Requirements for EGLdoc

***Draft**: 5 July 2012*

Ben Margolis, Advisory Writer, IBM

Table of Contents

Introduction

EGLdoc has the following aspects:

- Processes to copy doc from EGL code into a separate output.
- Rules for EGL developers to follow when writing the doc.

Any number of output formats might be supported. The initial requirement is to be compliant with XHTML 3.2. DITA might be of special interest in the next year or two.

The idea for EGLDoc is based on Javadoc™, which is described here:

javadoc - the Java API Documentation Generator (http://w3.java.ibm.com/java/docs/java7/technotes/tools/windows/javadoc.html)

Proposed Javadoc tags (http://java.sun.com/j2se/javadoc/proposed-tags.html)

The current spec relies on those descriptions, as customized for EGL.

Input doc components

The input doc has the following main components:

- Overview comment file

  Purpose: to describe an application or a group of EGL packages. One possible output is similar to this:

  http://docs.oracle.com/javase/7/docs/api/overview-summary.html

  The input file is HTML, with <html>, a subordinate <body> tag, and an optional set of overview annotations that are described later.

  Javadoc repeats the first sentence at the top and bottom of the output page. Seems wrong.

- Package comment file

  Purpose: to describe an EGL package. One possible output is similar to this:

  http://docs.oracle.com/javase/7/docs/api/java/math/package-summary.html

The input file is equivalent to the package-info.java file, which begins with a comment and ends with a package statement. The comment has an initial sentence with no annotations and can have additional content with an optional set of package annotations.

Again, Javadoc repeats the first sentence at the top and bottom of the output.

- Miscellaneous files

  Purpose: To provide content to the other files so that the output can include graphics, as well as more volume than is appropriately stored in an EGL source file.

- Comments embedded in source code

  Purpose: To document specific types, variables, constants, functions, and function prototypes.

  The input is composed of a main comment followed by a tag section. The main comment begins with the start characters “/**” and must not precede an import statement.

  A “block tag” is processed specially only if its @ symbol is first on the line except for any appropriately formatted start characters, white space, and asterisks. An “in-line tag” is embedded in curly braces.

Output format

The output for Javadoc can be appropriate for a multi-frame HTML (in which case the HTML output is prefaced with “HTML 4.0,” not 3.2) or for HTML without frames. The initial output for EGLdoc might be without frames, for inclusion in a doc plugin.

Tags

The next table lists the EGLdoc tags, with reference to Javadoc usage and with an indication of priority.

Priority	Tag	Purpose	Where available
	@author	To identify the developer. Can include multiple names in one tag, or multiple tags.	Overview document, package document, and types.
1	@code	To embed content in code font and without processing the content as HTML. For example, `@code <p>` shows `<p>`.	All.
1	@compat	To specify compatibility considerations.	All. A set of @compat entries are provided, one after the next. Each entry has two strings, separated by a vertical bar: the target followed by the compatibility issue. Subsequent vertical bars are processed as text. The developer should list the entries in alphabetical order by target; for example, Java then JavaScript.
	@deprecated	To identify an element as a candidate for nonsupport. See also @obsolete.	All.
	@docroot	To allow linking to a file higher in the hierarchy of destination folders. Such a file might be a company-specific logo or copyright statement.	All.
1	@example	To identify content that	All; but most important for types

Priority	Tag	Purpose	Where available
		goes under the "Example use" header. The tag does not automatically reference a file. @example is a proposed tag in Javadoc and would be useful to groups that want to explain their tech clearly. Here is a discussion: http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4075480. And here, for tutorials: http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4125834	and functions. The developer might reference tutorials.
	@link	To provide an in-line hyperlink rather than a hyperlink in a "See all" list, as is the case for @see. The Javadoc tag only supports linking to a package, class, or member.	All. Could support all the @see forms.
	@literal	To embed content in regular font and without processing the content as HTML. For example, `@code <p>` shows <p>.	All.
	@obsolete	To indicate that a construct is no longer supported though it remains in place. @obsolete is a proposed tag in Javadoc. See also @deprecated.	All.

