XMLmind XML Editor - Javadoc[tm] Format Plug-in

Hussein Shafie, Pixware

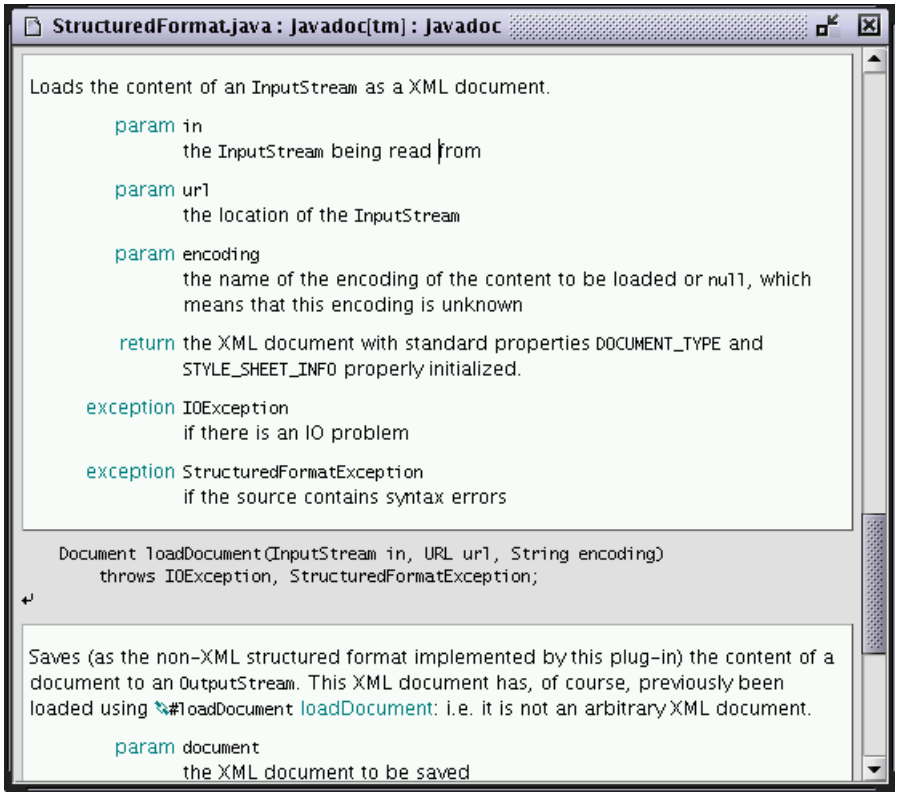
December 2, 2005

Table of Contents

Important: before using this plug-in for the first time, please take the time to configure it properly to make sure that its newline and tab policies are compatible with yours. See options below.

1. Writing Javadoc™ comments using XXE

Figure 1. A Java file as displayed by XXE



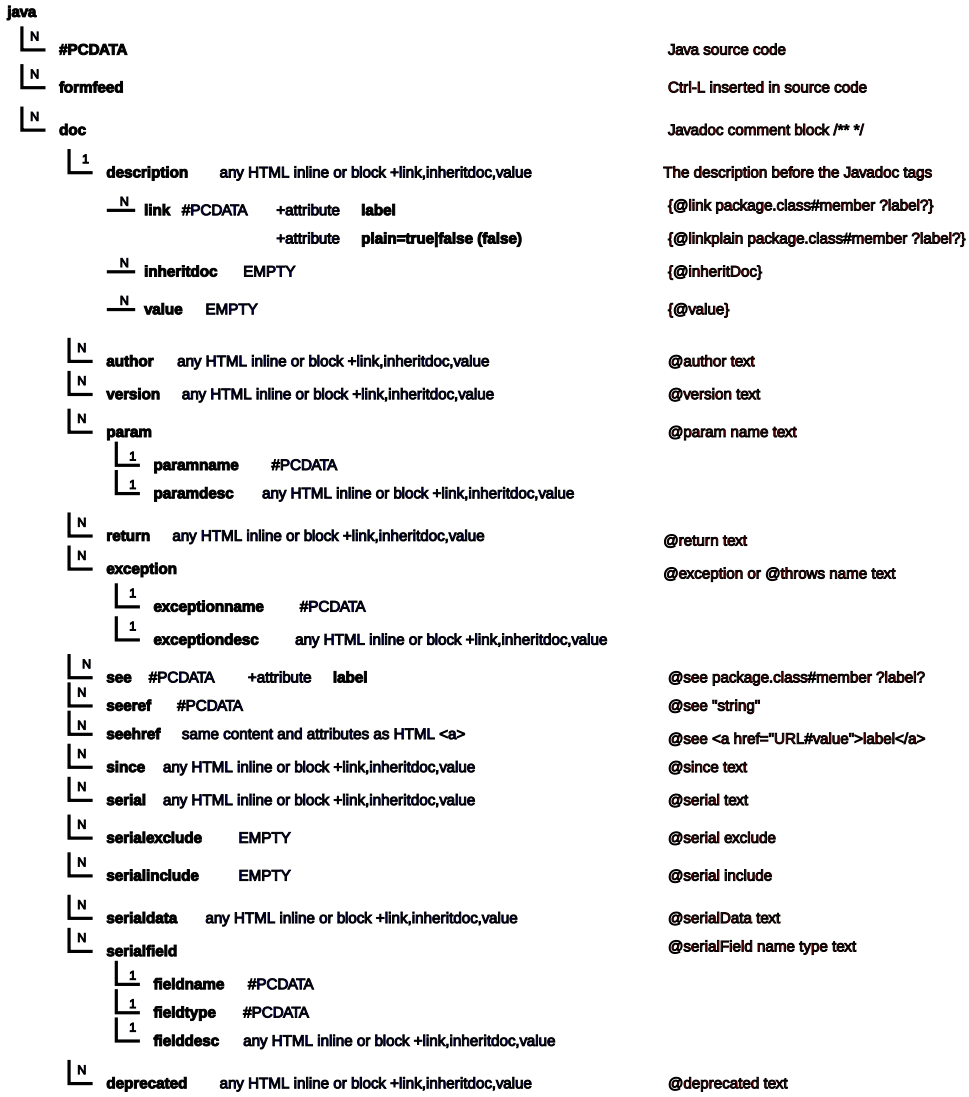
The intended audience for this plug-in is Java™ programmers and Javadoc™ writers. With XXE and this plug-in, it becomes possible to edit the Javadoc comments contained in a Java file using a word processor-like view. Writing Javadoc this way is less tedious (no manual formatting of comment lines) and is no longer error-prone (DTD-directed editing).

