

## **Javadocing in Netbeans (rev. 2011-05-20)**

This note describes how to embed HTML-style graphics within your Javadocs, if you are using Netbeans. Additionally, I provide a few hints for *package* level and *overview* level documentation and the role of the *properties* option within a Javadoc'ed project. This tutorial was tested with Netbeans 6.8. [rob rucker 2010-07-13].

For Netbeans 6.9.1, the modification of the build.xml file is no longer necessary so that part of the tutorial can be ignored. [2011-04-28]

### **Overview**

Displaying graphics is part of the general Javadoc documentation approach that supports 'literate programming'. Our text books often don't emphasize documentation except for end of line comments embedded within code using `'/'` or multiline comments using `'/* . . . */'`. While that is o.k. for developers maybe, clients don't want to have to read code to find out how the program works. So, to find out how the program works, a higher level of documentation is called for. We need some automated support and that's where *Javadoc* comes in. The word Javadoc is the general term used to describe the process of creating and displaying Java-based computer documentation. Creating the documentation depends on an executable program, *javadoc.exe*, that goes through all your package files, extracts out Javadoc comments, creates corresponding formatted HTML pages, hyper links them, and then automatically opens your browser to display them. This utility program is part of the standard Java distribution and is always available for you to use.

As you will see later in this tutorial, Javadoc not only goes through the *computer code* and extracts out distinguished comments, it also goes through *specialy named folders and files* and extracts additional text and graphics that are also displayed on the HTML pages.

### **Procedure for Netbeans 6.8 only**

(This glitch has been fixed in 6.9 so you don't need to modify the build.xml file but do need to do the other parts in this tutorial).

Currently, I don't know of a built-in way to embed graphics inside of Javadocs, so here is one way that does work: The problem is that the current versions of Netbeans don't automatically copy graphics from your source directory to the 'dist' directory where javadoc looks for data to insert into javadoc's HTML output. So, below is a way to do this by a small edit of the Ant build file (*build.xml*).

Below is a File view (not a Project view) of my project. (To get a File view, go to the main menu->Files).

The example project, **IT307Ch3DeitelGradeBook**, presented here, is (edited) code taken from the Deitel text chapter 3, which is being used for IT 307 and IT 408 during the 2010 sessions at WIU.

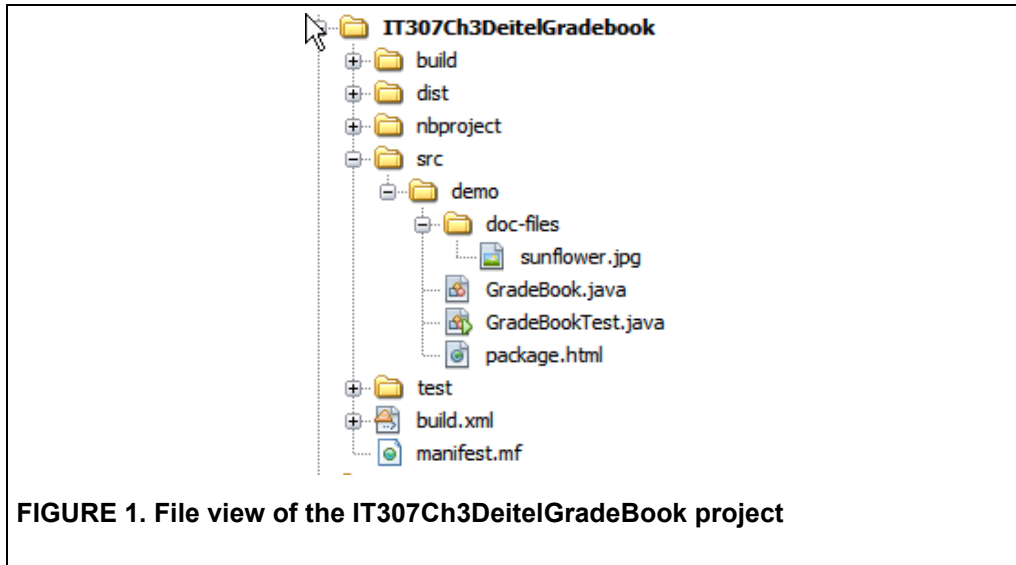
### **Cut to the Chase for embedding graphics (a quick overview for all versions of Netbeans)**

Within the project's package folder 'demo', I created the doc-files folder (a distinguished

name you must use) and copied in a graphic, *sunflower.jpg*. Then, in the *package.html* (a distinguished name for package level documentation) I inserted an `<img>` callout for the graphic *sunflower.jpg*. After a ‘Clean and Build’ I ran the Javadoc program and produced the browser displayed documentation. All these files are shown below.