Priority	Tag	Purpose	Where available
1	@operation	To indicate that an EGL function is converting data and is invoked by use of an operator.	Function and function prototypes.
1	@param	To state the purpose of a parameter and its range of argument values.	Functions and function prototypes.
1	@return	To state the purpose and range of a return value.	Functions and function prototypes.
	@see	To provide a hyperlink in a “See all” list rather than in line, as is the case for @link.	All. The Javadoc tag has several forms, and that spec has additional detail on how the compiler resolves references to types and members.
	@since	To indicate when a capability was introduced to an application.	Types, fields, methods, functions, function prototypes.
1	@throws	To indicate that a function throws an exception. If a function throws an exception that is not represented in a @throws tag, the exception is listed, without a description, in a “Throws” section of the output doc.	Functions, function prototypes.
1	@todo	To document what work is yet to be done. @todo is a proposed tag in Javadoc.	All.
	@value	To display the value of a constant, either in the context of that constant or elsewhere. Javadoc shows the collected values in a	Fields

Priority	Tag	Purpose	Where available
		page like this: http://w3.java.ibm.com/java/docs/java7/api/constant-values.html	
	@version	The current software version, not the version shown in @since.	Package overview, and types.

The following Javadoc tags are probably excluded:

- @exclude (a proposed tag), which identifies content to be excluded from the generated output. Not sure why you wouldn't leave this decision to the invocation command.

  Here are further details:
  http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4058216
- @index (a proposed tag), which specifies terms for use in a generated index. Few developers would use this, but the EDT project could. An alternative is to include index terms in a doc database that includes EGLdoc and topics, along with cross-reference links and possibly abstracts of the content.
- @internal (a proposed tag), which indicates that the content is for a company's internal use. Is similar to @exclude and is in some Javadoc implementations.

  Here are further details:
  http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4102647

The following Javadoc tags are excluded:

- @category (a proposed tag), which identifies grouped values.
- @exception, which gives details on class exceptions but is an alias of @throws.
- @inheritDoc, which retrieves data from elsewhere. Has one use that might make sense now or soon (when a method in a class overrides a method in an interface) and one use that is not current (when a method in a class or interface overrides a method in a superclass or superinterface).
- @linkplain, which is the same as @link but does not show the link in a fixed font.

  Couldn't the content of the @link tag indicate whether a font is needed? For the most part, we don't want tags that are specific to a kind of presentation. Also, couldn't the developer add @code tags if the font were needed, or @literal if not?
- @serial, @serialData, and @serialField, which concern serialization.
- @threadsafety, which indicates whether a class or method is thread safe.

- @tutorial (a proposed tag), which is specific to the Java tutorial and is fulfilled for EGL by use of the @example tag.

Javadoc does not include an @event tag, but later versions might list events based on an analysis of the code.

Someone used a @description tag in early EDT files, but the tag is unnecessary.

Last, any Javadoc user can add a block tag by including the -tag option on the command line. The practice should be made available to EGLdoc users, too.

Output

Choices for output:

- They might be more-or-less consistent with Javadoc or the Android variant:

  http://docs.oracle.com/javase/7/docs/api/index.html?java/util/ArrayList.html

  http://developer.android.com/reference/android/appwidget/AppWidgetHost.html

- They might mimic the reference-page templates now used in the EDT language reference. Those templates are consistent with standards that are typical for Eclipse doc; and, in this case, the HTML frames would be provided by Eclipse or by the Eclipse online Info Center, not by the output of the EGLdoc tool.

- Here's something, though. Tim Wilson points to the following site:

  http://docs.sencha.com/ext-js/4-0/#!/api/Ext.data.proxy.Ajax

EGLDoc output includes the following kinds of content:

- A list of all packages.
- A list of all types within a package.
- Package name.
- Type name.
- Kind of type; for example, annotation (a Record type), stereotype (a Record type), Handler type, and so forth.
- Type detail; sometimes the code itself, or the main features.
- A description of each field in the type.
- Example.
- For stereotypes, a list of member annotations, with links to the annotation page.
- Comments, for additional background.
- Compatibility, to describe differences by output language.
- Links to overview and, in some cases, task material.

The next sections give examples of EDT topics and the related HTML now being produced by DITA OT, including anchor tags that are specific to Eclipse. [How would references to other EGLdoc files be structured?] The output for user-defined records, services, and so forth would be similar, with template-based categories and tables.

Records

The EGLdoc output is different for the following kinds of Records: Records other than annotations, annotations other than stereotypes, and stereotypes.