This plug-in has full support for all Javadoc 1.4 tags (see Javadoc Home Page) plus most of HTML 3.2 tags.

The following HTML 3.2 tags and attributes are *not* supported:

- html,
- head, isindex, base, title, meta, link, script, style,
- body,
- object, applet, param,
- menu, dir,
- map, area, img usemap and ismap attributes,
- form, input, select, option, textarea,
- xmp, listing, plaintext.

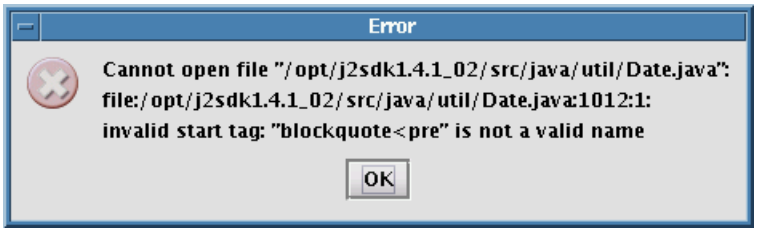
The Javadoc 1.4 tags are modeled in XML as follows:



The file name extension required for a ``Javadoc document'' is `.java`.

Unlike Web browsers, this plug-in is not designed to load broken HTML 3.2. However, this plug-in can help Javadoc writers to easily spot and fix the HTML errors contained in Javadoc comments. See next section for a real world case study.

Figure 2. The Javadoc plug-in refuses to load Sun's Date.java



When XXE refuses to load a Java file, an error dialog is displayed with

- an error message (not always easy to understand due to the layered architecture),
- the line number of the *start* of the Javadoc comment block where the error occurred,
- a column number always equal to 1.

Try to guess what the error message means and fix the problem using your favorite text editor, then reload the Java file into XXE.

For the above example, it is pretty easy to fix the problem:

```
    /**
     * Creates a string representation of this <tt>Date</tt> object of
     * the form:
     * <blockquote<pre>
     * d mon yyyy hh:mm:ss GMT</pre></blockquote>
```

1.1. Applying the Javadoc plug-in to Sun JDK 1.3.1 sources

The Javadoc plug-in has been tested on all the Java sources given by Sun for the Linux JDK 1.3.1.

The plug-in has succeeded to load 1750 out of 1877 Java files. It has failed 127 times generally for the following reasons:

- The Javadoc writer adds `<code></code>` tags at places where plain text (`#PCDATA`) is expected.

  Example: putting the name of a `@param` between `<code></code>`.

  Note that it is not useful to do so because Javadoc automatically adds a sensible style to this kind of plain text.

  This is the most important discrepancy between the Javadoc plug-in and Javadoc: Javadoc intelligently discards `<code></code>`, while the Javadoc plug-in stubbornly refuses to load the Java file.

- Typos such as:
  - typos in tag names (examples: `blockquoute`, `coder`)
  - forgetting `'>'` at the end of start or end tags
  - unknown character entities (example: `&le;`)
  - putting a space between the attribute name, the `'='` sign and the attribute value
  - using end tags (example: `</p>`) instead of start tags
  - duplicating attributes (example: `align`)
- The Javadoc comments writer sometimes forgets what he has learned about HTML.

  Example 1: putting plain text or inline elements such as `<code></code>` directly into a `blockquote`.

Example 2: putting a `pre` block between `<code></code>`.

- `'>'`, `'<'`, `'&'` not properly escaped. Here, the plug-in tries to do its best but sometimes it misses some of these special characters.