### End Cut to the Chase

Here is a File view of the overall project.



## Detailed Steps to Embed and Display Graphics within your JavaDocs

### Copy graphics files into your project ( all NB versions)

Go to your project, then your package (my package name is 'demo').

1. create a new empty FOLDER inside your package. You must name it *doc-files*. to create such a folder, right click on your package name -> new -> other -> folder
2. Copy your graphics into that folder. For example, I have put *sunflower.jpg* in my doc-files folder. Actually, you can put anything you want in there since a (relative) hyper-text link will retrieve it. I would also recommend placing a UML class diagram in the folder as well, if you are able to create one.

### Edit the Build File ( for NB 6.8 only)

Now go to the File view in your project, find the *build.xml* file, then right click ->open. This will open the xml file in the editor panel of Netbeans.

Right at the bottom of the file, immediately before the ending `</project>` tag, insert the following code. Note: use *your* package name in place of my ‘demo’ package name if yours differs.

```
<target name="-pre-compile">
    <copy todir="./dist/javadoc/demo/doc-files">
        <fileset dir="./src/demo/doc-files"/>
    </copy>
</target>
```

## Javadocing in Netbeans (rev. 2011-05-20)

The effect of this Ant command is to copy the content of *doc-files* to the distribution folder ('dist'). This is where javadoc looks for included files and now they will be there.

Save the *build.xml* file.

### Insert callouts in your HTML documents. ( all NB versions)

In your *package.html* file or in any of your source files *javadoc* sections, insert the following standard HTML code to access your doc-file graphics content.

Below is the *package.html* special package level documentation file that documents my *demo* package and the *GradeBook* suite of classes.

to create this HTML file, right click on your package name and navigate to find HTML File. Click that.

```
<!-- package.html  rob r 2010-07-13 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head> <title>GradeBook </title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <p>This package holds (improved!) GradeBook related
      files from the Deitel text (8th edition). </p>
    
    <h4>Description of the package 'demo'</h4>
    <p>This package, introduces the student to
      basic Java programming coding conventions. The
      Java classes shown improve on the Deitel conventions by
      using javadoc and modern code formats.
      The files are as follows:</p>
    <ul>
      <li>GradeBook.java - holds the details of a course.</li>
      <li>GradeBookTest - is a test driver program that invokes instances of
        gradeBook type (Java) classes. </li>
    </ul></body></html>
```

FIGURE 2. *package.html* package level documentation file

### Do a 'Clean and Build' to establish new linkages.

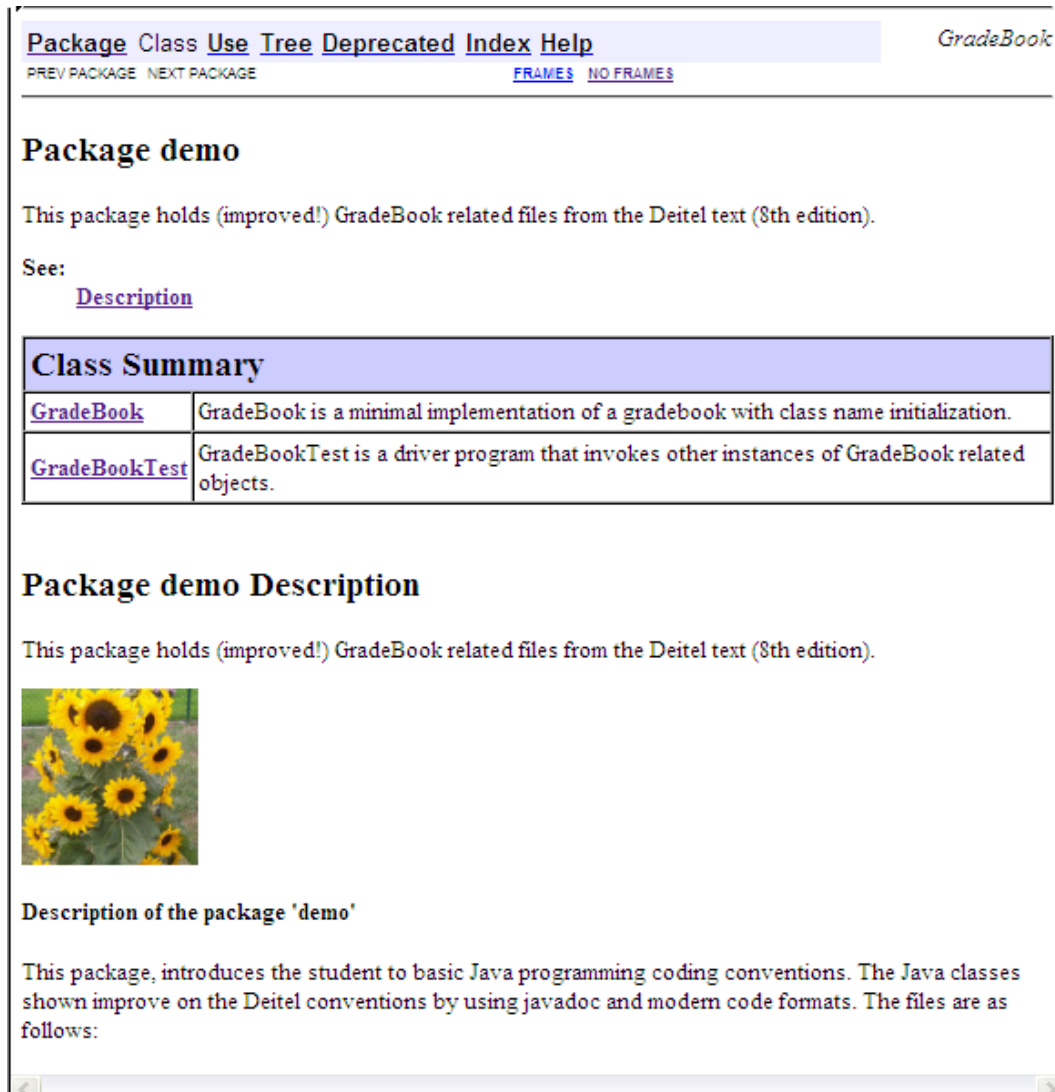
Doing a Clean and Build is a good idea in general after you make a few code changes.