Records other than annotations

Assume the following EGL source input:

```
package example.com.mydata;

/**
 * CustomerType identifies the attributes of a customer.
 *
 * This statement is a comment.
 *
 */
Record CustomerType {@Table {catalog = "MyCat", Name = "My_Table",
                     schema = "MySchema", shortName = "A"}}

  /**
   *  The customer name.
   *
   */
  name string {@ExternalName{"the_name"}};

  /**
   *  The customer number.
   *
   *  Is unique across all our divisions.
   */
   number int;

  /**
   *  A list of purchasing agents and their contact information.
   *
   */
   purchasers Person[];
end
```

Here is the displayed output, followed by the XHTML:

CustomerType Record

CustomerType identifies the attributes of a customer.

Type stereotype

None.

Type annotations

@Table { catalog = "MyCat", Name = "My_Table", schema = "MySchema", shortName = "A" }

EGL package name

example.com.mydata

Record fields

name String

The customer name.

Field annotations:

@ExternalName { value = "the_name" }

number Int

The customer number.

Is unique across all our divisions.

purchasers Person **[ ]**

A list of purchasing agents and their contact information.

Comments

This statement is a comment.

```
[ might have a compatibility table, as shown in later examples. ]
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xml:lang="en-us" lang="en-us">
<head>
<!-- have not looked into the meta tags too deeply;
     but the ones here are used in Eclipse. -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"></meta>
<meta name="DC.Type" content="topic"></meta>

<!-- from the type and its category -->
<meta name="DC.Title" content="CustomerType Record"></meta>

<!-- the next two come from the first paragraph -->
<meta name="abstract" content="CustomerType identifies the attributes of a
customer. "></meta>
<meta name="description" content="CustomerType identifies the attributes of a
customer."></meta>
```

```
<!-- the next two are used for search, not indexing -->
<meta name="DC.subject" content="CustomerType"></meta>
<meta name="keywords" content="CustomerType"></meta>


<meta name="copyright" content="(C) Copyright 2011, 2012" type="primary"></meta>
<meta name="DC.Rights.Owner" content="(C) Copyright 2011, 2012"
type="primary"></meta>
<meta name="DC.Format" content="XHTML"></meta>

<!-- is unique among pages:  the fully qualified type -->
<meta name="DC.Identifier" content="example.com.mydata.CustomerType"></meta>
<meta name="DC.Language" content="en-us"></meta>

<!-- assumes continued use of the CSS now in the help system -->
<link rel="stylesheet" type="text/css"
href="style/commonltr.css"></link>


<!-- if the type is a Record that is not stereotyped as an annotation, entitle the
page with the type name and the word "Record" -->
<title>CustomerType Record</title>
</head>
<body>
<h1 class="title topictitle1">CustomerType Record</h1>
<div class="body" id="body">

<!-- from the first paragraph in the EGLDoc description -->
<p class="shortdesc"><span class="ph synph"><span
class="keyword kwd">CustomerType</span></span> identifies the attributes of a
customer.</p>


<!-- start list -->
<dl class="dl" id="main">
<dt class="dt dlterm"><a name="typestereo"</a>Type stereotype</dt>


<!-- from type definition -->
<dd> <p class="p">None.</p>
     <p class="p"></p>
</dd></dt>


<!-- if multiple annotations, separate them with empty paragraphs -->
<dt class="dt dlterm"><a name="typeanno"</a>Type annotations</dt>
<dd>
```

<!-- from type definition. Leave a single space between the curly bracket and content, on each side, and ensure commas and single spaces are used internally. -->

```
@<a href="eglx.persistence.Table">Table</a> { catalog = "MyCat", Name = "My_Table",

schema = "MySchema", shortName = "A" }


<p class="p"></p>

</dd></dt>


<dt class="dt dlterm"><a name="package"></a>EGL package name</dt>
```

<!-- from the package statement and some naming convention for file access -->

```
<dd class="dd"><p class="p"><a
href="example.com.mydata">example.com.mydata</a></p></dd>

<dt class="dt dlterm"><a name="recordfields"></a>Record fields</dt>
```

<!-- is the destination for EGLDoc comments that precede Record fields.

