Proposed JAVA Comment/Header Coding Standard

Comment Style:

The C++ comment, "//", will be used in all places except for javadoc comments.

If the field to document, such as "@exception", does not apply, then it should not be specified at all.

javadoc Images:

A zip file of the images is available. If there are any missing please let me know.

javadoc Documentation:

For instructions on how to use javadoc see the javadoc documentation.

javadoc gotchas:

- You must begin the @param, @exception, etc. lines with a space... NOT a tab.
- Private variables and methods are currently not printed, but document them just the same. We have hopes that they will be pulled out by javadoc in the future.
- Don't put Ctrl-L in the code.
- Indent the entire comment block with the code.

File Header:

```
//
// sample.java
//
// Copyright 1997  DataBeam Corporation.  All rights reserved.
//
// U.S. Government Rights
//
// Use, duplication, or disclosure by the U.S. Government is subject to
// restrictions set forth in the DataBeam Corporation license agreement
// and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS
// 252.227-7013(c)(1)(ii)(OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19,
// or FAR 52.227-14 (ALT III), as appliable.  DataBeam Corporation.
//
```

Class Header (for use with javadoc)

```
/**
 * <p>
 * The functional description of the class would go here.  This
 * description can span multiple lines.
 *
 * @author      Jane Doe
 * @author      Jimmy C. Corn
 *
 * @see         SomeRelatedClass
 */
```

Constructor/Method Header

```
/**
 * The functional description of the constructor or method would go here.
 * This description can span multiple lines.
 *
 * @param param1
 *      A description of "param1".  This description can span multiple
 *      lines.
 *
 * @param param2
 *      A description of "param2".  This description can span multiple
 *      lines.
 *
 * @param paramX
 *      A description of "paramX".  This description can span multiple
 *      lines.
 *
 * @return
 *      A description of the return value goes here.
 *
 * @exception SomeException
 *      Description of when this exception is thrown.
 *
 * @see FirstRelatedClass
 * @see SecondRelatedClass#MethodName
 * @see SecondRelatedClass#MemberVariableName
 */
```

Variable Header

```
/**
 * <p>
 * Description of variable.
 *
 * @see RelatedClass
 */
```

[Engineering | Beamer | DataBeam]

javadoc - The Java API Documentation Generator

Generates HTML pages of API documentation from source files.

SYNOPSIS

```
javadoc [ options ] package | class1.java class2.java...
```

DESCRIPTION

javadoc parses the declarations and documentation comments in a set of Java source files and produces a set of HTML pages describing the public and protected classes, constructors, methods, and fields. In addition, it produces a list of classes, a class hierarchy and an index of all APIs. As an argument to **javadoc** you must pass in either a package name or a series of Java source files.

Commenting the Source Code

While **javadoc** automatically parses class, interface, methods and variable declarations, you can add further documentation via doc comments in the source code. These comments may include HTML tags. A doc comment consists of the characters between the `/**` that begins the comment and the `*/` that ends it. The text is divided into one or more lines. On each of these lines, leading * characters are ignored; for lines other than the first, blanks and tabs preceding the initial * characters are also discarded.

```
/**
 * This is a doc comment.
 */
```

The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the declared entity. This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tagline (as defined below).

Documentation comments are only recognized when placed immediately before class, constructor, method, or field declarations.

Standard HTML

You can embed standard HTML tags within a doc comment. You should not use heading tags such as <h1> and <h2>, or a horizontal rule <hr>, because **javadoc** creates an entire structured document and these structural tags interfere with the formatting of the generated document.

javadoc Tags

javadoc parses special tags that are recognized when they are embedded within a Java doc comment. These doc tags enable you to autogenerate a complete, well-formatted API from your source code. The tags start with an "at" sign (@).

Tags must start at the beginning of a line. Keep tags with the same name together within a doc comment. For example, put all @author tags together so that **javadoc** can tell where the list ends.

Class and Interface Documentation Tags

`@author` *name-text*

Creates an "Author" entry. The text has no special internal structure. A doc comment may contain multiple `@author` tags.

`@version` *version-text*

Adds a "Version" entry. The text has no special internal structure. A doc comment may contain at most one `@version` tag.

