Freie Universität Berlin

Working Group Software Engineering

Winter Semester 2006

WikiJavadoc - A collaborative documentation system for Java

Jing Zhao

zhao@inf.fu-berlin.de

Supervisor: Christopher Oezbek, Prof. Dr. Lutz Prechelt

09.11.2006

Abstract

Over the past twenty years, Free and Open Source Software (F/OSS) has risen to great prominence. Studies have shown in many cases, F/OSS programs provide a reasonable or even a superior approach compared to their proprietary competition.

Unfortunately, documentation is one of the weakest aspects of open source programs. Updating the documentation is sometimes an afterthought. And the maintenance of the documentation becomes difficult due to the nature of F/OSS. This paper describes an attempt to merge a JavaDoc-style in-line documentation system with the online collaboration paradigm of WikiWiki. ~~This work is focused on developing an open source inline documentation tool enhanced with Wiki functionality.~~ With this documentation system users can not only read but also edit the JavaDoc HTML pages. And these modifications can be rewritten in the source code. The aim of this work is to enable users to improve their Java documentation without too much effort. ~~This paper will also discuss programming details of WikiJavadoc.~~

Eidesstattliche Erklaerung

Hiermit versichere ich, die vorliegende Arbeit selbstandig und unter ausschliessicher Verwendung der angegebenen Literatur erstellt zu haben.

Die Arbeit wurde im November 2006 fertiggestellt und kommt hier erstmalig zur Vorlage.

Contents

List of Figures

1 Introduction

1.1 Free and Open Source Software and Documentations

Free and Open Source Software (F/OSS) refers to software whose licenses permit users to run program for any purpose; to study and modify the program freely; and to redistribute copies of either the original or modified programs, without having to pay royalties to previous developers. F/OSS has been building momentum for over twenty years, and is breaking into commercial world. Some F/OSS programs remain non-commercial, but more and more are designed for profit. Many studies have shown that, in many cases, F/OSS programs provide a reasonable or even a superior approach compared to their proprietary competition. As evidenced by the number of stable and robust F/OSS programs, including Linux, Apache, Mozilla, MySQL, Perl, Python, F/OSS has been extremely successful on the technological level. Unfortunately, documentation is one of the weakest aspects of F/OSS. F/OSS are usually documented in the following forms : as an introductory text on the home page, in the FAQ(Frequently Asked Questions), use scenarios, case studies, readme file, user's guides, or API documentation.

When a programmer wants to develop a new software, and needs some F/OOS tools. Then how can he find the right tool? It is not easy when one just rely on the information in the web, because most information in the web is mainly limited to introductory text describing for first-time-users. That information primarily addresses just simple questions, like "What is the tool?", and "What function can the tool perform?". But ~~the~~ quality documentation, such as that provided by API documentation and user's guides, tutorials are often missing. To some degree, the quality documentation is perhaps more of a programming tool than a strict text. And thus that proves more helpful for software developers.

Jack Herrington[1] finds that better documentation directly improves adoption rates of F/OSS . In his study about the state of open source introductory documentation, he took the top 20 entries in Sourceforge.net[2] and analyzed their documentation. He found that if documentation was available, it was centrally located and easy to find. That means that the efficiency and quality documentation affects the cost, quality and reusability of software. In his study, all of the 20 project samples were documented to some degree. After assessing the quality of the introductory documentation, ~~Jack Herrington~~ has concluded that there is still a lot of room for improvement.

All of the sites in the study included some brief description of what function the tool performed. "Only one in every 10 projects had a clear and concise statement of what problem the tool was meant to solve"[Her03]. Almost all projects provided a statement of the function of the tool, but only one of the 20 projects that included list of graphics to illustrated the function.

Only half of the projects had FAQs. Of those, less than half again addressed the simple question of hat the tool was meant to do. Half of the projects had some type of tutorial or sample code. Of those, only a third had documentation that provided "scenarios for when the tool was useful and in what configuration"[Her03]. Information about the hardware or system requirements for the software was given by Only 15 percent of the projects.

Of course, updating the documentation is often merely an afterthought. Because there are so many developers who work according to the motto "release early, release often", open source software can change very frequently. Thus, due to the very nature of F/OSS, the maintenance of these documents remains difficult. However, since F/OSS are becoming more and more popular, quality documentation and API documentation should be provided to users in light of its increasing importance.

[1]~~Jack Herrington is the author of the paper, Is documentation holding open source back?~~

[2]SourceForge.net is the world's largest development and download repository of Open Source code and applications.

1.2 JavaDoc + Wiki = Wiki-collaborative construction of documentation for open source software

There are many documentation types for F/OSS according to the program types. ~~Java is a sophisticated and popular programming language.~~ Many F/OSS are written in Java, so the Java documentation frameworks are widely used, e.g. Javadoc, and gjdoc.

Javadoc is a Java documentation framework from Sun Microsystems. It is a tool for generating API documentation from Java source code into HTML format. Javadoc is the industry standard for documenting Java classes. In fact, most IDEs[3], such as the Eclipse IDE, NetBeans or Microsoft Visual Studio, can generate Javadoc HTML automatically. Another Java documentation framework is gjdoc. It is a Javadoc replacement from GNU Classpath[4]. In another words, just as Javadoc is included in Java SDK, so too is gjdoc part of GNU Class. Gjdoc wii be discussed in this paper in more detail. Because WikiJavadoc is an extension to gjdoc.

Gjdoc documentation is similar to Javadoc documentation. They are both hypertext documentation listing available software components at class level along with a series of navigational indices[Ber00]. Hyperlinks are used to cross-reference among class documents via parameters, types, return values, written comments, and so on. Erik Berglund finds that Javadoc has become somewhat of a model for online software reference documentation. Javadoc can also be viewed as a source-code browser, providing the reader with what is assumed to be a correct typeset and relevant extract of Java source-code files (including written comments)[Ber00].

Wiki is an effective tool for collaborative authoring, since it allows visitors to easily add, remove, edit and change some available content, sometimes without the need for registration. It is increasingly common to use some type of Wiki as a platform for de-

[3]IDE is an abbreviation of Integrated Development Environment

[4]GNU Classpath is a project aiming to create a free implementation of the standard class library for Java.

veloping F/OSS documentation. This is due to the fact that Wikis have more flexibility in terms of layout design. More importantly, open source projects depend on community involvement in the development, which applies to both the application and the documentation.

As mentioned above, the maintenance of the Java documentation is difficult. Javadoc ~~is a very strong framework, however, it~~ only delivers static text that does not change. To facilitate a better Java documentation, a Javadoc-style in-line documentation system with the online collaboration paradigm of WikiWiki named WikiJavaDoc is proposed.

The idea is that WikiJavadoc looks and feels like a normal Javadoc page, but it has an edit link at the bottom of every class or class members' Javadoc comment. Users can edit each Javadoc comment on a Javadoc site using the wiki command "edit" without the need for registration. Changes that the user makes to the comment are directly reflected both on the HTML page and in the underlying source code.

WikiJavadoc is a freely available and modifiable Java documentation
tool, a Javadoc enhancement. It extends the standard Javadoc with Wiki functionality. WikiJavadoc users benefit from a variety of features:

- Javadoc collaboration: WikiJavadoc makes a normal Javadoc become a collaborative Java document. For a Java project, non-developer users can also document it. WikiJavadoc is effective to put together collaborative Javadoc-style comments in a short time based on its Wiki functionality.

- Linking HTML source Page: WikiJavadoc provides also HTML source code pages. That makes the non-developer users can write comments better according to HTML source pages.

- Writing back modifications to source: Users can reflect the changes from the Javadoc page back into the source code directly. That makes contributing to favorite APIs much easier.

- Users collaboration: Users can be asked to collaborate as well. For instance, if a user can send a question to others via a mailing list, others may contribute their solutions to the WikiJavadoc documentation.

1.3 Usage Scenarios

The WikiJavadoc was written with the following scenario in mind:

For a small scale F/OSS project, a library is written in Java and documented using Javadoc comment tags. F/OSS projects implementation is always more important than the quality of the documentation. So large amounts of code are often undocumented, as a result of the personal preference of some developers. In many cases, some comments can be insufficient, outdated, incorrect, or even misleading. Nonetheless, the popularity of the library has been steadily increasing. Imagine another Java developer downloads the library from the Internet and wants to learn how to use it. In this typical scenario the developer receives the jar file containing the binaries (and any other binaries needed to use that jar file) and the automatically generated Javadoc documentation. However, this documentation is can insufficient and might be outdated. The project experienced an increase in the number of questions asked about the use of the library on the mailing list.

The library developers clearly feel the need to provide better, richer, and more accurate documentation. But they are too deeply time-committed to bug-hunting and expanding features for the next release. The developer generates the Javadoc documentation with WikiJavadoc and then enriches it and distributes the rich documentation with the library.

Generally they have to start an IDE and then document every Javadoc comment of every file, even though there are thousands of Java files to be documented. Instead of using that normal way, they can use WikiJavadoc framework, which will facilitate the document work.

They can install a web container Tomcat quickly and copy pre-built WAR[5] file to the dictionary ”webapps” with minimum configuration. ~~That war file is deployed and~~ a WikiJavadoc project is created by giving the location of the current source code. An administrator can use standard version control tools to check out the source files from CVS repository to the WikiJavadoc version, then the administrator can create WikiJavadoc project. The source code is copied to another dictionary and the API is created as well. In addition, the anonymous users can browse APIs, source files, edit Javadoc comments, and save changes to the Javadoc Version in WikiJavadoc Server. These changes are not directly fed back into the source code of the project but kept in a separate source code in WikiJavadoc system. At last the administer can commit the WikiJavadoc version to CVS repository if the new version are found to be appropriate. Of course administrator can also execute other version control operations, or update and delete WikiJavadoc projects. Furthermore, a web interface simplifies the document work.

[5] WAR is abbreviation for Web ARchive, a WAR file is a ZIP file used to distribute a set of Java classes.

2 Related Work

2.1 JavadocOnline

JavadocOnline [6]is a search engine project based on the web service Google Web API. In another words, JavadocOnline searches for the classes in Google, building better automatic queries, in order to show only filtered Javadoc webs.[JAV06a] Users can find Java API documentation from the JavadocOnline homepage, but he can neither edit documentation, nor overwrite the source code.

2.2 KickJava

KickJava [7]has introduced a Java Source Codes Repository. With this the user can browse Java APIs, and search Java source code and code examples. KickJava provides just original API documentation, and like JavadocOnline it does not allow users to edit or add further information to documentation and source code.

2.3 JDocs

JDocs [8]collects the online APIs for many popular Java tools and libraries, totalling 205 APIs as of this writing. The APIs for those popular tools are loaded in one db-driven system. JDocs, which has a familiar "Javadocs"interface, also allows the users to write comments for individual JavaDoc pages. Because the information in JDocs is indexed, users can search the information in the documentation. Because each registered users can provide additional information by annotating the JavaDoc APIs and share their knowledge by adding useful hints and information, Jdocs has potential to become a

[6]www.javadoconline.com
[7]www.kickjava.com
[8]www.jdocs.com/

very helpful resource. Although the annotating facilities of JDocs are promising, Jdocs differs from WikiJavadoc in other aspects, as it does not link HTMl source page, or allow users to rewrite modifications to source files.

2.4 JSourcery

JSourcery [9]is a web site like JDocs, and has thus far collected 100 open source Libraries. It provides the Javadoc documentation, which link HTML pages with syntax-highlighted source code for each class. Further, the source code HTML page contains embedded link so that its possible to trace program execution by following these links. JSourcery can also generate source code references in Javadocs and all libraries are searchable. It does not, however, provide functions for editing Javadocs and rewriting source files.

2.5 Docenhancer

Docenhancer [10]from IBM adds further semantic information to the Java documentation. That kind of information is gathered by statically analyzing the corresponding Java class files, for example, Call-graph information or Effect information. Docenhancer is not an online framework. It needs Javadoc API and the associated library as input to generate a new version eJavadocs. An overview of Docenhancer is illustrated in Fig.1.

[9]www.Jsourcery.com
[10]www.docenhancer.com

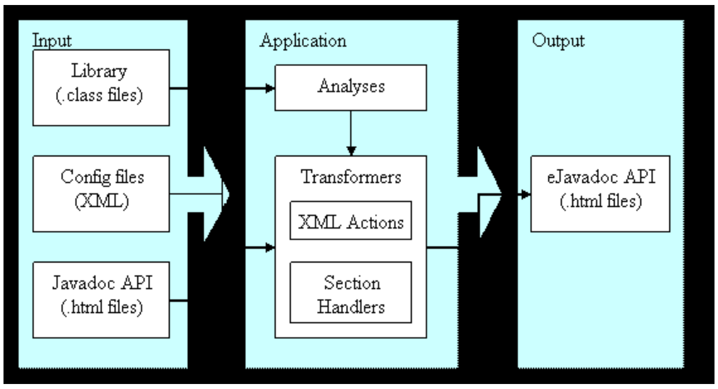


Figure 1: Overview of Docenhancer [DOC05]

2.6 Faq-o-matic

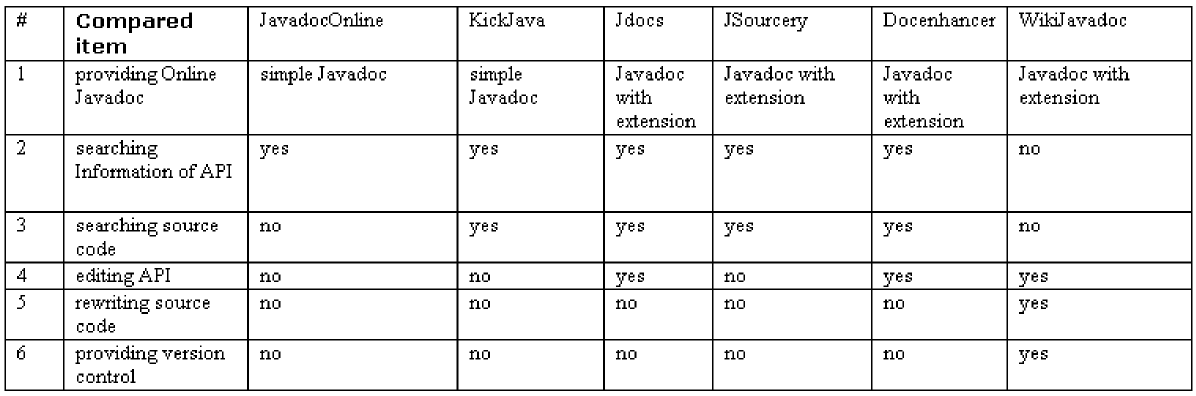
Faq-o-matic [11]provides a good example of an approach that is more involved with content-management than a pure wiki.