Include the last , whether or not the content is supplied, but include "None" if none. -->

```
<dd class="dd"><dl class="dl parml"><p class="p"></p>

<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">name</span>
```

<!-- uses the Estring definition, calls it String. do an equivalent type renaming when refering to any of the EGL simple-type definitions -->

```
<a href=eglx.lang.EString>String</a></span></dt>

<dd class="dd pd">The customer name.
```

<!-- automatically handle field annotations, each separated from the next by a paragraph -->

```
<p class="p"></p>

Field annotations:

<dl dlterm><dt class="dt"></dt>

<dd class="dd">@<a href="eglx.lang.ExternalName">ExternalName</a> { value =
"the_name" }</dd><p class="p"></p></dl></dd>


<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">number</span>

<a href="eglx.lang.EInt">Int</a>

</span></dt>

<dd class="dd pd"><class="p">The customer number.
<p class="p">Is unique across all our divisions.</p>
<p class="p"></p></dd>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
```

```
kwd">purchasers</span>
<a href="example.org.Person">Person</a> [ ]</span></dt>
<dd class="dd pd">A list of purchasing agents and their contact information.
</dd></dd>

<!-- after the two </dd> end tags -->
<p class="p"></p>
<!-- from paragraphs that come after the first and that precede either the first
EGLDoc tag or (if no tags) the end of the initial EGLDoc comment. Ensure the
presence of <p></p>, whether or not the content is supplied. →
<dt class="dt dlterm"><a name="comments"></a>Comments</dt>
<dd class="dd">This statement is a comment.</dd>
<!-- for potential additions of links in a postprocess. →
<anchor id="related_links"></anchor>
</div>
</body>
</html>
```

Annotations other than stereotypes

Assume the following EGL source input:

```
package eglx.xml.binding.annotation;

/**
 * XMLAttribute provides details for a Record field
 * that represents an XML attribute.
 *
 * This statement is a comment.
 *
 * @compat Java | No issues.
 *
 * @compat JavaScript | No issues.
 *
 * @example See <a href=
"www.eclipse.org/edt/helptop/org.eclipse.edt.core.doc.lr/topics/redt00254.html">
”Correspondence between an XML string and an EGL variable.”</a>
 *
 *
 */
Record XMLAttribute type Annotation
{ targets = [ ElementKind.fieldMbr ]}

  /**
```

```
   *    The name of the XML attribute.
   *
   *    If you are writing a record to an XML string...
   */
  name string  = "##default";

  /**
   *  The XML namespace associated with the XML attribute.  Note:
   *    <ul>
   *      <li> abc </li>
   *      <li> def </li>
   *    </ul>
   *
   *  When reading an XML string...
   */
   namespace string = "##default";

    /**
     *
     *  Indicates whether the attribute must be specified in the XML.
     *
     */
   required boolean = false;
end
```

Here is the displayed output, followed by the XHTML:

XMLAttribute annotation

XMLAttribute provides details for a Record field that represents an XML attribute.

EGL package name

eglx.xml.binding.annotation

Example use

See Correspondence between an XML string and an EGL variable.

Annotation detail

```
Record XMLAttribute type Annotation {targets =
[ ElementKind.fieldMbr ]}
   name string  = "##default";
   namespace string = "##default";
   required boolean = false;
end
```

Annotation fields

name String

The name of the XML attribute. The default value is the name of the record field.

If you are writing a record to an XML string...

namespace String

The XML namespace associated with the XML attribute. If Note:

- abc
- def

When reading an XML string...

required String

Indicates whether the attribute must be specified in the XML.

Comments

This statement is a comment.

Compatibility

Target	Issue
Java	No issues.
JavaScript	No issues.

[late change: include no “Annotation detail” section. Instead, place a section (in the same place) named “Annotation targets” and identify the targets, one paragraph after the next. Add <p class=”p”</p> in between them and at the end.]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xml:lang="en-us" lang="en-us">
<head>

<!-- have not looked into the meta tags too deeply;
     but the ones here are used in Eclipse. -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"></meta>
```

```
<meta name="DC.Type" content="topic"></meta>

<!-- from the type and its category -->
<meta name="DC.Title" content="XMLAttribute annotation"></meta>

<!-- the next two come from the first paragraph -->
<meta name="abstract" content="XMLAttribute provides details for a Record field
that represents an XML attribute."></meta>
<meta name="description" content="XMLAttribute provides details for a Record
field that represents an XML attribute."></meta>

<!-- the next two are used for search, not indexing -->
<meta name="DC.subject" content="XMLAttribute, EGL annotation,
annotations"></meta>
<meta name="keywords" content="XMLAttribute, EGL annotation,
annotations"></meta>

<meta name="copyright" content="(C) Copyright 2011, 2012" type="primary"></meta>
<meta name="DC.Rights.Owner" content="(C) Copyright 2011, 2012"
type="primary"></meta>
<meta name="DC.Format" content="XHTML"></meta>