Note that all the errors described above are impossible to do if XXE is used to edit the Javadoc comments.

Having succeeded to load a Java file into XXE does not mean that this file is valid: 45 out the 1750 loaded files were found having validity problems.

Generally these validity problems are minor and are caused by missing attributes or invalid attribute values. It is possible to quickly fix them by using menu command **Tools**|**Check Validity**.

2. Getting started

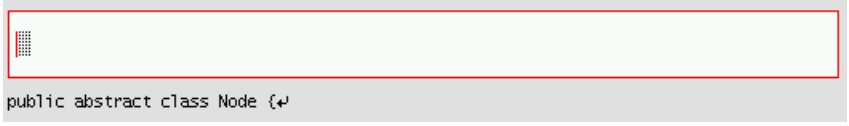
1. Open a `.java` file in XXE.

   XXE uses the Javadoc plug-in to convert the Java file on the fly to an equivalent XML file. Then, it associates a configuration with the newly opened Javadoc document. The XXE configuration file bundled with the plug-in (if deployed properly) creates a very handy Javadoc tool bar.

2. Move caret to the beginning of the line where the class is declared.

   ```
   +-- Move caret here
   |
   |public abstract class Node {
   ```

3. Insert a `doc` element here (for example, used **Edit**|**Insert** or simply click on the **Insert doc** icon of the Javadoc tool bar).

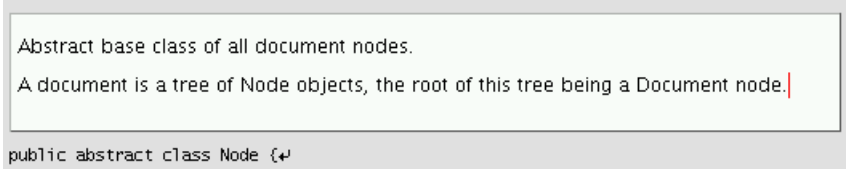
4. First sentence, ended by a period, should describe the purpose of the class. It may be followed by other sentences. Type these sentences in the text placeholder.

   For example, in what follows, we just type a single sentence.

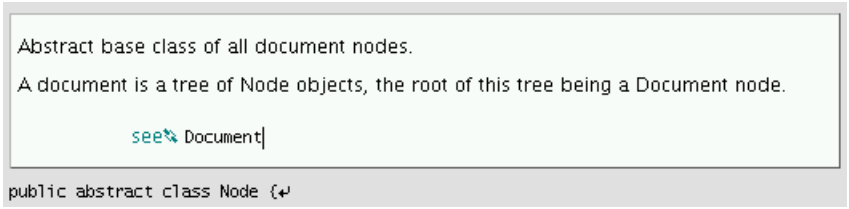
5. The sentences found at the beginning of a `doc`/`description` may be followed by any HTML 3.2 block (`p`, `pre`, `ul`, `ol`, `dl`, `table`, etc) if needed too.

   For example, in what follows, we insert a `p` after first sentence (use **Edit**|**Insert** or click on the **Insert or add block** icon of the Javadoc tool bar and choose the **p** entry).

6. The `description` element may be followed by elements corresponding to Javadoc tags (`@param`, `@return`, `@exception`, `@see`, etc).

For example, in what follows, we insert a see after the description (explicitly select `description` and then use **Edit**|**Insert After** or more simply, just click on the **Add see** icon of the Javadoc tool bar).



Documenting a field is similar to what has been described above. Move caret to the beginning of the line where the field is declared and insert a `doc` element here.

```
+-- Move caret here
|
|    public static final int ELEMENT = 4;
```

Documenting a method is similar to what has been described above. Move caret to the beginning of the line where the method is declared and insert a `doc` element here.

```
+-- Move caret here
|
|    public Tree getParent() {
         return parent;
     }
```

3. Plug-in options

Expand/unexpand tabs
: If this option is turned on:

  - When loading a Java file, replace all tab characters contained in source code by equivalent space characters using the number specified by the **Tab width** field.
  - When saving a Java file, replace space characters contained in source code by equivalent tab characters using the number specified by the **Tab width** field.

  This option is needed because currently, unlike all text editors, XXE cannot *display* tab characters expanded.

  Turning this option off makes Java source code less readable in XXE but has the advantage of not modifying the source code at all (which may be very important for source code managed using a version control system).

  Default: checked.

Tab width
: Distance in characters between tab stops.

  Default: 8.

Line separator
: When saving a Java file, use this string to separate lines. Line separator is "`\n`" on Unix/Linux/MacOS X, "`\r\n`" on Windows, "`\r`" on Mac (before MacOS X).

  Use "`-`" to specify the native line separator of the platform whatever it is.

  Default: "`-`" (platform native line separator).

Max. line length
: When saving a Java file, try not to generate Javadoc lines that exceed this length.

  Default: 78.

Default encoding

Encoding used when loading and saving a Java file if the encoding has not been specified by other means (for example by a HTTP connection).

Use "-" to specify the native encoding of the platform whatever it is.

Default: "-" (platform native encoding).