2.7 Conclusion and Comparison

All of the frameworks discussed abovee, except Faq-o-matic, are intended for Java developers. They all provide users standard Java API documentation or Documentation with extended information about libraries and applications. Some provide additional services. for example, some allow users to search documentation information, and others can show examples of the source codes.

Note. This comparison cannot be considered as of final, since most of the systems are still being developed. Last update was made for versions 10. 2006.

[11]www.faq-o-matic.com



#	**Compared item**	JavadocOnline	KickJava	Jdocs	JSourcery	Docenhancer	WikiJavadoc
1	providing Online Javadoc	simple Javadoc	simple Javadoc	Javadoc with extension	Javadoc with extension	Javadoc with extension	Javadoc with extension
2	searching Information of API	yes	yes	yes	yes	yes	no
3	searching source code	no	yes	yes	yes	yes	no
4	editing API	no	no	yes	no	yes	yes
5	rewriting source code	no	no	no	no	no	yes
6	providing version control	no	no	no	no	no	yes

Figure 2: A comparison of various Java documentation tools

The above comparison illustrates the innovative aspects of WikiJavadoc. To summarize, WikiJavadoc provides Java developers with different functionalities from the existing Java documentation framework, and focuses on editing Java API Documentation and rewriting source code collaboratively.

3 Requirements

According to the usage Scenario discussed above, all the functional and non-functional requirements for the system will be explained in this section. This section serves as a guideline for detail design, implementation and testing of the software.

3.1 Authentication

In WikiJavadoc system there are two types of users, authorized users and anonymous users. Only the authorized users are permitted to manage WikiJavadoc projects, e.g. creating, updating, or deleting projects. The anonymous users can only edit the existing APIs. Therefore, authentication mechanisms should be used to protect the project management resources.

3.2 Compatible look

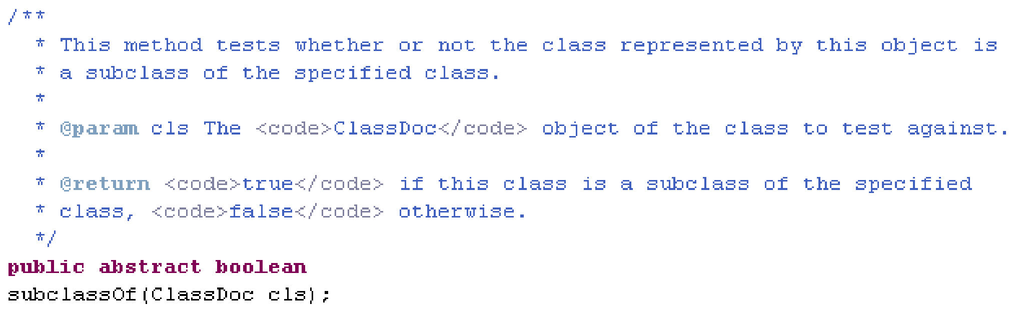
WikiJavaDoc should have a similar JavaDoc Interface, since most Java developers are already familiar with JavaDoc. However, it should have an edit link at the bottom of every class or class member. So that users can click on it to edit Javadoc documentation.

3.3 Wiki-Capabilities

Wiki should allow users to edit individual comments, and to add further contents to the Javadoc of that program entity. WikiJavadoc is able to reflect the modifications from the Javadoc page back into the source code directly. The updating of source code should be done without changing the general format of the document.

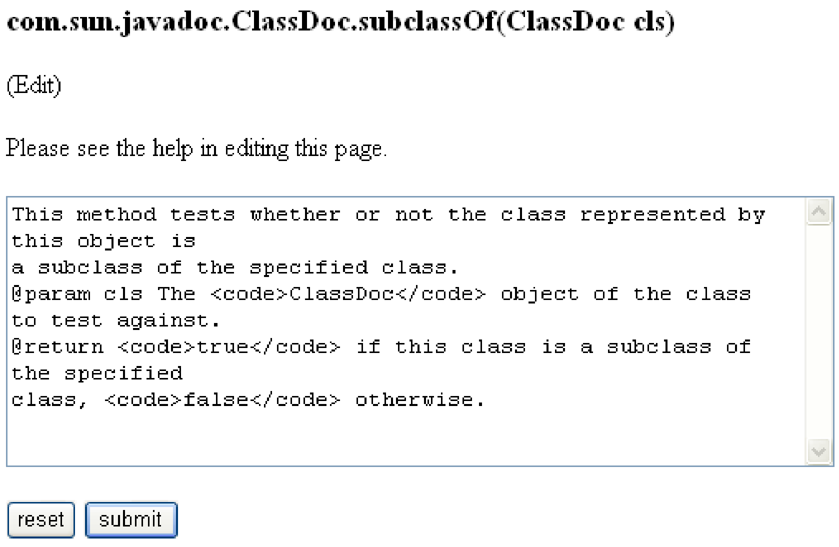
For generating proper Javadoc source code and HTML files, users must write the comments according to the specified rules. Those rules are fully compatible with the standard Javadoc rules used for the same problem. Fig.3 and Fig.4 are examples of

Javadoc comment parts in a Java source file and the corresponding comments in user interface. With the specified rules, the comments can be extracted automatically and properly. These rules will be explained in Appendix of this Paper.



```
/**
  * This method tests whether or not the class represented by this object is
  * a subclass of the specified class.
  *
  * @param cls The <code>ClassDoc</code> object of the class to test against.
  *
  * @return <code>true</code> if this class is a subclass of the specified
  * class, <code>false</code> otherwise.
  */
public abstract boolean
subclassOf(ClassDoc cls);
```

Figure 3: Javadoc comment in source code



com.sun.javadoc.ClassDoc.subclassOf(ClassDoc cls)

(Edit)

Please see the help in editing this page.

```
This method tests whether or not the class represented by
this object is
a subclass of the specified class.
@param cls The <code>ClassDoc</code> object of the class
to test against.
@return <code>true</code> if this class is a subclass of
the specified
class, <code>false</code> otherwise.
```

Figure 4: Javadoc comment in edit window

3.4 Synchronization of the source code

The system should be able to adjust API documentation to the Java files. Changes in the source code should be integrated into API documentation, and changes in API documentation should be able to be written back into the source code meaningfully .

3.5 Abilities of loading source code

The WikiJavadoc system created Java API documentation from Javadoc comments in source code, so it should be able to check out the sources from CVS or Subversion repository. After the source code is modified, it should be able to commit source code to CVS and Subversion repository.

3.6 Link source code

Sometimes a user wants to edit Javadoc documentation, but doesn't read the source code, it can be difficult to write comments. WikiJavadoc should be able to generate a html page with source code for each class. When the user edit some program entity's Javadoc, if he needs, he just clicks on the name of programming entity in the detail section of a class documentation page to display the page of the source code. The user can then refer to source code and notify better Javadoc comments.

3.7 Abilities of Logging

Modifications by users should be saved in a log file, and it will be accessible to the authorized users if needed.

3.8 Performance Requirements

The major performance requirements for WikiJavadoc:

1. The operation of user interface should not lead to human noticeable time delay beyond the usual HTTP performance.
2. Modifications should be implemented into the WikiJavaDoc server directly and become applicable for other operations.
3. Another important aspect is that the information should be fed back to users while operating the programm. For example, information, such as, the user-requested operation progress should be sent to the user when the operation lasts more than 5 seconds.
4. Concurrent processing capability. A well-developed WikiJavadoc system on a sever should be able to support at least 20 concurrent viewers plus 5 concurrent editors. This function consumes a lot of memory, however. In order to reduce the consumption of memory, the aim is to run the entire system on the visual-machine memory with less than 128 MB (not a restrictive limit).

3.9 Robustness

The system should run stably, and be able to treat errors meaningfully.

3.10 Notification Service

Users who are interested in WikiJavaDoc platform major changes can be notified, when changes occur.

3.11 Installation Requirements

In order to make the WikiJavadoc system widely used in F/OSS projects, the installation should have no hurdles and be executed as quickly as possible, ideally within a few minutes. The purpose is to minimize the installation time and the other computer resources by programming the WikiJavadoc software efficiently. For comparison, the other required components like JDK, Tomcat, can be installed within 30 minutes. In order to ease the burden of users, the number of those kind of add-on components should be less than 6.

3.12 Optional Requirements

Ideas that could improve this work with Wiki function, or in other Wiki software work satisfactorily, should be collected, although this is not mandatory. For example, a mailing-list or a forum could help resolve disputes arising from a particular. The RSS[12]-Feed function could be added to the Wikijavadoc system to inform users about new contributions and changes.

[12]Really Simple Syndication

4 Architecture Concept

4.1 Design Goals

The goal of this architecture model is to support software re-use for the sub systems wherever possible. This model will support integration of the modules and also parallel development of every components.

4.2 System Decomposition

The architecture model of the WikiJavadoc tool consists of a few components.

The first contains the security portal, the second contains project management, which contains create, update, and delete projects. The third contains the UpdateJavadoc tool, which manipulate source files. The fourth is edit window, which is a Equinox Incubator plug-in. The last one is version control component.

4.2.1 Security Portal

The security portal provides authentication and authorization for individuals of the two user-groups, namely users and members. An anonymous user can edit Javadoc comments of one WikiJavadoc project. If the user has been authenticated, he can also access protected project management resources, for example, create, update or delete project; otherwise, the user will be asked for a username and password. If the name and password cannot be authenticated, an error page will be displayed and the user will have an opportunity to enter a new username and password.

4.2.2 Projects Management

Only the authorized users can create, update or delete a project. To create a project it is necessary to do two steps: In a first step the Java source files must be loaded. In a second step the JavaDoc HTML pages, including HTML source code pages, are created from loaded the Java source files. The created project can be updated or deleted. To update an existed project the source file will be reloaded, then Java documentation is created. To delete a project all APIs including their sub directories should be deleted.

4.2.3 UpdateJavadoc tool

UpdateJavadoc tool receives changes by users, UpdateJavadoc tool parses the underlying source files provided by the version control layer, the new version of source code is generated from the changes in Javadoc. It is identical, including indentation and whitespace, to the original version in all parts but where the changes were applied.

4.2.4 Edit Window

When a user browse the Javadoc documentation, he click on a edit link, a new window will be displayed. After the user write some comments, click on "Submt" button. Then the modifications will be written into underlying Java source files, edit window component can call the UpdateJavadoc tool, additional, the edited HTML page should be updated, and the associated HTML source page should be updated. EDIT Window component answers for these work.

4.2.5 Version Control

The version control component should support all version control operations, such as, check out, commit, update, ect. And the version control component provides the source files to Project management component.

4.3 System Overview

A web-based portal with authentication and authorization mechanisms controls the provision and use of WikiJavadoc. All the compositions have high cohesion and less coupling. Local Error Handling will be used.

Fig. 5 illustrates the compositions of the WikiJavadoc and the corresponding technologies for implementation. First, users need to pass the portal of authentication and authorization before be allowed to manage projects. Form-based authentication is selected for security setting in this project. Only the authenticated users can check out the project's source codes to the local disk with version control tool, such as, Tortoise SVN. Then the authenticated users can create, update or even delete a project. Ant scripts and extended gjdoc is used to implement project management. Anonymous users can browse Javadoc and source code HTML pages, or edit Javadoc comments without the need for registration. When Javadoc comment of the documentation is edited by the user, the edit window will be displayed. The Implementation of the edit window uses Equinox Incubator technology. The modifications by users are rewritten into source files via UpdateJavadoc tool, which is a headless Eclipse plug-in. The edited Javadoc HTML page and the associated source HTML page are updated via the method in the extended gjdoc. WikiJavadoc requites version control function, including CVS and SVN. The authorized users can execute all the CVS operations, e.g. they can check out projects from the CVS repository, after the users modified the Javadoc comments, commit, or update to repository. All details will be discussed in the next two sections.

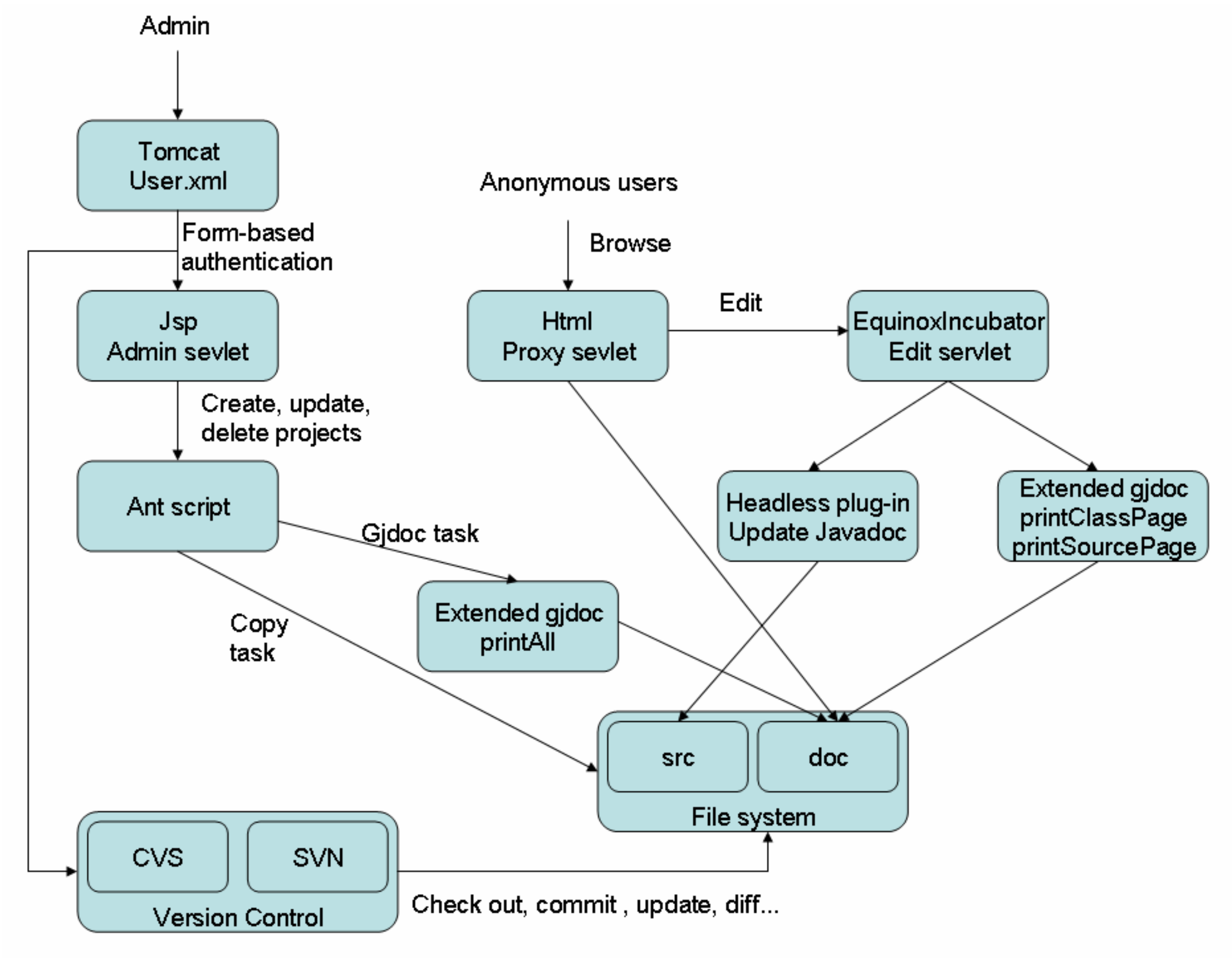

Admin
Tomcat User.xml
Form-based authentication
Jsp Admin sevlet
Create, update, delete projects
Ant script
Gjdoc task
Copy task
Extended gjdoc printAll
Anonymous users
Browse
Html Proxy sevlet
Edit
EquinoxIncubator Edit servlet
Headless plug-in Update Javadoc
Extended gjdoc printClassPage printSourcePage
src
doc
File system
CVS
SVN
Version Control
Check out, commit , update, diff...