<!-- is unique among pages:  the fully qualified type -->
<meta name="DC.Identifier"
content="eglx.xml.binding.annotation.XMLAttribute"></meta>
<meta name="DC.Language" content="en-us"></meta>

<!-- assumes continued use of the CSS now in the help system -->
<link rel="stylesheet" type="text/css"
href="style/commonltr.css"></link>

<!-- if the type is a Record that is stereotyped as an annotation and that lacks
the @Stereotype annotation, entitle the page with the type name and the word
"annotation" -->
<title>XMLAttribute annotation</title>
</head>
<body>
<h1 class="title topictitle1">XMLAttribute annotation</h1>
<div class="body" id="body">

<!-- from the first paragraph in the EGLDoc description. Use of type name (with
first-letter capped) causes use of the CSS keyword kwd class. -->
<p class="shortdesc"><span class="ph synph"><span
class="keyword kwd">XMLAttribute</span></span> provides
details for a Record field that represents an XML attribute.</p>

<dl class="dl" id="main">
<dt class="dt dlterm"><a name="package"></a>EGL package name</dt>

<!-- from the package statement and some naming convention -->
<dd class="dd"><p class="p"><a
href=eglx.xml.binding.annotation>eglx.xml.binding.annotation</a></p>
</dd>
<dt class="dt dlterm"><a name="use"></a>Example use</dt>

<!-- from the @example tag. Even if no such tag is present, include <p
class="p"></p>. -->
```

```
<dd class="dd"><p class="p">See
<a href="doc/topics/org.eclipse.edt.core.doc.lr/topics/redt00254.html">
Correspondence between an XML string and an EGL variable</a>.</p>
</dd>
<dt class="dt dlterm"><a name="annorec"></a>Annotation detail</dt>

<!-- is the definition itself, minus any comments embedded there -->
<dd class="dd"><pre class="pre codeblock">Record XMLAttribute type Annotation
{targets =
[ ElementKind.fieldMbr ]}
   name string  = "##default";
   namespace string = "##default";
   required boolean = false;
end</pre>
</dd>
<dt class="dt dlterm"><a name="annofields"></a>Annotation fields</dt>

<!-- is the destination for EGLDoc comments that precede annotation fields.
Include the last <p class="p"></p>, whether or not the content is supplied. -->
<dd class="dd"><dl class="dl parml"><p class="p"></p>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">name</span>

<!-- uses the Estring definition, calls it String. do an equivalent type
renaming when refering to any of the EGL simple-type definitions -->
<a href=eglx.lang.EString>String</a></span>

</dt>
<dd class="dd pd">The name of the XML attribute. The default value is the name
of the record field.<p class="p">If you are writing a record to an XML
string...</p>
</dd>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">namespace</span>
<a href=eglx.lang.EString>String</a></span></dt>

<dd class="dd pd"><div class="p">The XML namespace associated with the XML
attribute. If
Note:
<ul class="ul">
<li class="li">abc</li>
<li class="li">def</li>
</ul>
</div>
<p class="p">When reading an XML string...</p>
</dd>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">required</span>
<a href=eglx.lang.EString>String</a></span></dt>
<dd class="dd pd">Indicates whether the attribute must be specified in the
XML.</dd>
</dl><p class="p"></p>
</dd>


<!-- from paragraphs that come after the first and that precede either the first
EGLDoc tag or (if no tags) the end of the initial EGLDoc comment. Ensure the
```

```
<!-- from the set of @compat tags, in developer-specified order (should be
alphabetic). Ensure the presence of <p></p>, whether or not the content is
supplied; but include "None" if none.-->
<dt class="dt dlterm"><a name="compat"></a>Compatibility</dt>
<dd class="dd"><p class="p"></p>
<div class="tablenoborder"><table cellpadding="4" cellspacing="0" summary=""
id="compat_table" class="table" frame="border" border="1"
rules="all">
<thead class="thead" align="left">
<tr class="row" valign="bottom">
<th class="entry" valign="bottom" width="20%">Target</th>
<th class="entry" valign="bottom" width="80%">Issue</th></tr>
</thead>
<tbody class="tbody">
<tr id="compat_java" class="row">
<td class="entry" valign="top" width="20%">Java</td>
<td class="entry" valign="top" width="80%">No issues.</td>
</tr>
<tr id="compat_javascript" class="row">
<td class="entry" valign="top" width="20%">JavaScript</td>
<td class="entry" valign="top" width="80%">No issues.</td>
</tr>
</tbody>
</table>
</div>
</dd>
</dl>
</div>