### Run -> Generate Javadoc

This invokes the javadoc.exe executable that is in your jdk 1.6 distribution *bin* directory.

## Javadoc output

Running javadoc does a compile and then composes linked HTML files based on what is in your javadoc comments. Then it automatically calls your browser (check your bottom toolbar of programs since your browser icon may only show up there).



The screenshot shows a Javadoc HTML page for a package named 'demo'. At the top, there is a navigation bar with links for 'Package', 'Class', 'Use', 'Tree', 'Deprecated', 'Index', and 'Help'. The title 'GradeBook' is displayed in the top right corner. Below the navigation bar, the package name 'Package demo' is shown in a large, bold font. A paragraph of text describes the package: 'This package holds (improved!) GradeBook related files from the Deitel text (8th edition)'. Below this, there is a 'See:' section with a link to 'Description'. A 'Class Summary' table follows, listing two classes: 'GradeBook' and 'GradeBookTest'. The 'GradeBook' class is described as a minimal implementation of a gradebook with class name initialization. The 'GradeBookTest' class is described as a driver program that invokes other instances of GradeBook related objects. Below the table, the section 'Package demo Description' is shown, followed by the same descriptive paragraph as above. An image of sunflowers is displayed below the description. At the bottom, there is a section titled 'Description of the package 'demo'' with a paragraph of text: 'This package, introduces the student to basic Java programming coding conventions. The Java classes shown improve on the Deitel conventions by using javadoc and modern code formats. The files are as follows:'. The page ends with a scrollbar.

Class Summary	
<a href="#">GradeBook</a>	GradeBook is a minimal implementation of a gradebook with class name initialization.
<a href="#">GradeBookTest</a>	GradeBookTest is a driver program that invokes other instances of GradeBook related objects.