Figure 5: Overview of WikiJavadoc system

5 Detail Design

The detailed design will be discussed in this section. This part is separated into 6 subsections. The first subcsection is about user authentication design. The second subsection contains project management design. The third subsection covers source code rewrite. An undependent tool, namely, the UpdateJavadoc tool, is designed. The fifth subsection contains logging resolution. The last subsection is the conclusion about the whole work procedure of the WikiJavadoc system.

5.1 Using Technology

5.1.1 Programming Language

Since an inline documentation tool (Javadoc) with Wiki functionality is to be created, and Java source codes need to be manipulated, Java is the best choice instead of scripting languages such as Perl, Python, Ruby and more visual IDE-based ones like Visual Basic.

5.1.2 Form based Authentication

Form based Authentication provides more control over the look and feel of the login process.

5.1.3 JSP, Servlets, HTML

- Servlets:

  Srevlets provide a way to generate dynamic documents.

  Pros: Servlets are both easier to write and faster to run

  Cons: HTML embedded in Java source file.

  JSP:

JSP is based on servlet, It does not give anything that could not in principle be done with a servlet.

Pros: Compared to pure Servlets, JSP separates look and content clearly, so that programers can first insert the dynamic content and later design the web page.

Cons: Run-Time, since all pages need to be recreated after every modification.

- HTML: Pages could be statically generated. HTML pages can be linked to servlet script for edit action by entering the adress.

  Pros: HTML is simple and a open standard. It can be cached.

  Cons: Changes could trigger long update operations on files that are rarely accessed.

5.1.4 Gjdoc, Eclipse JDK, ASTParser and Equinox Incubator

- Gjdoc[13]:

  Gjdoc is a replacement of Javadoc from GNU Classpath. It uses Antlr as the parser. Gjdoc can be used as a driver for a user-specified doclet. It is compatible with Javadoc up to and including version 1.4.

  Pros: (vs. Javadoc) GPL (GNU General Public License)

  Cons: Thus far there is no support for Java 1.5. It can not recreate the source files from within the doclet.

- Eclipse Java Development Tools and ASTParser:

  Eclipse's JDT has its own Document Object Model (DOM) and the Abstract Syntax Tree (AST). Eclipse's JDT generates an AST from the java source which holds all the information about the source (including location).

[13]http://www.gnu.org/software/classpath/cp-tools/

Pros: It performs all required functions and supports Java 1.5.

Cons: It uses its own complex API

- Equinox Incubator:

  It is embeded in a servlet Container and builds a common infrastructure for launching eclipse from a servlet container and creating OSGi based servlet applications.

  Pros: It is the only choice for server side eclipse.

  Cons: It needs eclipse 3.2 or later.

5.1.5 Ant, Logj4

- Ant: Ant is a Java based build tool. Like java, it is platform independent.

  Pros: It has a large variety of common tasks. It is extensible, and it is not hard to create the user's own tasks.

- Logj4:

  Log4j is a powerful log manager.

  Pros: It supports the Apach Commons Logging interface.

  Cons: Log4j as the Tomcat system logger needs to be installed and configured .

5.2 Security

In the WikiJavadoc system, only authorized users are able to manage Wiki projects, e.g. execute creating, updating, or deleting projects. The anonymous users can only edit the exsiting APIs. Therefore, the authentication mechanisms should be used to protect the project management resources.

There are different ways to implement page security. The servlet specification requires servlet containers to provide implementations of basic and digest authentication, as defined in the HTTP/1.1 specification. Additionally, servlet containers must provide form-based security that allows developers to control the layout of login screens. Servlet containers may provide SSL and client certificate authentication. However, the non-J2EE compliant containers are not required to do so.

Form-based authentication is a bit more effort to provide a well-designed, descriptive, and friendly login page. Obviously, it is more feasible for the WikiJavadoc system. The WikiJavadoc system requires a login page. And an error page will be displayed if the login fails. The security page in WikiJavadoc will be implemented similar to Tomcat's project management page.

5.3 Project Management

Project management contains creatinf, updating deleting projects. To create a project it is necessary to do two steps: In a first step the Java source files must be loaded. In a second step the JavaDoc HTML pages, including HTML source code pages, are created from loaded the Java source files. The created project can be updated or deleted. To update an existed project the source file will be reloaded, then Java documentation is created. To delete a project all APIs including their sub directories should be deleted.

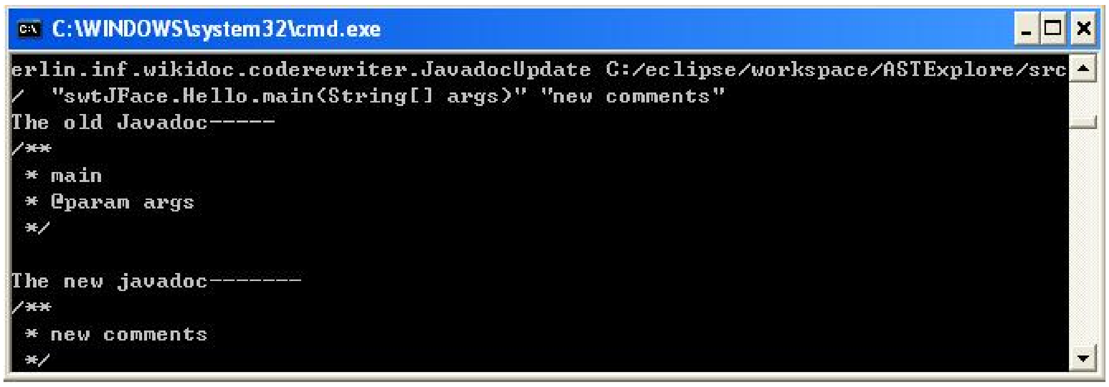
For these work procedures the development tool ANT is suitable. With ANT programs are executing automated. It is comparable compared to other development tools spreading in the UNIX world. To execute a program, targets need be defined in an ANT script. The targets summarize individual work procedures, which are necessary for executing a program. Dependence between individual targets can be defined. ANT solves these dependence through calling targets in the appropriate order. In the targets ANT tasks will be called. A task represent particulars, indivisible work steps. It has a large variety of common tasks already catered for. Loading source can be realized with the standard ANT tasks completely . Generating documentation can be implemented easily as ANT task.

5.4 UpdateJavadoc Tool

The updateJavadoc tool is designed to update JavaDoc comments in source code in a selective manner. The tool is a subproject of the WikiJavdoc system. It can be used independently from the command line.

When one begin to write an application that manipulates Javadoc comments in source code, the procesure can become a little complicated. A Java Parser is needed to parse Java source code. Considering of the powerful parsing abilities of Eclipse, we decided to take advantage of the Java Development Tooling (JDT), which has a powerful ASTParser. Eclipse's JDT has its own Document Object Model (DOM) and the Abstract Syntax Tree (AST). The Eclipse's JDT allows the developers to create and modify Java source code. Even compilation of the created or modified source code can be invoked programmatically by the developer. ASTParser can generate new Java source files from abstract syntax Trees. An AST contains all necessary information, which can be extracted by a deep run and be written into a new Java source code file. With this procedure the modifications from the Wiki are replaced and written back in the branch. Because AST parser is part of eclipse IDE, the UpdateJavadoc tool should be a headless Eclipse plug-in. The ASTParser will be discussed in detail in Section 5.

Fig. 6 is an example to use this tool with the following commands.
"java -cp startup.jar org.eclipse.core.launcher.Main -application" starts eclipse runtime from console, but no UI of eclipse show up. "de.fu_berlin.inf.wikidoc.coderewriter" is the name of the UpdateJavadoc tool. JavadocUpdate is ID of the extension point. "C:/eclipse/workspace/ASTExplore/src" is the path of the Java source files. "swtJFace.Hello.main(String[] args)" is the qualified name of the class or the class number. "new comments" refers to the modifcations that the user edited. After the modifica-



```
erlin.inf.wikidoc.coderewriter.JavadocUpdate C:/eclipse/workspace/ASTExplore/src
/  "swtJFace.Hello.main(String[] args)" "new comments"
The old Javadoc-----
/**
 * main
 * @param args
 */

The new javadoc-------
/**
 * new comments
 */
```

Figure 6: Using UpdateJavadoc tool from conmand line

tions is rewritten to source code, the original Javadoc and the modified Javadoc will be printed in console, so that the user can compare them easily.

An UpdateJavadoc tool command consists of a few elements: java -cp startup.jar org.eclipse.core.launcher.Main -application

de.fu_berlin.inf.wikidoc.coderewriter.JavadocUpdate

[path of java source file] [class or class number] [new comments]

For the purposes of using UpdateJavadoc tool from the command line, the user needs to give the correct parameters.

- path of java source files:

  The absolute path of a java file is needed and it ends with suffix ".java". Such as, C:/eclipse/workspace/classpath/gnu/classpath/src

- class or class number:

  The fully qualified expression is needed. It identifies the method, attribute or class to be changed.

  Such as: swtJFace.Hello.main(String[] args)

  swtJFace is package name, Hello is class name, main(String[] args) is method name.

- new comments:

  The comments must be written according to the specified rules.The rules are fully compatible to the JavaDoc rules used for the same problem.

5.5 JavadocGenerator Tool

This tool generates API documentation in HTML format from Javadoc comments in source code. Gjdoc may be used under the GNU General Public License, so this Javadoc-Generator tool an be implemented through extending gjdoc, especially its doclet.

5.6 Logging

There are two different methods for logging.

- Debug information:

  The important work procedures, like data that are written to standard output and standard error, should be written in a log file, in order to reconstruct the function of the program better and to find errors more easily.

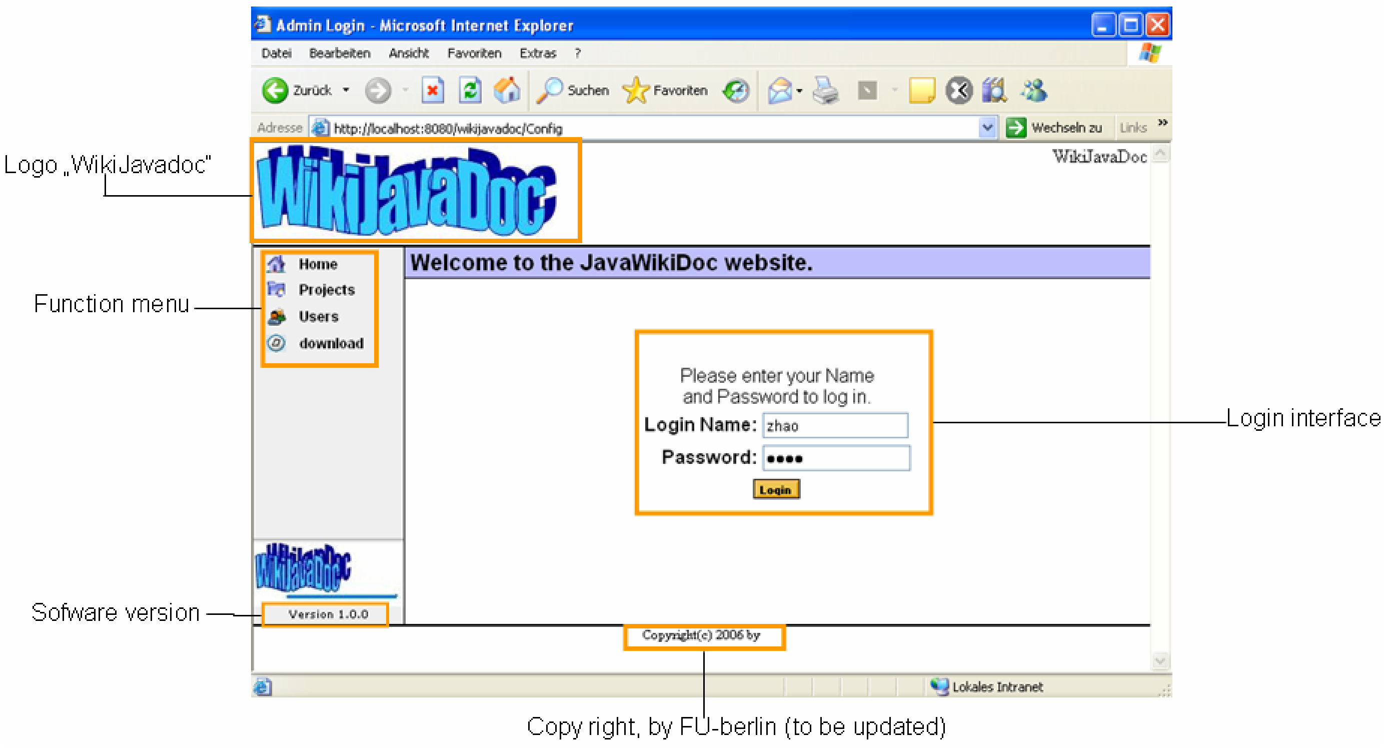
- Security information:

  Additional, changes of the Wikis should also be logged. The changes of Wikis should therefor be retracable. The logging definition should also cover the IP address and the registered user.