`@see` *classname*

Adds a hyperlinked "See Also" entry to the class. Some examples are:

```
@see java.lang.String
@see String
@see String#equals
@see java.lang.Object#wait(int)
@see Character#MAX_RADIX
@see <a href="spec.html">Java Spec</a>
```

The character # separates the name of a class from the name of one of its fields, methods, or constructors. One of several overloaded methods or constructors may be selected by including a parenthesized list of argument types after the method or constructor name. Whitespace in `@see`'s *classname* is significant. If there is more than one argument, there must be a single blank character between the arguments. For example:

```
@see java.io.File#File(java.io.File, java.lang.String)
```

A doc comment may contain more than one `@see` tag.

An example of a class comment:

```
/**
 * A class representing a window on the screen.
 * For example:
 * <pre>
 *    Window win = new Window(parent);
 *    win.show();
 * </pre>
 *
 * @author  Sami Shaio
 * @version %I%, %G%
 * @see     java.awt.BaseWindow
 * @see     java.awt.Button
 */
class Window extends BaseWindow {
   ...
}
```

Field Documentation Tags

The only documentation tag a field comment can contain is the `@see` tag (as described above).

An example of a variable comment:

```
/**
 * The X-coordinate of the window.
 *
 * @see window#1
 */
int x = 1263732;
```

Constructor and Method Documentation Tags

Can contain `@see` tags, as well as:

`@param` *parameter-name description*
: Adds a parameter to the "Parameters" section. The description may be continued on the next line.

`@return` *description*
: Adds a "Returns" section, which contains the description of the return value.

`@exception` *fully-qualified-class-name description*
: Adds a "Throws" section, which contains the name of the exception that may be thrown by the method. The exception is linked to its class documentation.

An example of a method doc comment:

```
/**
 * Returns the character at the specified index. An index
 * ranges from <tt>0</tt> to <tt>length() - 1</tt>.
 *
 * @param     index  the index of the desired character.
 * @return    the desired character.
 * @exception StringIndexOutOfRangeException
 *              if the index is not in the range <tt>0</tt>
 *              to <tt>length()&nbsp;-&nbsp;1</tt>.
 */
public char charAt(int index) {
   ...
}
```

OPTIONS

`-classpath` *path*
: Specifies the path javadoc uses to look up the `.java` files. Also specifies the directories from which **javadoc** is to load its own `.class` files. (Use **`-sourcepath`** to specify a source path separately.) Each *path* element should be a directory that contains the topmost packages. Overrides the default or the CLASSPATH

environment variable, if it is set. The *path* can contain multiple paths by separating them with a semi-colon. For example, the following sets two paths, the first of which is the current directory (.):

```
javadoc -classpath .;C:\opus\javasrc
```

-version
Include @version tags, which are omitted by default.

-author
Include @author tags, which are omitted by default.

-noindex
Omit the package index, which is produced by default.

-notree
Omit the class/interface hierarchy, which is produced by default.

-d *directory*
Specifies the destination directory where **javadoc** stores the generated HTML files. (The "d" means "destination.") The directory can be absolute or relative to the current working directory. For example, the following generates the documentation for the java.lang package (using CLASSPATH to find it) and stores the results in the directory specified by the `-d` option:

```
javadoc -d C:\opus\public_html\doc java.lang
```

-verbose
Without the `verbose` option, messages appear for loading the source files, generating the documentation (one message per source file), and sorting. The `verbose` option causes the printing of additional messages specifying the number of milliseconds to parse each java source file.

-sourcepath *path*
Specifies the search path for source files. Does not affect the loading of class files.

NOTE: The `-doctype` option is no longer available. Only HTML documentation can be produced.

EXAMPLES

```
javadoc -classpath C:\wise\src java.io
```

Generates HTML-formatted documentation for all classes, interfaces, methods and variables in the Java source files belonging to the `java.io` package located below the specified directory:

```
C:\wise\src
```

ENVIRONMENT

CLASSPATH

Provides the system a path to the user-defined classes to be processed by javadoc. Separate directories with a semi-colon, for example,

```
.;C:\avh\classes;C:\usr\java\classes
```

SEE ALSO

javac, java, jdb, javah, javap,