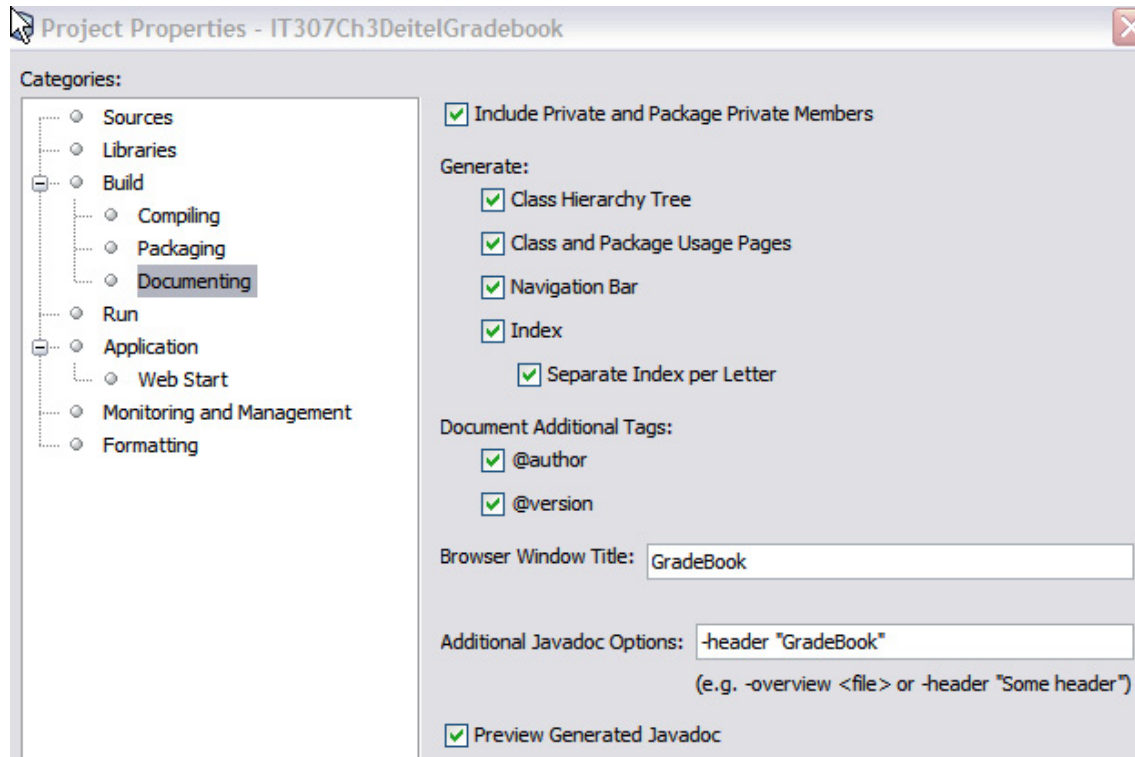
### Configuring your project to display private variables, titles and headers.

By default, javadoc will not show private variables or some of the @ parameters. So, do the following.

Right click on your project name and scroll to the bottom of the options and choose *properties*. (see dialog box below)

Then click on *Documenting* and check everything, as well as entering title and header text.

As an aside, clicking on the *Run* option allows you to enter command line arguments that are picked up in the *String[] args* array from the *main()* method.



## Documenting Individual Packages

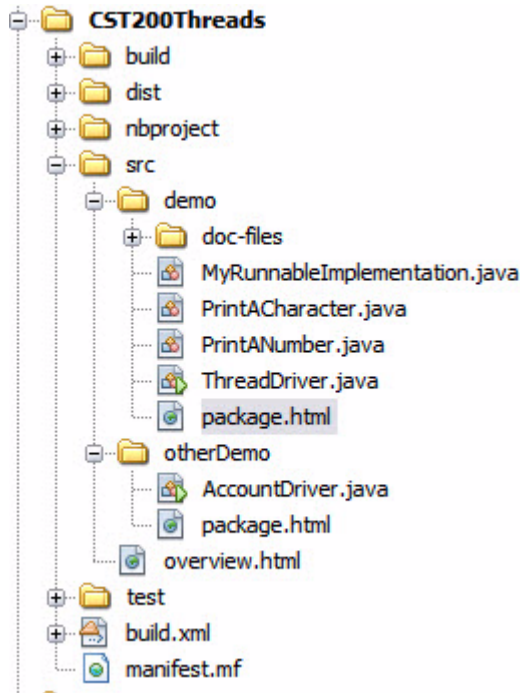
To document a *package level collection of files*, say a package named ‘demo’, as in the above example code, you place a specially named HTML file, *package.html*, in the package folder. That *package.html* file contains text that will document the classes in the ‘demo’ package. If you want to produce graphics for that package, then you would create a *doc-files* folder and place your graphics in there. Callouts for graphics use the standard HTML tags, for example, here is callout from a Javadoc section of comments that will cause the browser to get the jpg file referenced and insert it into the HTML page.

```
<img src = "doc-files/demoUMLDiagram.jpg" alt = "UML diagram"/>
```

If you have a second package, named say, ‘otherDemo’, you would create *another package.html* file in that package folder that documents the classes in *that* package. For displaying graphics associated with the ‘production’ package, create another *doc-files* inside the package folder and place graphic file there. (See “File view of the CST200Threads Project” on page 6).