  In order to fulfill these requirements, the Logging Framework Log4j should be used. The output is nearly arbitrarily configurable and different kinds of loggings would be treated separately. For example, it is possible to reject the logging information on the work process of the Wikis, but to store the logged accesses durably on the hard-disk.

5.7 Mockup Walkthrough

1. If a user wants to create a WikiProject, he or she should install JDK 1.5 , Tomcat 5.5, Tortoise SVN and Tortoise SVN. The user will download the pre-built web application archive wikijavadoc.war file and adds it to the webapps dictionary in the tomcat installation directory. For security reasons, use of the administration WikiJavadoc is restricted to users with the role ”admin” and ”wikidocAdmin”. Users are defined in $CATALINA_HOME/conf/tomcat-users.xml. she adds the role "wikidocAdmin"in tomcat-users.xml. Of course The administrator of Tomcat is the default Administrator of WikiJavadoc.The URL is the server’s base with the /wikijavadoc/ context, e.g., http://localhost:8080/wikijavadoc.



Welcome to the JavaWikiDoc website.

Please enter your Name and Password to log in.

Login Name: zhao

Password: ●●●●

Figure 7: Login interface of the WikiJavadoc

This is the login user Interface of WikiJavadoc system designed for this project. A WikiJavadoc logo is placed at the upper left hand corner. Software version and copy right information are also shown in this interface. The side menu is a block containing function options.

- The option ″Home″ links to the general introduction about the wikiJavadoc system. ″Was is the tool?″ and ″How to config the tool?″. On the home page one can also download the WikiJavadoc User's Guide.
- The option ″Project″ links to the list of projects that were created in this system. The anonymous user can edit Javadoc APIs there.
- The option ″Admin″ links to the admin page, where admin can creat API for new projects, or update and delete the old APIs. If the admin has not registered, the login page will be forwarded first.
- The option ″download″ links to our project page in sourcrfoge.net. From there everyone can download the project, including the software, source codes and User's Guides.

2. After login, the admin page illustrated in Fig.9 is displayed.

   Since WikiJavadoc lacks the required version control functionality, so we have replaced it with another software, e.g. Tortoise CVS and Tortoise SNV, which both have a graphical user interface. So the inexperienced users for version control can use it easily. Fig.11 illustrated the authorized user check out project source files to the local disk with Tortoise SVN.

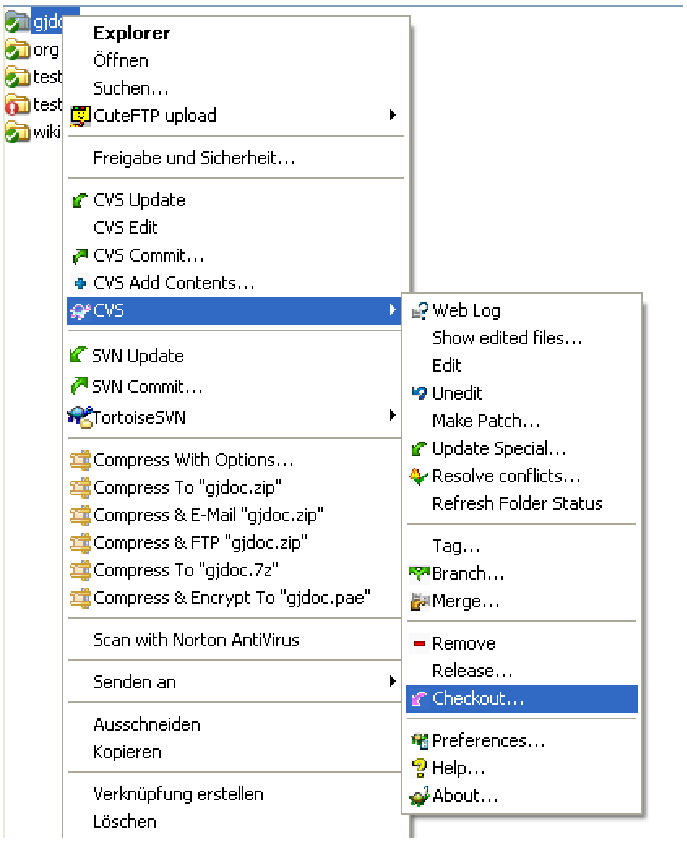
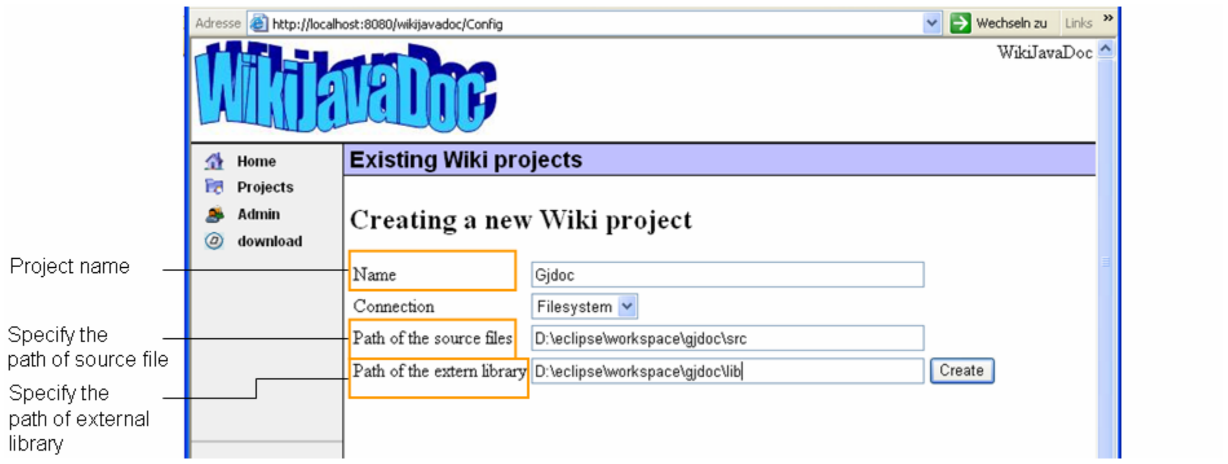


Figure 8: Check out source files with Tortoise CVS

3. After the source files are checked out to the local disk, In the admin page the user must type the project name, the local path of the Java source files and the local path of the java library, and then click on the ”Create” button.The creating WikiJavadoc work begins. Fig. 10 illustrates the admininistrator page of the WikiJavadoc.

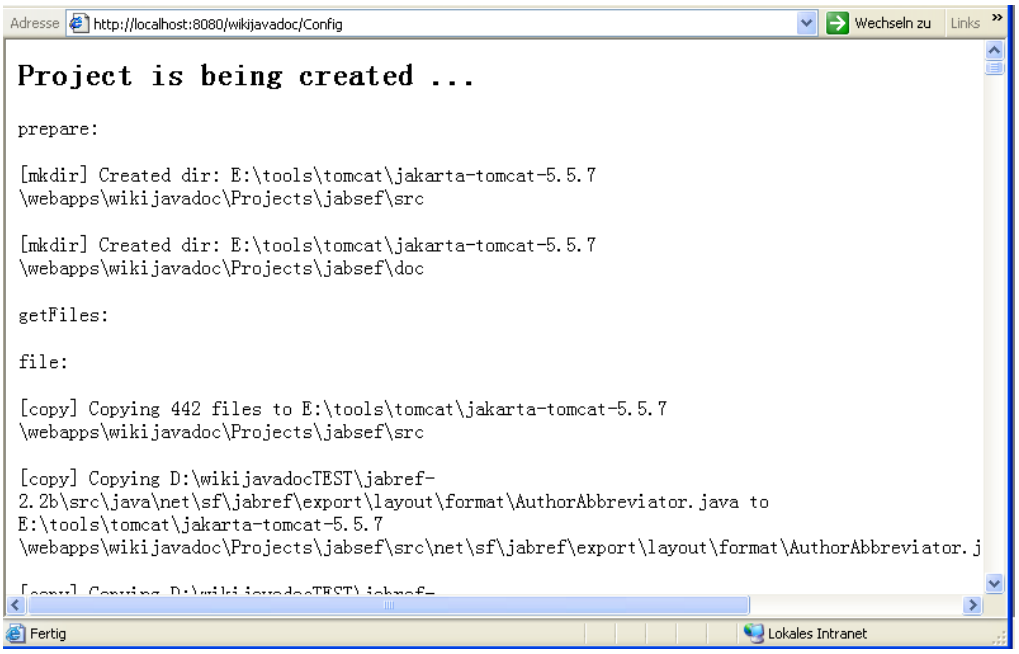


Creating a new Wiki project

Name	Gjdoc
Connection	Filesystem
Path of the source files	D:\eclipse\workspace\gjdoc\src
Path of the extern library	D:\eclipse\workspace\gjdoc\lib

Figure 9: Admininistrator page of the WikiJavadoc

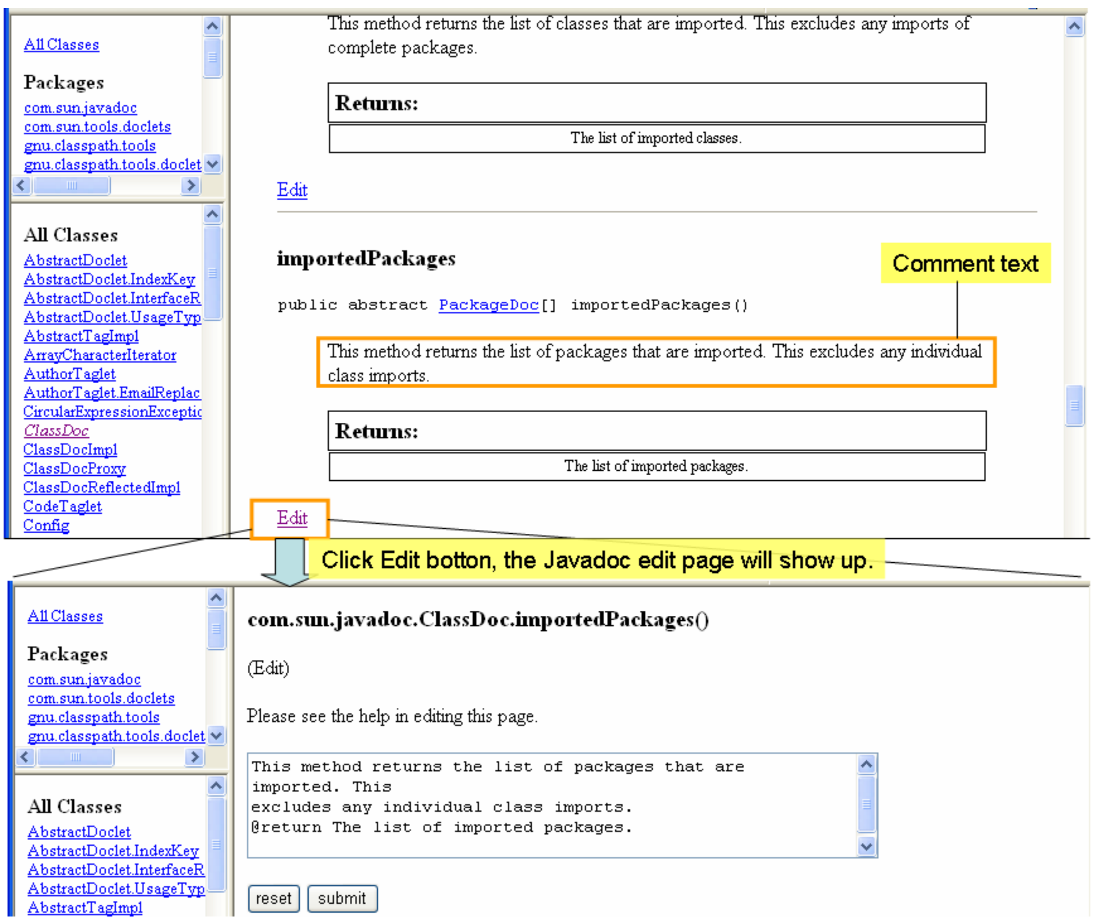
While the user-requested operation is being executed, the information is being fed back to the user during operating. Fig.11 tells the users the information about the creating operation progress. Creating project is in the prepare phase, then in copy phase, etc.



```
Project is being created ...

prepare:

[mkdir] Created dir: E:\tools\tomcat\jakarta-tomcat-5.5.7
\webapps\wikijavadoc\Projects\jabsef\src

[mkdir] Created dir: E:\tools\tomcat\jakarta-tomcat-5.5.7
\webapps\wikijavadoc\Projects\jabsef\doc

getFiles:

file:

[copy] Copying 442 files to E:\tools\tomcat\jakarta-tomcat-5.5.7
\webapps\wikijavadoc\Projects\jabsef\src

[copy] Copying D:\wikijavadocTEST\jabref-
2.2b\src\java\net\sf\jabref\export\layout\format\AuthorAbbreviator.java to
E:\tools\tomcat\jakarta-tomcat-5.5.7
\webapps\wikijavadoc\Projects\jabsef\src\net\sf\jabref\export\layout\format\AuthorAbbreviator.j
```

Figure 10: Information about creating the WikiJavadoc work process

4. After the project has been created, in the projects' page a list of project names will be displayed. Every project name includs a link to its Javadoc documentation. Other anonymous users can also display and edit the APIs. For instance, a user wants to edit the entry of a class she is cuurently browsing, because the user finds the comment inadequate. In this case, the "Edit"button will provide this functionality. After the user clicks on the ”Edit” button, a Javadoc edit page will be displayed. Fig. 12 is just one simple example process of this. The user can edit comment texts, all the Javadoc tags content, like the paprameters, return content, exceptions ect.



This method returns the list of classes that are imported. This excludes any imports of complete packages.

