C. Learning to use Javadoc

C.1 Introduction to Javadoc

Javadoc is a tool that automates the production of documentation from Java source code. The documentation is placed into several files using one of several available file formats. A full description of Javadoc usage, for windows, is available at:

```
<http://java.sun.com
        /products/jdk/javadoc/writingdoccomments.html>
```

The Javadoc comments are always surrounded by "/**" and ended with "*/". For example:

```
/**
      This is a Javadoc comment
*/
```

C.2 The tags of JavaDoc

In this section we cover the following Javadoc tags:

```
@author, @deprecated , @exception , @param , @return , @see,
        @since , @throws , @version , {@link}.
```

These tags are covered, each in their own subsection. It is important to remember that many of these tags were introduced in Jdk 1.2 and that more tags will be introduce after new versions of the Jdk come out. The version number of the JDK under which the tag was release is shown next to the tag in parenthesis.

C.2.1. @author (since jdk 1.0)

The *author* tag is used to name the author of the code. Authors can be listed as a comma delimited list. For example:

```
/**
      @author Douglas Lyon, Joe Blow
*/
```

C.2.2. @deprecated (since jdk 1.0)

The *deprecated* tag is used to add a comment that the method or instance variable should not be used. Support for deprecated features can be withdrawn at any time.

```
/**
      @deprecated As of Kahindu 1.0, replaced by {@link
        show()}
*/
```

C.2.3. @exception (since jdk 1.0)

The *exception* tag is used to indicate that an exception may be thrown. The *exception* tag adds a *throws* subheading with the *class-name* and the *description* text. For example:

```
/**
      @exception OutOfMemoryException Thrown when too many
        windows are open.
*/
```

C.2.4. @param (since jdk 1.0)

The *param* tag is used to denote a passed in parameter with a description. While Java permits multiple instance variables in a single line, better practice would be to place the instance variables on different lines. For example:

```
/**
      @param g is a graphics context
      @param c is the color
*/
```

C.2.5. @return (since jdk 1.0)

The *return* tag denotes the type of the value returned by a method. For example

```
/**
      @return a copy of the frame {@link java.awt.Frame Frame}
*/
```

C.2.6. @see (since jdk 1.0, broken in 1.2, fixed in 1.2.2)

The *see* tag is used to introduce a link to a web page or to another part of the API. For example

```
/**
      @see java.awt.Frame
      @see java.awt.Frame#setVisible
      @see <a href="www.docjava.com"> DocJava, Inc.</a>
*/
```

C.2.7. @since (since jdk 1.1)

The *since* tag takes a description that indicates when the method or instance variable was introduced. It can also be used to indicate when a method was deprecated.

C.2.8. @throws (since jdk 1.2)

The *throws* tag is just like the *exception* tag. For example:

```
/**
      @param arg is the argument to a method
      @throws ArithmeticOverflowException if argument is zero.
*/
public void f(double arg) throws ArithmeticOverflowException
       {...}
```

C.2.9. @version (since jdk 1.0)

The *version* tag is used to indicate the version of the class where the method or member was introduced. For example:

```
/**
      @version 1.2
*/
```

C.2.10. {@link} (since jdk 1.2)

The *link* tag is used to introduce a hyper-text reference in the produced Javadoc code. The reference can be to a literal *href*, a class or a method. For example:

```
/**
      @deprecated as of version 0.9
        replace by {@link #setN(int)} and the
        {@link gui.AdaptiveLog#setVisible AdaptiveLog}
*/
```

Only the *AdaptiveLog* label is seen in the link. A reference is generated that will locate the *setVisible* method in the *AdaptiveLog* dialog.

The "#" sign is used to link to a specific method in a class.

C.3 Common HTML tags

The HTML tags used to formulate the JavaDoc comments are almost always authored without the aid of a WYSIWYG composition tool. As a result, most programmers need to know at least a few of the basic HTML tags, in order to make the API documentation more readable. Figure C.3-1 shows a summary of some of the more common HTML tags.

