifier) { … }
```

Special tags

tbd

Non-English languages

The standard language Javadocs are written in is English. Currently there is only Japanese as additional allowed documentation language. When documenting in English you would have some method:

```
/**
 * Cover method for returning a float for a given system property.
 *
 * @param key system property
 * @return float value of the system property or 0
 */
public static float floatForKey(String key) { … }
```

As soon as you add the second language like Japanese you should convert this to the following form:

```
/**
 * <div class="en">Cover method for returning a float for a given system
property.</div>
 * <div class="ja"> float </div>
 *
 * @param key <div class="en">system property</div>
 *            <div class="ja"></div>
 * @return <div class="en">float value of the system property or 0</div>
 *         <div class="ja">float  0</div>
 */
public static float floatForKey(String key) { … }
```

You need to enclose the blocks for every language with a *div* tag and set a class attribute corresponding to the locale code, i.e. *en* for English and *ja* for Japanese. Those classes will be used in the generated Javadoc HTML pages to let the user switch between languages to display.

default Javadoc overlay in Eclipse	Javadoc overlay with multiple languages in Eclipse
float er.extensions.foundation.ERXProperties.floatForKey(String key) Cover method for returning a float for a given system property. **Parameters:** **key** system property **Returns:** float value of the system property or 0	

It should be checked if it is possible to add functionality to WOLips to be able to choose the displayed Javadoc language within Eclipse tooltips so even if multiple languages are defined in the Javadocs only a specific one is displayed. Any volunteers? 🙂

Multi-language Javadoc update

If a method has a multi-language Javadoc and someone is modifying text in only one of those languages then we need a marker signaling that a specific language needs yet to be updated. For that all affected div blocks should be tagged with an additional CSS class *needs_update* and have an '!' inserted at the beginning of their contents. Lets say the above Javadoc for floatForKey has been updated to reflect some changes in the method:

```
/**
 * <div class="en">Cover method for returning a float for a given system
property prefixed with "test.".</div>
 * <div class="ja needs_update">! float </div>
 *
 * @param key <div class="en">system property</div>
 *            <div class="ja"></div>
 * @return <div class="en">float value of the prefixed system property or
0</div>
 *         <div class="ja needs_update">!float  0</div>
 */
public static float floatForKey(String key) { … }
```

The English method and return descriptions have been altered and thus both corresponding Japanese div blocks got the additional CSS class *ne*

eds_update. As the param description for *key* didn't change no CSS class was added there. By this it is possible to search for those Javadoc pieces that need to be updated later on by anyone who is proficient in both languages. But as those CSS classes are not visible within the tooltips of Eclipse we need to insert a '!' character to the text so the user gets visual feedback that the description may be inexact.