Returns:

The list of imported classes.

importedPackages

```
public abstract PackageDoc[] importedPackages()
```

This method returns the list of packages that are imported. This excludes any individual class imports.

Returns:

The list of imported packages.

com.sun.javadoc.ClassDoc.importedPackages()

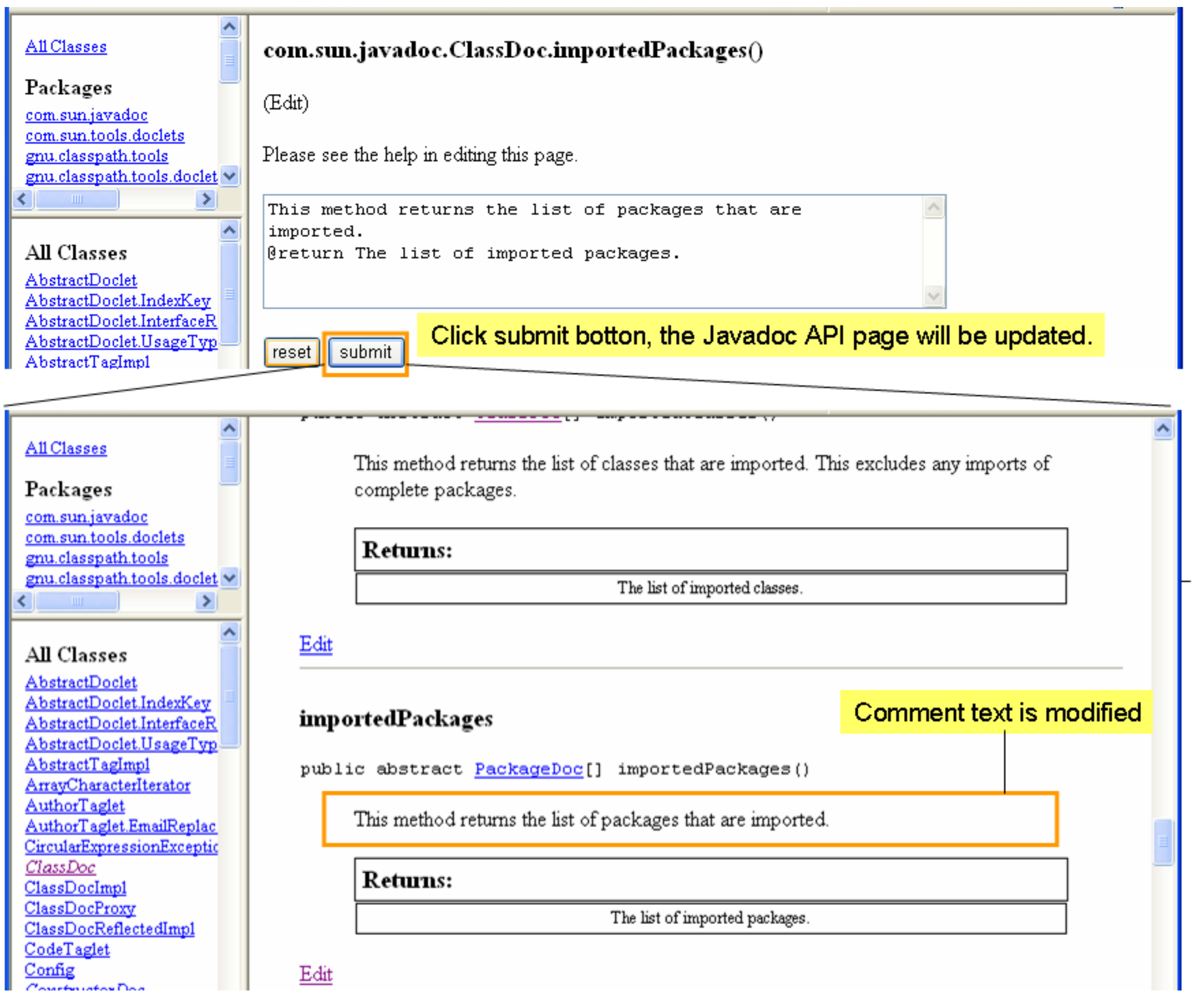
(Edit)

Please see the help in editing this page.

```
This method returns the list of packages that are
imported. This
excludes any individual class imports.
@return The list of imported packages.
```

Figure 11: Edit page of the WikiJavadoc

5. If the user is satiesfied with the changes she can save them using the ”Submit” button, to save all modifications. In our example part of Javadoc text is deleted.



com.sun.javadoc.ClassDoc.importedPackages()

(Edit)

Please see the help in editing this page.

```
This method returns the list of packages that are
imported.
@return The list of imported packages.
```

This method returns the list of classes that are imported. This excludes any imports of complete packages.

Returns:

The list of imported classes.

importedPackages

```
public abstract PackageDoc[] importedPackages()
```

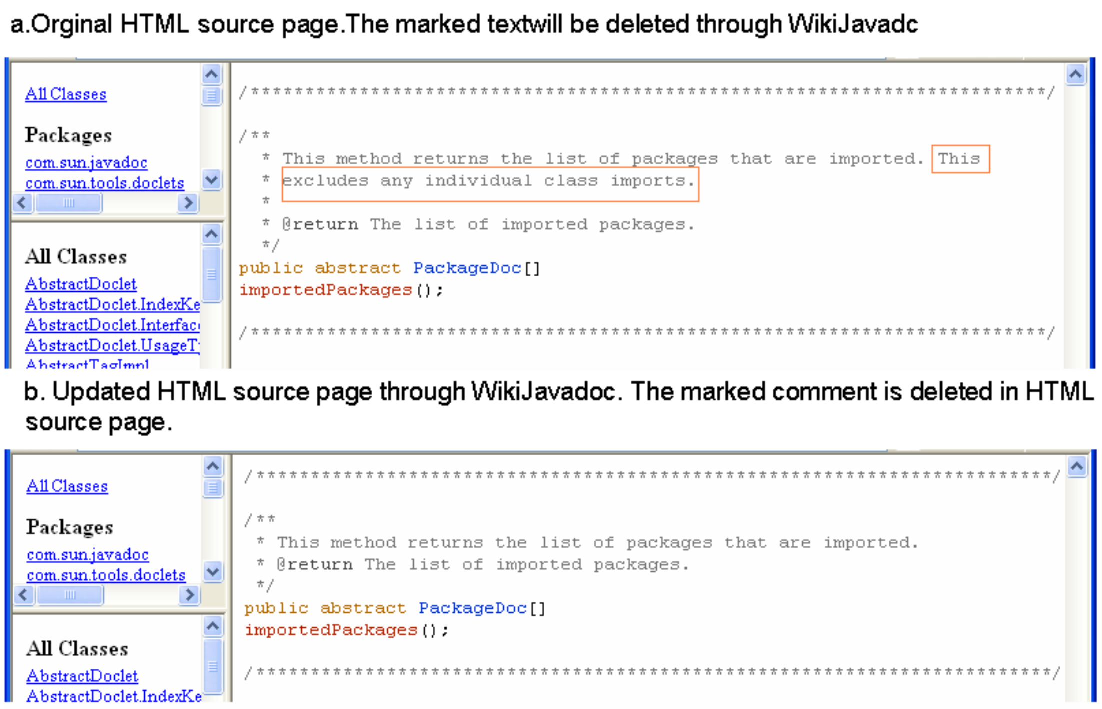
This method returns the list of packages that are imported.

Returns:

The list of imported packages.

Figure 12: Updated html page of the WikiJavadoc

6. The system regenerates the Javadoc appropriately and displays the new version HTML page to the user, at the same time changes are added in the associated java source file. Fig.14 provides the comparison of the HTML source codes page before and after editing. Fig.15 provides the comparison of the source codes before and after editing.



a.Orginal HTML source page.The marked textwill be deleted through WikiJavadc

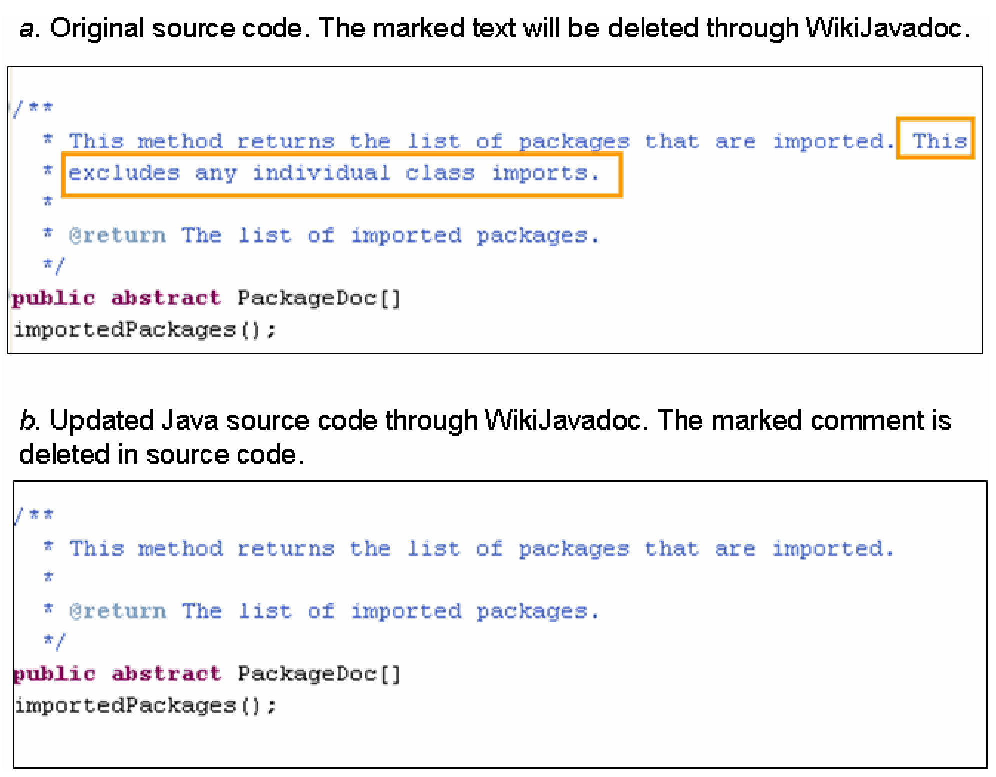
```
/***************************************************************************/

/**
  * This method returns the list of packages that are imported. This
  * excludes any individual class imports.
  *
  * @return The list of imported packages.
  */
public abstract PackageDoc[]
importedPackages();

/***************************************************************************/
```

b. Updated HTML source page through WikiJavadoc. The marked comment is deleted in HTML source page.

```
/***************************************************************************/

/**
 * This method returns the list of packages that are imported.
 * @return The list of imported packages.
 */
public abstract PackageDoc[]
importedPackages();

/***************************************************************************/
```

Figure 13: Comparison between the original and the modified source code

7. If the user is satisfied with the modified Javadoc comments, she can commit the new version source to the CVS respository with Tortoises CVS.



a. Original source code. The marked text will be deleted through WikiJavadoc.

```
/**
  * This method returns the list of packages that are imported. This
  * excludes any individual class imports.
  *
  * @return The list of imported packages.
  */
public abstract PackageDoc[]
importedPackages();
```

b. Updated Java source code through WikiJavadoc. The marked comment is deleted in source code.

```
/**
  * This method returns the list of packages that are imported.
  *
  * @return The list of imported packages.
  */
public abstract PackageDoc[]
importedPackages();
```

Figure 14: Comparison between the original and the modified source code

6 Implementation

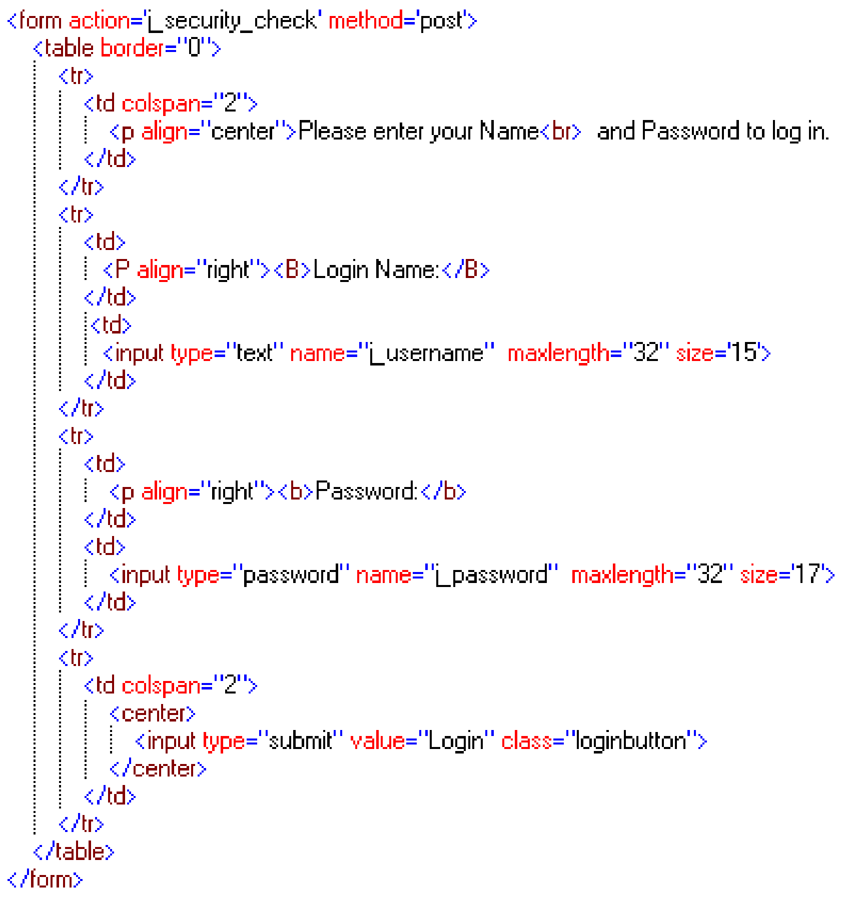
The project WikiJavadoc is a web application and a Tomcat project. This section will be also discussed in 6 subsections.

6.1 Implementation of Form-based authentication

The following steps are performed to implement Form-based authentication of Wiki-Javadoc system:

- Create a login page.

  Login Form Attributes, j_username, j_password, and j_security_check are defined in the Servlet Specification. Fig. 11 is a part of login.jsp. The three attributes are used. ″j_username″ is the name of the username field. ″j_password″ is the name of the password field. And ″j_security_check″ is used in the login form′s action field.