## Documenting Multiple Packages, the Overview of the Suite

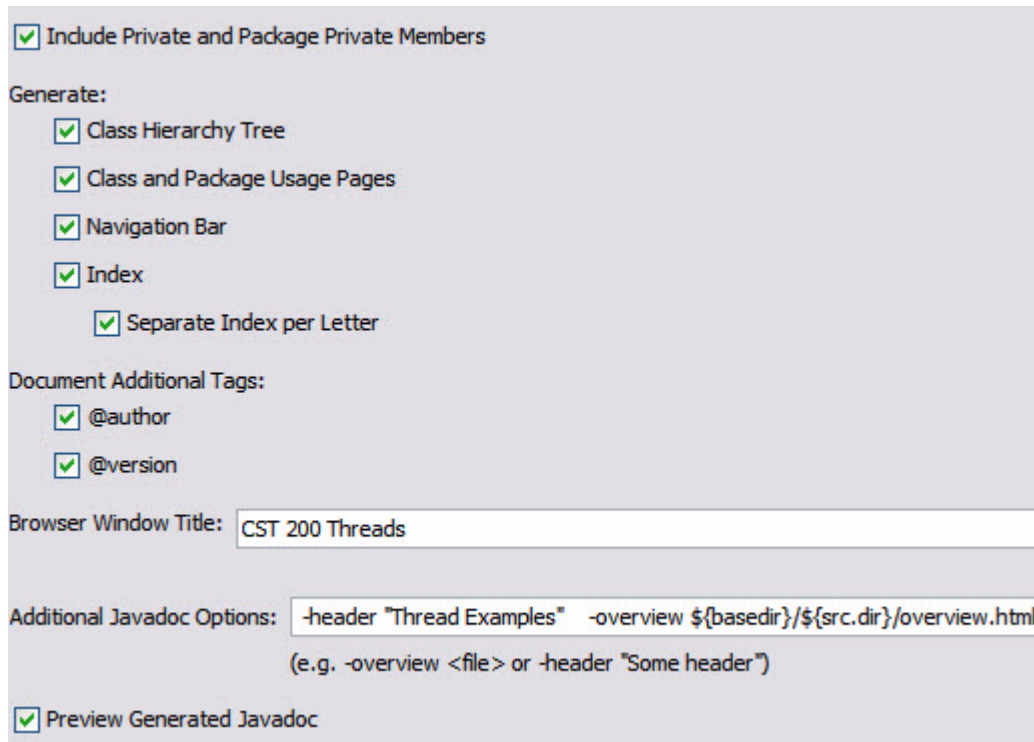
Things get more interesting when you need to document multiple packages that represent the *overall suite*. Now you want to document individual packages using a *package.html* file in each package folder as well as, an *overall* descriptive HTML file for the *collection of packages*. To create this overall documentation, create a specially named HTML file called *overview.html* and place it in the <src> folder, as shown in the diagram below. ( You need to be in File view to see these files). <src> is the folder that holds all your packages.



**FIGURE 3.** File view of the CST200Threads Project

### **Configuring Javadoc to Recognize the Location of overview.html**

Go to your project properties dialog box and add in the following within the Additional Javadoc Options.



Configuring Javadoc to override the built-in CSS file *javadoc.css*

In the additional Javadoc Options as above, include the following (notice the placement of the ‘periods’):

```
-stylesheetfile ${basedir}/${src.dir}/style.css
```

And, I have named my css file *style.css*.

Place this file in your <src> folder.

Additionally, in my package and overview html files, I include a

```
<link href="style.css" type = "text/css", rel = "stylesheet" />
```

## Templates for Your Java Classes

Under Tools>Templates>Java>*Java Class*

Go to *edit* and replace the contents with the first section of code shown below.

Then go to *Java Main Class Template* and replace its contents with the respective code below

Same for Interface Template

Java Templates

These templates will replace the ones in your Netbeans Tools Templates files.

## *Javadocing in Netbeans (rev. 2011-05-20)*

r.r 2011-07-25

\*\*\*\*\*Java Class Template

```
/* ${package}.${name} by ${user} on ${date} */
<#if package?? && package != "">
package ${package};
</#if>
/**${name}  shows ?? .
 *
 * @author ${user}
 * @version 1. 0  ${date}
 * @since jdk 1.6 upd 21
 * @see ""
 */
public class ${name}
{

} //end ${name}
*****END Java Class Template
```

\*\*\*\*\*Java Main Class Template

```
/* ${package}.${name} by ${user} on ${date} */
<#if package?? && package != "">
package ${package};
</#if>
/**${name}  shows ?? .
 * <p>
 * </p>
 * @author ${user}
 * @version 1. 0  ${date}
 * @since jdk 1.6 upd 21
 * @see ""
 */
public class ${name}
{
    public static void main(String[] args)
    {

    } //end main()
}
```