<!-- for potential additions of links in a postprocess. -->
<anchor id="related_links"></anchor>
</body>
</html>
```

Stereotypes

Assume the following EGL source input:

```
package eglx.persistence.entity

/**
 * Entity indicates that a Record, Handler, or External type
 * represents a value that can be persisted and that
 * might be related to other such values.
 *
 * This statement is a comment.
 *
 * @compat Java | No issues.
 *
```

```
 * @compat JavaScript | No issues.
 *
 * @example See <a href=
"www.eclipse.org/edt/helptop/org.eclipse.edt.core.doc.lr/topics/redt00081.html">
SQL example</a>.
 *
 *
 */
Record Entity type Annotation {
   targets = [ExternalTypePart, RecordPart, HandlerPart ],
   @Stereotype {
      memberAnnotations = [OneAnno, TwoAnno]
   }
}
   /**
    *    Enables...
    *
    *    Moreover...
    */
   name string  = "##default";

   /**
    *  Correlates two details:
    *    <ul>
    *      <li>the first</li>
    *      <li>the second</li>
    *    </ul>
    *
    *  If the correlation enables...
    */
    namespace string = "##default"
end
```

Here is the displayed output, followed by the XHTML:

Entity stereotype

Entity indicates that a Record, Handler, or External type represents a value that can be persisted and that might be related to other such values.

EGL package name

eglx.persistence

Example use

See SQL example.

Stereotype detail

```
Record Entity type Annotation {
   targets = [ExternalTypePart, RecordPart, HandlerPart ],
   @Stereotype {
        memberAnnotations = [OneAnno, TwoAnno]
   }
}
end
```

Stereotype fields

name String

Enables...

Moreover...

namespace String

Correlates two details:

- the first
- the second

If the correlation enables...

Annotations for each member field

- OneAnno
- TwoAnno

Comments

This statement is a comment.

Compatibility

Target	Issue
Java	No issues.
JavaScript	No issues.

[late change: include no “Stereotype detail” section. Instead, place a section (in the same place) named “Annotation targets” and identify the targets, one paragraph after the next. Add <p class=”p”</p> in between them and at the end.]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xml:lang="en-us" lang="en-us">
```

```
<head>

<!-- have not looked into the meta tags too deeply;
     but the ones here are used in Eclipse. -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"></meta>
<meta name="DC.Type" content="topic"></meta>

<!-- from the type and its category -->
<meta name="DC.Title" content="Entity stereotype"></meta>

<!-- the next two come from the first paragraph -->
<meta name="abstract" content="Entity indicates that a Record, Handler, or
External type represents a value that can be persisted and that might be related
to other such values."></meta>
<meta name="description" content="Entity indicates that a Record, Handler, or
External type represents a value that can be persisted and that might be related
to other such values."></meta>

<!-- the next two are used for search, not indexing -->
<meta name="DC.subject" content="Entity, EGL stereotype, stereotypes"></meta>
<meta name="keywords" content="EGL stereotypes, Entity, Entity stereotype,
Record classifier, ExternalType classifier, Handler classifier"></meta>

<meta name="copyright" content="(C) Copyright 2011, 2012" type="primary"></meta>
<meta name="DC.Rights.Owner" content="(C) Copyright 2011, 2012"
type="primary"></meta>
<meta name="DC.Format" content="XHTML"></meta>

<!-- is unique among pages:  the fully qualified type -->
<meta name="DC.Identifier" content="eglx.persistence.Entity"></meta>
<meta name="DC.Language" content="en-us"></meta>

<!-- assumes continued use of the CSS now in the help system -->
<link rel="stylesheet" type="text/css"
href="style/commonltr.css"></link>

<!-- if the type is a Record that is stereotyped as an annotation, and if the
@Stereotype annotation is present, entitle the page with the type name and the
word "stereotype" -->
<title>Entity stereotype</title>
</head>
<body>
<h1 class="title topictitle1">Entity stereotype</h1>
<div class="body" id="body">

<!-- from the first paragraph in the EGLDoc description. Use of type name (with
first-letter capped) causes use of the CSS keyword kwd class. -->
<p class="shortdesc"><span class="ph synph"><span
class="keyword kwd">Entity</span></span> indicates that a Record, Handler, or
External type represents a value that can be persisted and that might be related
to other such values.</p>

<dl class="dl" id="main">
<dt class="dt dlterm"><a name="package"></a>EGL package name</dt>

<!-- from the package statement -->
<dd class="dd"><p class="p">eglx.persistence</p>
```

```
</dd>
<dt class="dt dlterm"><a name="use"></a>Example use</dt>

<!-- from the @example tag. Even if no such tag is present, include <p
class="p"></p>. -->
<dd class="dd"><p class="p">See
<a
href="www.eclipse.org/edt/helptop/org.eclipse.edt.core.doc.lr/topics/redt00081.h
tml">
SQL example</a>.</p>
</dd>
<dt class="dt dlterm"><a name="stereorec"></a>Stereotype detail</dt>

<!-- is the definition itself, minus any comments embedded there -->
<dd class="dd"><pre class="pre codeblock">Record Entity type Annotation {
   targets = [ExternalTypePart, RecordPart, HandlerPart ],
   @Stereotype {
      memberAnnotations = [OneAnno, TwoAnno]
   }
}
end</pre>
</dd>
<dt class="dt dlterm"><a name="stereofields"></a>Stereotype fields</dt>

<!-- is the destination for EGLDoc comments that precede stereotype fields.
Include the last <p class="p"></p>, whether or not the content is supplied, but
include "None" if none. -->
<dd class="dd"><dl class="dl parml"><p class="p"></p>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">name</span>


<!-- uses the Estring definition, calls it String. do an equivalent type
renaming when referring to any of the EGL simple-type definitions -->
<a href=eglx.lang.EString>String</a></span>

</dt>
<dd class="dd pd">Enables... <p class="p">Moreover...</p>
</dd>
<dt class="dt pt dlterm"><span class="ph synph"><span class="keyword
kwd">namespace</span>

<a href=eglx.lang.EString>String</a></span></dt>
<dd class="dd pd"><div class="p">Correlates two details:
<ul class="ul">
<li class="li">the first</li>
<li class="li">the second</li>
</ul>
</div>
<p class="p">If the correlation enables...</p>
<p class="p"></p></dd></dd>

<dt class="dt dlterm"><a name="memberannos"></a>Annotations for each member
field</dt>

<!-- only links to the EGLDoc for each annotation.
Include the last <p class="p"></p>, whether or not the content is supplied, but
```