<A> Anchor	<HEAD> Head	<P> Paragraph
<ADDRESS> Address	<HTML> HTML	<PLAINTEXT> Plain Text
<B> Bold	<I> Italic	<PRE> Preformatted Text
<BASE> Base	<IMG> Inline Image	<SAMP> Sample
<BLOCKQUOTE> Block Quote	<INPUT> Form Input	<SELECT> Form Select
<BODY> Body	<KBD> Keyboard	<STRIKE> Strikethrough
  Line Break	<LI> List Item	<STRONG> Strong
<CITE> Citation	<LINK> Link	<TEXTAREA> Form Text Area
<CODE> Code	<LISTING> Listing	<TITLE> Title
<DIR> Directory List	<MENU> Menu List	<TT> Teletype
<DL> Definition List	<META> Meta	<U> Underlined
<EM> Emphasized	<OBJECT> Object	<UL> Unordered List
<FORM> Form	<OL> Ordered List	<VAR> Variable

Figure C.3-1. Summary of Common HTML Tags.

Structural tags (i.e., *h1, h2, head, etc.*) may interact with the structure of the JavaDoc generated HTML in unanticipated ways and should be used with care. As a result, they are not listed in Figure C.3-1.

The tags typically take the form of *<a> anchor text *. Often a hypertext reference (called an *href*) is embedded in the first anchor tag. For example:

```
<A HREF="backups/backups.html">Backup Services!</A>
```

For a complete list of tags in a variety of HTML versions, see <http://www.w3.org>.

C.4 How to Run JavaDoc in CodeWarrior

In order to run Javadoc in CodeWarrior, select the *Application Settings* in the edit menu, as shown in Figure C.4-1.

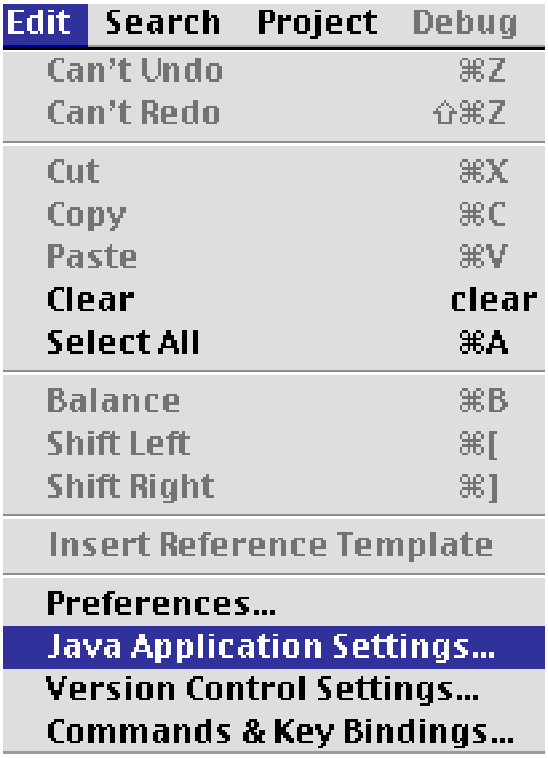

Figure C.4-1. Selecting the Application Settings.

Upon selection a pop-up dialog appears which enables to select the *Target Settings* as shown in Figure C.4-2.

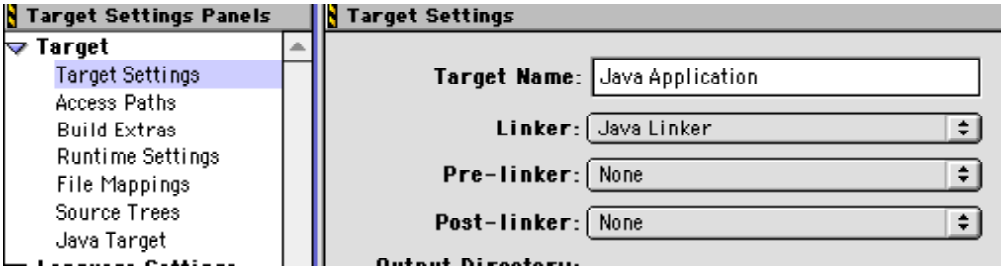

Figure C.4-2. Select the Target Settings.

Use the *Pre-linker* setting to select *JavaDoc*, as shown in Figure C.4-3.

Figure C.4-3. Select the JavaDoc Pre Linker

Now, when the program is compiled the JavaDoc output will be generated automatically.

C.5 Summary

The JavaDoc output is able to embed HTML code. Sorry to say, HTML is beyond the scope of this section and will have to be covered elsewhere. There are excellent primers on HTML on the web, and excellent tools available for the generation of HTML.

As of this writing, most HTML documents do not represent vector images or mathematics very well. This may change soon, however. Typically, documentation does not need such features, and as they are unsupported by most browsers, we may do well to avoid them.