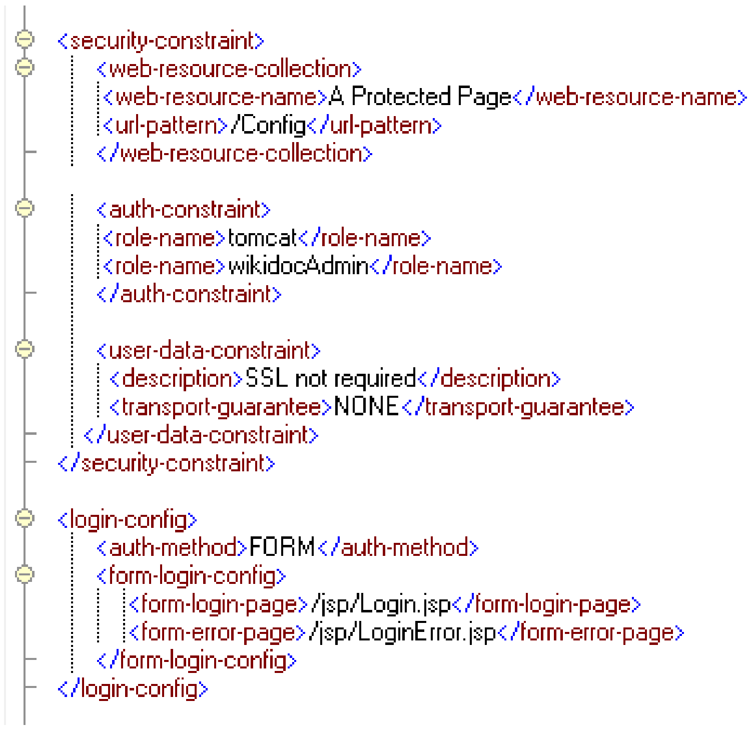
```
<form action='j_security_check' method='post'>
  <table border="0">
    <tr>
      <td colspan="2">
        <p align="center">Please enter your Name<br>  and Password to log in.
      </td>
    </tr>
    <tr>
      <td>
        <P align="right"><B>Login Name:</B>
      </td>
      <td>
        <input type="text" name="j_username"  maxlength="32" size='15'>
      </td>
    </tr>
    <tr>
      <td>
        <p align="right"><b>Password:</b>
      </td>
      <td>
        <input type="password" name="j_password"  maxlength="32" size='17'>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <center>
          <input type="submit" value="Login" class="loginbutton">
        </center>
      </td>
    </tr>
  </table>
</form>
```

Figure 15: Excerpt of code in login page

- Create an error page that will be displayed if login fails.

  The error page shows an error message and provides a link back to the login page. Since it's easy to implement such an error.jsp, it will not be explained here.

- In the deployment descriptor, specify form-based authentication and the login and error pages. Part of the deployment descriptor for the application is shown in Fig. 16. It specifies the security constraint that restricts access to /config to principals in the role of tomcat and wikidocAdmin. The authentication method is specified as FORM, and the login and error pages are identified. The level of security in the transport mechanism is declarant with "NONE", because in our work no encryption is required.



```
<security-constraint>
    <web-resource-collection>
     <web-resource-name>A Protected Page</web-resource-name>
     <url-pattern>/Config</url-pattern>
    </web-resource-collection>

    <auth-constraint>
     <role-name>tomcat</role-name>
     <role-name>wikidocAdmin</role-name>
    </auth-constraint>

    <user-data-constraint>
      <description>SSL not required</description>
      <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/jsp/Login.jsp</form-login-page>
        <form-error-page>/jsp/LoginError.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Figure 16: Specify form-based authentication and the login and error pages

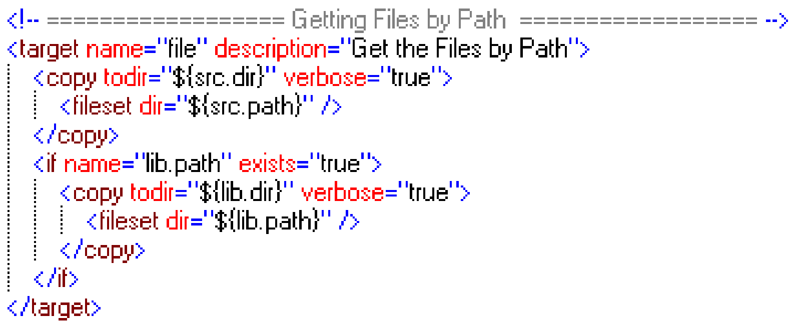
6.2 Project Management

Project management contains creating , updating, and deleting Projects. The popular Apache's Ant build tool is used to manage projects. In this thesis only creating a project will be in details explained.

In WikiJavadoc home, there is a directory named projects which is the root directory of Ant. For creating a WikiJavadoc project two things needed be done.

- copy source codes from local path to the root-directory.
- generate API from those sources.

For the first task the following xml is used in build.xml file.



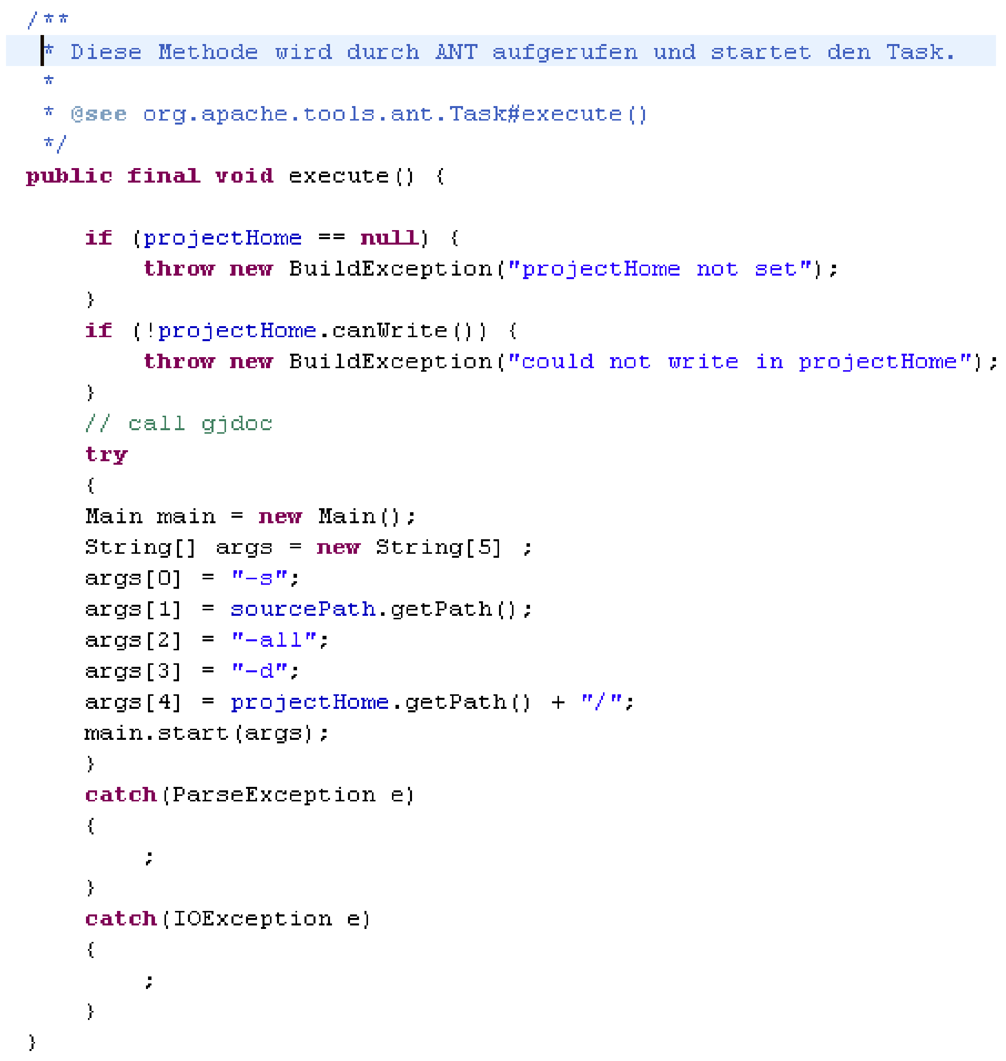
```
<!-- ================== Getting Files by Path  ================== -->
<target name="file" description="Get the Files by Path">
  <copy todir="${src.dir}" verbose="true">
    <fileset dir="${src.path}" />
  </copy>
  <if name="lib.path" exists="true">
    <copy todir="${lib.dir}" verbose="true">
      <fileset dir="${lib.path}" />
    </copy>
  </if>
</target>
```

Figure 17: Activate the copy task

For the sencond task we write a class GjdocTask.java extends org.apache.tools.ant.Task which is integrated in Ant. It gets the project-reference, provides documentation fields, provides easier access to the logging facility. It gives the exact location in the buildfile build.xml where this task instance is used. In the GjdocTask.java the extended Gjdoc method is called.

The following is a part of GjdocTask.java codes.

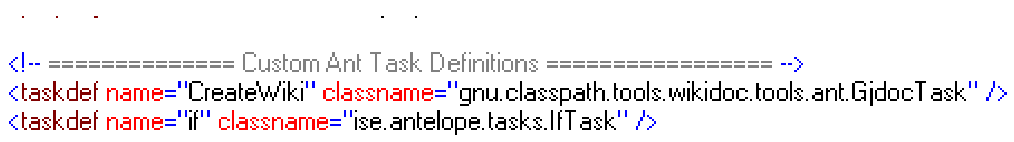
For the deletion of a project Ant execute the task to delete the project dictionary with all contained files and dictionaries.



```
/**
 * Diese Methode wird durch ANT aufgerufen und startet den Task.
 *
 * @see org.apache.tools.ant.Task#execute()
 */
public final void execute() {

    if (projectHome == null) {
        throw new BuildException("projectHome not set");
    }
    if (!projectHome.canWrite()) {
        throw new BuildException("could not write in projectHome");
    }
    // call gjdoc
    try
    {
    Main main = new Main();
    String[] args = new String[5] ;
    args[0] = "-s";
    args[1] = sourcePath.getPath();
    args[2] = "-all";
    args[3] = "-d";
    args[4] = projectHome.getPath() + "/";
    main.start(args);
    }
    catch(ParseException e)
    {
        ;
    }
    catch(IOException e)
    {
        ;
    }
}
```

Figure 18: Excerpt of GjdocTask.java

In the build file, first a task named "CreateWiki"is defined that specifies the Gjdoc-Task class. The following xml is involved in the buildfile:



```
<!-- ============= Custom Ant Task Definitions ================= -->
<taskdef name="CreateWiki" classname="gnu.classpath.tools.wikidoc.tools.ant.GjdocTask" />
<taskdef name="if" classname="ise.antelope.tasks.IfTask" />
```

Figure 19: The definition of GjdocTask class in Ant build file

6.3 Implementation of UpdateJavadoc Tool

UpdateJavadoc Tool is a headless (console-mode, non-GUI) application plug-in for Eclipse. Eclipse project consists of Eclipse Platform, IDE, and RCP. The Java IDE is called JDT and compiler that comes as part of Eclipse. Eclipse's JDT has its own Document Object Model (DOM) and the Abstract Syntax Tree (AST). The DOM/AST is the set of classes that manipulate Java source code, detect errors, perform compilations, and launch programs. The Java DOM/AST classes is contained in package org.eclipse.jdt.core.dom. In particular, it provides a full abstract syntax tree for a Java compilation unit, which can be queried for resolved type information, and modified. Its principal classes are AST ASTNode, and ASTParser, which are used in our project.

Eclipse V3.0.2 supports the Java Language Specification, Second Edition (JLS2). It will correctly parse programs written in all versions of the Java language up to and including J2SE 1.4. Eclipse since V3.1 supports JLS3. The JLS3 API can be used to manipulate programs written in all versions of the Java language up to and including J2SE 5 (JDK 1.5). AST.JLS3 has a different structure and more nodes than AST.JLS2: In our work the AST is created with JLS3. If users run this project with old API, an IllegalArgumentException will be thrown.

There are three types of Comment nodes in Java: BlockComment, Javadoc, and LineComment. The AST Tree supports the creation and insertion of Javadoc nodes only. It considers the exact positioning of BlockComment and LineComment nodes problematic. In fact, the UpdateJavadoc tool only needs to modify Javadoc.

How does the tool updateJavadoc work?

First, an AST is created by paring the source code of the given Java file, then with a search algorithm the right node, whose JavaDoc comment is edited by the user, is found. Now changes need be written back to source, and instead of changing the AST, just the Javadoc node is changed. The last step is to get the actual source code, from the Compilation Unit.

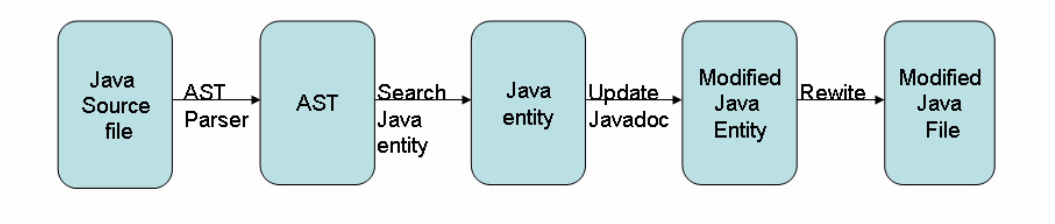


Figure 20: Work procedure of UpdateJavadoc tool

Since Javadoc can be involved in one of nested classes, the search algorithm is a recursive algorithm.

From the AST of one file the TypeDeclaration is provided, and from the TypeDeclaration a list of BodyDeclarations is provided. The BodyDeclarations has 8 different types, like FieldDeclaration, MethodDeclaration except that it can be a nested class. For different type of BodyDeclarations the comparison is done, the right bodyDeclaration is returned. If the BodyDeclaration is a nested class, call recursively the algorithm.

At first I tried to write a normal Java application for this tool. It turns out that method doesn't work, although the logic is correct. The changed Javadoc node can not be returned. The problem is that ASTParser can only work well with eclipse plug-in mode. That's why a headless (console-mode, non-GUI) application plug-in for Eclipse will be created. In other words, via a console without GUI an eclipse plug-in is started, which will access appropriate eclipse jar files from the eclipse platform, .

The OSGi run time, the Extension Registry of Eclipse, as well as eclipse runtime is the mentioned headless eclipse.

A minimal Eclipse headless application will require only:

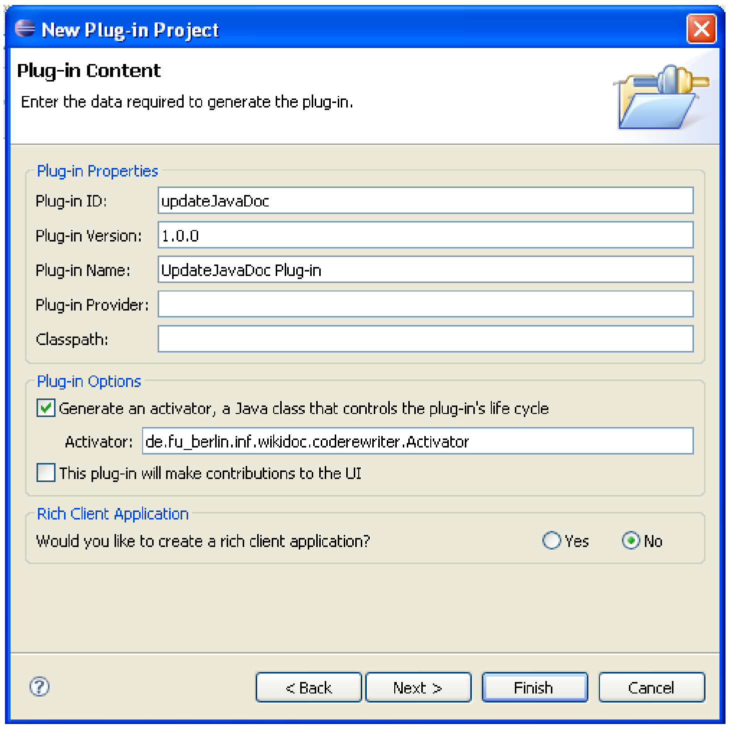
- org.eclipse.osgi
- org.eclipse.core.runtime that requires org.eclipse.osgi.
- a user plug-in providing an application

The plug-in org.eclipse.update.configurator will also be needed if one wants Eclipse-style support for discovering which plug-ins ought to be installed. Otherwise, one needs to add user plug-in to the list of plug-ins to be installed.

Plug-in org.eclipse.core.runtime offers applications an extension point, which is evaluated when starting and serves as a starting point of the Application. To be able to start our application, we use startup.jar provided by Eclipse or the OS-specific Launcher. With the program parameter - application <applicationID> we specify which extension point is to be addressed. Also here the OSGi Configuration file must be indicated or existing in the sub folder configuration. It is not difficult to create a headless eclipse plug-in with eclipse. Because It's difficult to find information or tutorials about how to create headless eclipse Plug-in, that will be explained here. At last useful plug-ins from all the eclipse plug-ins are separated.

The pictorial guide for building the UpdateJavadoc headless plug-in is shown here.

To create the headless plug-in perform the following steps, select ″File->new->Project...″ from the main menu and choose "Plug-in Project"from the resulting "New Project"dialog box. Input project name into the ″project name″ text box. Accept the rest of the default preloaded values on the ″plug-in Project″ and click ″next″.In the next dialog, accept the default preloaded values on the ″Plug-in Content″ page and click ″Next″.


New Plug-in Project

Plug-in Content

Enter the data required to generate the plug-in.

Plug-in Properties

Plug-in ID:	updateJavaDoc
Plug-in Version:	1.0.0
Plug-in Name:	UpdateJavaDoc Plug-in
Plug-in Provider:	
Classpath:	

Plug-in Options

[✓] Generate an activator, a Java class that controls the plug-in's life cycle

Activator: de.fu_berlin.inf.wikidoc.coderewriter.Activator

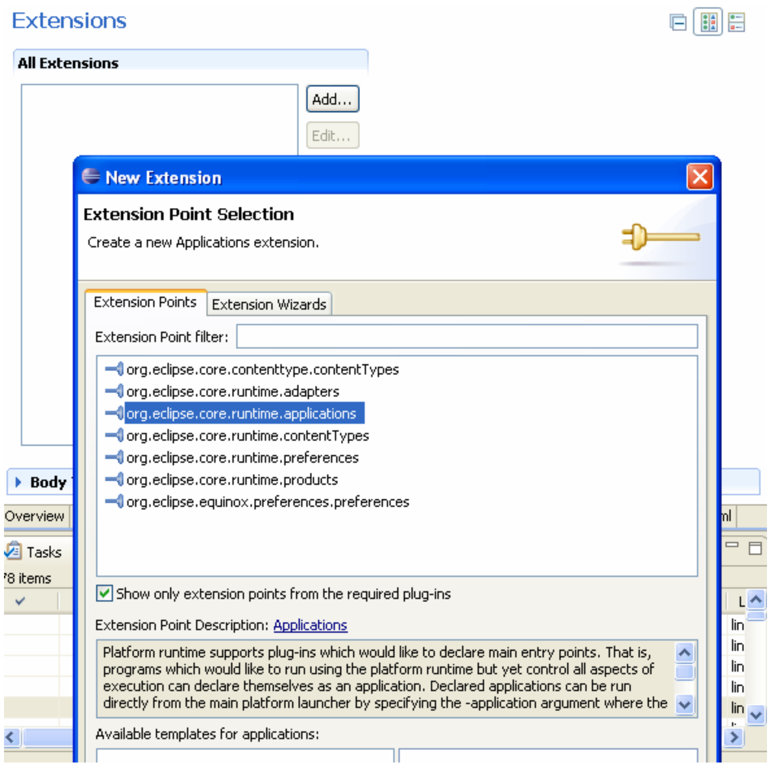
[ ] This plug-in will make contributions to the UI

Rich Client Application

Would you like to create a rich client application? Yes [✓ No]

Figure 21: Create the headless eclipse plug-in II

Directly click on ”Finish” without selecting any templates from the list of ”Available Templatees”. As mentioned above, the headless Plug-in needs an extension point from ”org.eclipse.core.runtime.applications”. To create a plug-in extension, first click on ”Extensions” in the under menu. The extensions page of this project will be displayed. Then click on ”add” button. A dialog will be displayed shown in fig 19. Form the list of extension points org.eclipse.core.runtime.applications is selected. At last click on ”finish”.



Extension Point Selection

Create a new Applications extension.

Extension Points | Extension Wizards

Extension Point filter:

- org.eclipse.core.contenttype.contentTypes
- org.eclipse.core.runtime.adapters
- org.eclipse.core.runtime.applications
- org.eclipse.core.runtime.contentTypes
- org.eclipse.core.runtime.preferences
- org.eclipse.core.runtime.products
- org.eclipse.equinox.preferences.preferences

[✓] Show only extension points from the required plug-ins

Extension Point Description: Applications

Platform runtime supports plug-ins which would like to declare main entry points. That is, programs which would like to run using the platform runtime but yet control all aspects of execution can declare themselves as an application. Declared applications can be run directly from the main platform launcher by specifying the -application argument where the

Available templates for applications:

Figure 22: Create a plug-in extension

To add an application to the Extension, select org.eclipse.core.runtime.applications, right click mouse, then ”New->application”.

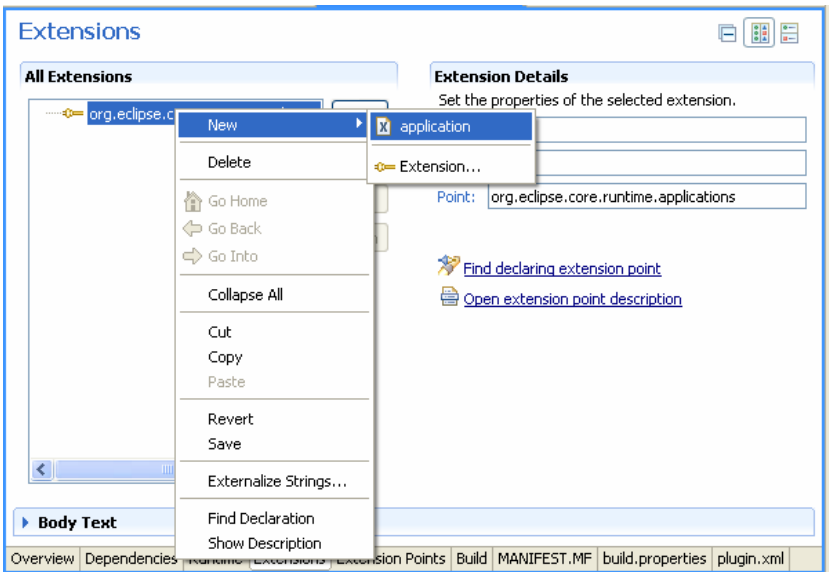


Figure 23: Procedure to add an application to the Extension

To add a run() method to the application, select application, right click mouse, then ”New->run”.

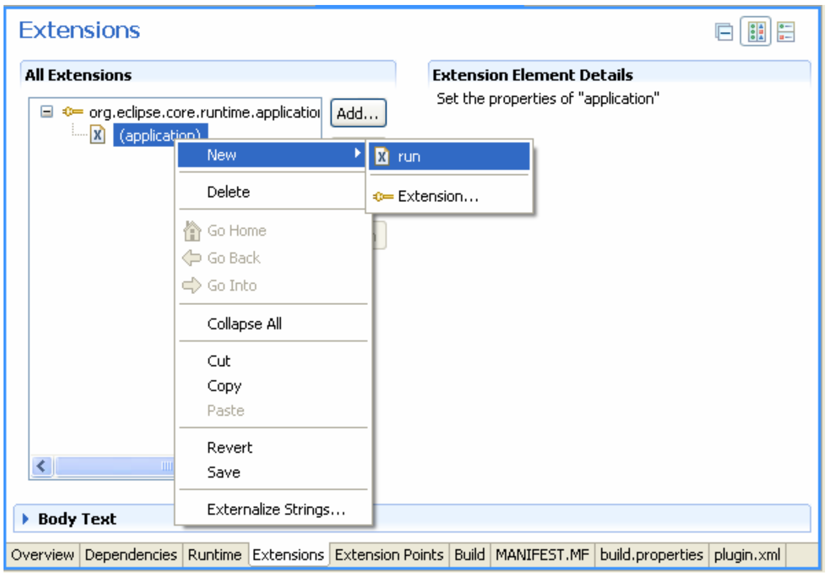
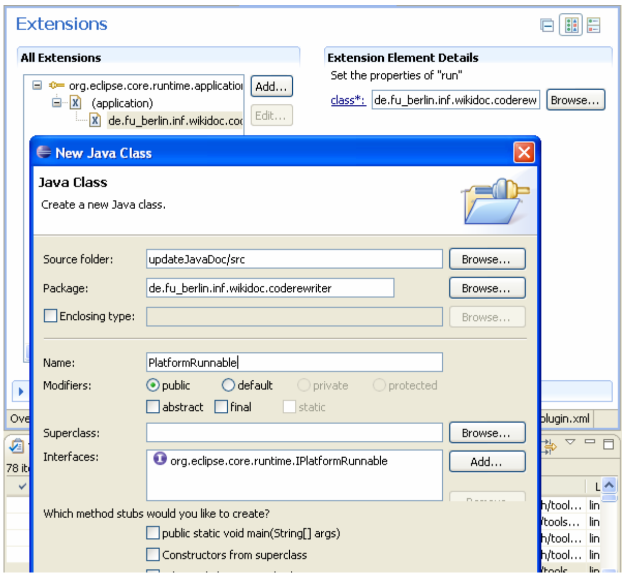


Figure 24: Procedure to add a run() method to the application

To create the class de.fu_berlin.wikidoc.codeRewriter.Runnable, click on ”class*:” with green color, a dialog will be displayed, shown in fig 22. Input package name and class name, accept the rest of default preloaded values, click on ”finish”.

A class which implements Interface IPlatformRunnable is created. In run() method of this class the functional method should be called.



org.eclipse.core.runtime.applicatio
(application)
de.fu_berlin.inf.wikidoc.coc

Set the properties of "run"
class*: de.fu_berlin.inf.wikidoc.coderew

Source folder: updateJavaDoc/src
Package: de.fu_berlin.inf.wikidoc.coderewriter
☐ Enclosing type:

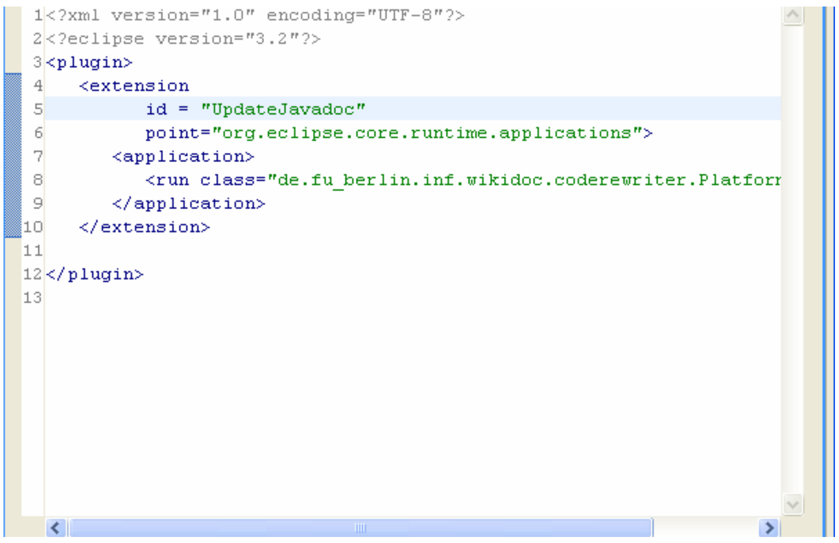
Name: PlatformRunnable
Modifiers: [✓ public] default private protected
☐ abstract ☐ final ☐ static
Superclass:
Interfaces: org.eclipse.core.runtime.IPlatformRunnable

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass

Figure 25: Create the class de.fu_berlin.wikidoc.codeRewriter.PlateformRunnable

After the extension point and application are added, the extension ID must be added, as well.

At last, export the plug-in to a dictionary. Select" File->Export->Deployable plug-ins and fragments", click"next", select a destination directory from the local disk. Click "Finish".