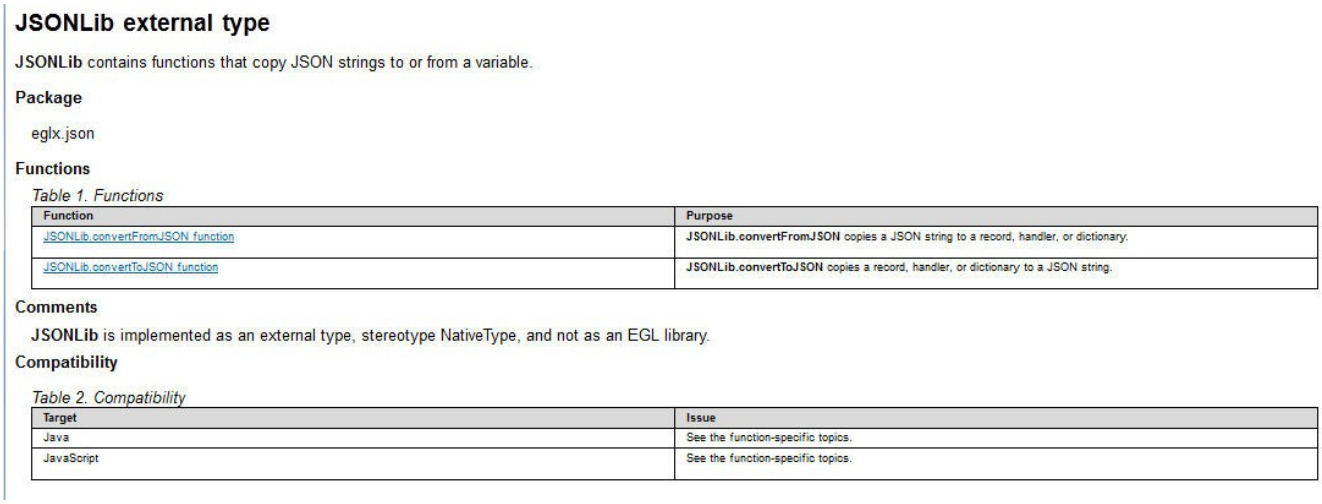
```
include "None" if none. -->
<ul class="ul">
<li class="li">
<a href="www.eclipse.org/edt/helptop/redt00001.html">OneAnno<a/></li>
<li class="li"><a
href="www.eclipse.org/edt/helptop/redt00002.html">TwoAnno<a/></li>
</ul>
<p class="p"></p></dd>

<!-- from paragraphs that come after the first and that precede either the first
EGLDoc tag or (if no tags) the end of the initial EGLDoc comment. Ensure the
presence of <p></p>, whether or not the content is supplied. -->
<dt class="dt dlterm"><a name="comments"></a>Comments</dt>
<dd class="dd">This statement is a comment.</dd>

<!-- from the set of @compat tags, in developer-specified order (should be
alphabetic). Ensure the presence of <p></p>, whether or not the content is
supplied.-->
<dt class="dt dlterm"><a name="compat"></a>Compatibility</dt>
<dd class="dd"><p class="p"></p>
<div class="tablenoborder"><table cellpadding="4" cellspacing="0" summary=""
id="compat_table" class="table" frame="border" border="1"
rules="all">
<thead class="thead" align="left">
<tr class="row" valign="bottom">
<th class="entry" valign="bottom" width="20%">Target</th>
<th class="entry" valign="bottom" width="80%">Issue</th></tr>
</thead>
<tbody class="tbody">
<tr id="compat_java" class="row">
<td class="entry" valign="top" width="20%">Java</td>
<td class="entry" valign="top" width="80%">No issues.</td>
</tr>
<tr id="compat_javascript" class="row">
<td class="entry" valign="top" width="20%">JavaScript</td>
<td class="entry" valign="top" width="80%">No issues.</td>
</tr>
</tbody>
</table>
</div>
</dd>
</dl>
</div>

<!-- for potential additions of links in a postprocess. -->
<anchor id="related_links"></anchor>
</body>
</html>
```

External types

The web page for an external type has several characteristics of other pages: an introduction; a type annotation, a package identifier; a field list, if appropriate; comments, a compatibility table (without a table heading), and the bottommost placeholder for related links.

The table of functions (without a table heading) is structured the same as a compatibility table.

At this time, constants are not valid in an external type; but that's a bug. Constants should be listed in a “Constants” section, preceding “Functions” and in the same format as the fields.



JSONLib external type

JSONLib contains functions that copy JSON strings to or from a variable.

Package

eglx.json

Functions

Table 1. Functions

Function	Purpose
JSONLib.convertFromJSON function	**JSONLib.convertFromJSON** copies a JSON string to a record, handler, or dictionary.
JSONLib.convertToJSON function	**JSONLib.convertToJSON** copies a record, handler, or dictionary to a JSON string.

Comments

JSONLib is implemented as an external type, stereotype NativeType, and not as an EGL library.

Compatibility

Table 2. Compatibility

Target	Issue
Java	See the function-specific topics.
JavaScript	See the function-specific topics.

Functions

The web page for a function n external type has several characteristics of other pages: an introduction; a package identifier; comments; a compatibility table (without a table heading); and the bottommost placeholder for related links.

The <pre></pre> entry in the syntax section is followed by the definition list (<dl>) tags, as shown earlier. The EGLDoc @return tag needs to be represented...

JSONLib.convertFromJSON function

JSONLib.convertFromJSON copies a JSON string to a record, handler, or dictionary.

Package

eglx.json

Syntax

```
static function convertFromJSON(json string in,  eglType any const in);
```

json

A JSON string.

eglType

A record, handler, or dictionary.

Example use

Although the second parameter is declared as ANY type, the EGL runtime verifies that the input is a record, handler, or dictionary. Because of the declaration, you can use the function in a library function that accepts any type of variable, as in the following example:

```
function convert(myRecord MyRecordPart in, record2Populate ANY const in)
   jsonLib.convertFromJSON(myRecord.data, record2Populate);
end
```

Comments

For other details on the conversion, see "Correspondence between a JSON string and an EGL variable."

JSONLib.convertFromJSON is the complement of the **JSONLib.convertToJSON** function.

Compatibility

Table 1. Compatibility

Target	Issue
Java	No issues.
JavaScript	The EGL runtime code rounds any numeric data that is greater than 15 significant digits.

[other types must be presented, with a format for function prototypes...]

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Android is a trademark of Google Inc.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names, may be trademarks or service marks of others.