```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id = "UpdateJavadoc"
         point="org.eclipse.core.runtime.applications">
      <application>
         <run class="de.fu_berlin.inf.wikidoc.coderewriter.Platforr
      </application>
   </extension>

</plugin>
```

Figure 26: Plug.xml file of the headless plug-in

6.4 JavadocGenerator

Gjdoc is a documentation generation framework for Java source files. It is a replacement of Javadoc from GNU Classpath. Gjdoc can be used in two ways: as a stand-alone documentation tool, or as a driver for a user-specified Doclet. In the default mode, gjdoc uses the Standard Doclet HtmlDoclet to generate a set of HTML pages. In this work the latest version 0.7.7-3 of gjdoc is used as a driver. And its standard HtmlDoclet is extended so that it generates a set of Html pages with wiki functionility.

How does gjdoc work?

1. First, gjdoc fetches the Class object for the Doclet, then finds the optionLength method and the validOptions method in the Doclet class. It's also OK if the Doclet class doesn't define this method. Then gjdoc finds the start method in the Doclet class, feeds the custom command line tokens to the Doclet, and stores all recognized options.

2. Gjdoc stores packages and classes defined on the command line. For each package specified with the -subpackages option on the command line, gjdoc recursively finds all valid java files beneath it. For each class or package specified on the command line, gjdoc checks that it exists and finds out whether it is a class or a package. The gjdoc creates one file object each for a possible package directory and a possible class file, and finds out if they exist. Gjdoc adds all classes and packages to the RootDocImpl, which is required by Doclet. Then gjdoc validates custom options passed on command line by asking the Doclet if they are OK. Gjdoc adds the valid operations to RootDocImpl. ;

3. RootDocImpl builds. RootDocImpl creates a temporary random access file for caching comment text. Then RootDocImpl parses all files in explicitly specified package directories and all files in "java.lang". RootDocImpl translates every package into an PackageDocImpl, every file into a ClassDocImpl. After all specified

classes are loaded, RootDocImpl loads all classes implicitly referenced by explicitly specified classes. Then RootDocImpl resolves references in comments, resolve pending references in all PackageDocImpls and ClassDocImpls, in comment data of all packages. At last RootdocImpl create an array of ClassDocImpl with all loaded classes.

4. The start method in the Doclet mentioned above is running.

   Doclet generates HTML files for overview, HTML files for full tree in the navigation bar, HTML files for index and HTML files for every package;

We need extend gjdoc's Doclet, so that on one Html class page the class and every class memeber has its own edit link. When a WikiJavadoc project is created, Ant script calls gjdoc. The complete suite of Javadoc HTML pages with edit links will be generated, including HTML pages with syntax-highlighted source code for all classes. Additional, after the Javadoc comments are edited, the edited Javadoc HTML page and its source HTML page should be updated. Two methods are extended in gjdoc, printClassPage(...) and printSourcePage1(...). To run the two extended methods, A new RootDocImpl is needed for the extended Doclet. The ideal approach is that the RootDocImpl, including all its ClassDocImpls is saved in an object oriented database, such as DB4o[14], while a WikiJavadoc project is created. When one class page is edited, retrieve the associated ClassDocImpl, change it refereing to modifications by the user. Thus, we get a modified RootDocImpl. But gjdoc uses Antlr as a Parser, which has not such the powerful parser ability as ASTParser. Another, because of the high coupling between the RootDocImpl and its parser; separation of ClassDocImpl and Javadoc comments in Hashmap, this ideal approach is impossible to implement. Then we select another available approach without database. After the modifications is written into the source file via UpdateJavadoc tool, we directly parse all the files, including the modified

[14]DB4o is the open source object database that enables Java and .NET developers to slash development time and costs and achieve unprecedented levels of performance.

Java file, and create a new RootDocImpl. This approach is fit for most open source software in sourceforge.net. For larger projects with more than 1000 classes, visual-machine memory with 128 MB is not enough. To handel this problem, we should consider implementing a total new documentation framework instead of extending gjdoc. Fortunately, today's computer has 2 G memories, it is no problem to give visual-machine memory more than 128 M.

6.5 Equinox Incubator

WikiJavadoc needs to call the UpdateJavadoc tool, which is a headless plug-in. Equinox Incubator is used in this project to help to run UpdateJavadoc on server side.

Eclipse overview

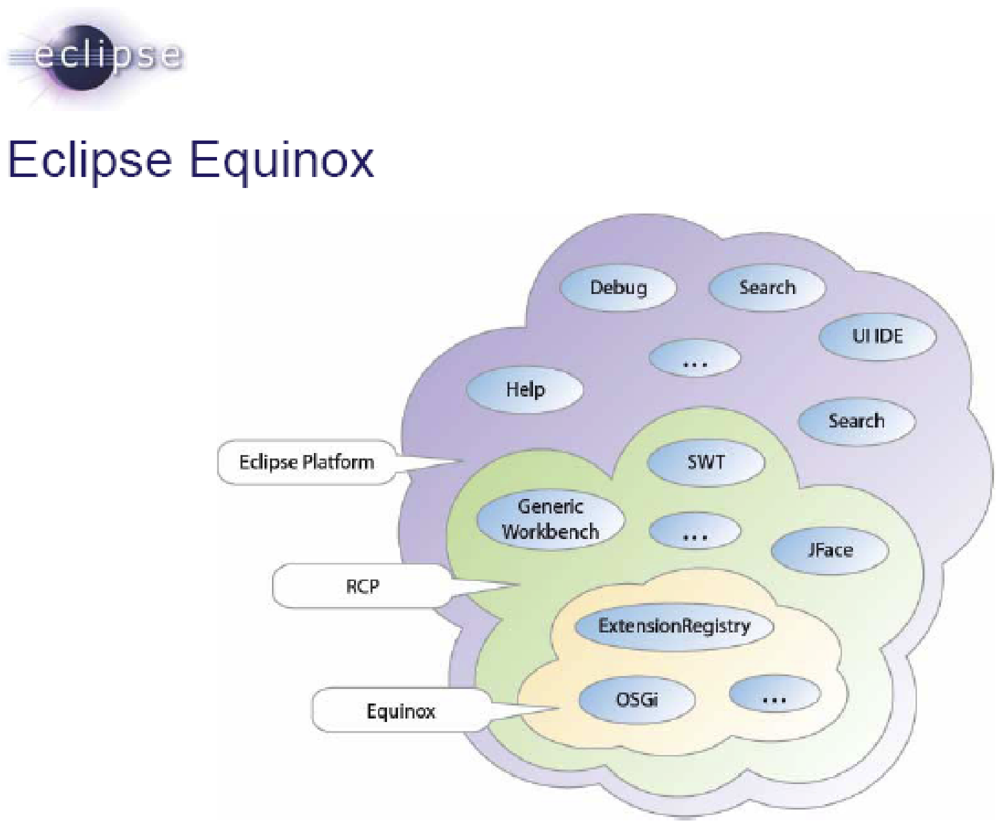


Figure 27: Overview of the Eclipse project[Lip06]

Eclipse is a nice Java-IDE and a well-known framework for developing Rich-Client-Applications(RCP). Fig. 27 shows the relationships among Eclipse platform, RCP, and Equinox. In 2003 Eclipse selected OSGi[15] as the underlying runtime for the plug-in architecture. The goal of the Equinox project is to be ”a first class OSGi community and foster the vision of Eclipse as a landscape of bundles”[EQU06b]. Equinox Incubator is ”an experiment with techniques for broadening the range of Eclipse platform runtime configurations”[EQU06a]. Equinox has a variety of work areas, such as, security, the use of Equinox on the server-side. Since WikiJavadoc needs run a headless plug-in on the server-side, Incubator is used in this project. With Incubator servlets can be packaged in Eclipse-style plug-ins for the server-side. Incubator plug-in cab be deployed and un-deployed while the container WAR file keeps running.

The configuration steps:

1. Download bridge.war from Eclipse page. It is a web application. It launches the framework, provides a place for the framework to hook back into the servlet container, then hooks back into the servlet bridge, proxies through the servlet container to provide an OSGi Http Service.

2. Copy bridge.war to the Tomcat webapps folder.

   Start Tomcat, so that an unpacked bridge project is generated.

   Copy all files and subfolders of the bridge project, except its deployment descriptor(web.xml), to our WikiJavadoc root folder.

   The eclipse Bundles, which are involved in the bridge project, are also required by WikiJavadoc.

3. Specify our own framework launcher in WikiJavadoc’s deployment descriptor, and with one servlet entry assign an incoming request to the BridgeServlet.

[15]OSGI is an abbravation of Open Services Gateway initiative. Here it means a Java-based service platform that can be remotely managed

4. Then begin creating our own deployment Bundle based on an OSGi HttpService.

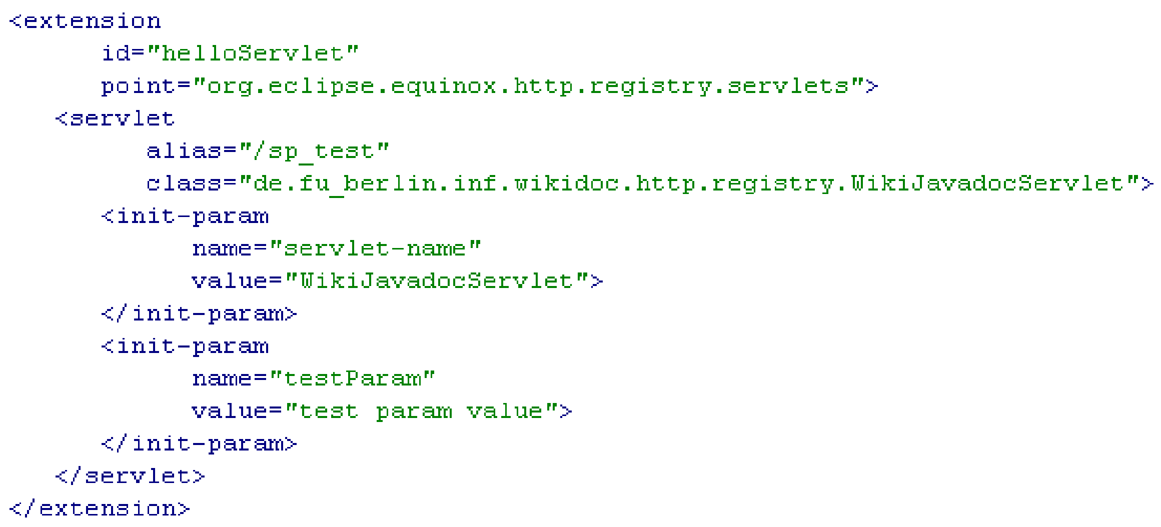
   On Eclipse page it says that there are two approaches of writing a bundle-based server application. One is to write a bundle that uses the OSGi HttpService registered by org.eclipse.eqinox.servlet.bridge.http. The other is to write a bundle that adds extensions from org.eclipse.equinox.http.registry. [EQU06a] Via the extension points of Org.eclipse.equinox.http.registry, we can map servlet, static resources and HTTP service. But we can not map JSP.

   Since WikiJavadoc needs an edit window, a servlet is suitable to our project. The servlet registry is realized via the extension point org.eclipse.equinox.http.registr.servlets.

   - A new bundle project is created in the Eclipse IDE and is called it de.fuberlin.inf.wikidoc.http.registry.
   - In MANIFEST.MF file Require-Bundle, UpdateJavadoc and Import-Package that is from the headless plug-in was added.
   - A new class named WikiJavadocServlet that extends javax.serlvet.http.HttpServlet was created. In WikiJavadocServlet the headless plug-in created earlier was used.
   - The following XML is added to the plugin.xml file of the bundle de.fu_berlin.inf.wikidoc.http.registry. This new element is placed within the <plugin> element.

     In the extension above, the alias attribute locates the servlet in URL space and the class attribute identifies the class that implements the servlet. So the server knows where the servlets are and where they should show up in URL space.

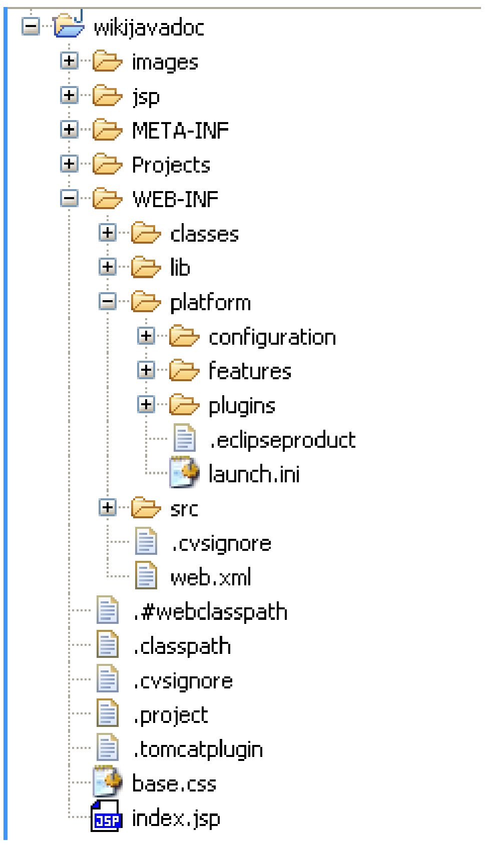
Compared to the normal web application file system structure, the structure of Wik-



```
<extension
      id="helloServlet"
      point="org.eclipse.equinox.http.registry.servlets">
   <servlet
         alias="/sp_test"
         class="de.fu_berlin.inf.wikidoc.http.registry.WikiJavadocServlet">
      <init-param
            name="servlet-name"
            value="WikiJavadocServlet">
      </init-param>
      <init-param
            name="testParam"
            value="test param value">
      </init-param>
   </servlet>
</extension>
```

Figure 28: Excerpt of Equinox Incubator plug-in's Plug.xml file

iJavadoc is a little complicated and is shown in Fig.29. The structure of Equinox incubator is meant to be very close to an RCP application with the /platform directory containing components suitable for server side interaction. In folder wikijavadoc/WEB-INF/plateform/plugins, all Eclipse bundles, which are required in our project, are saved, including also the bundles which UpdateJavadoc requires.



```
wikijavadoc
    images
    jsp
    META-INF
    Projects
    WEB-INF
        classes
        lib
        platform
            configuration
            features
            plugins
            .eclipseproduct
            launch.ini
        src
        .cvsignore
        web.xml
    .#webclasspath
    .classpath
    .cvsignore
    .project
    .tomcatplugin
    base.css
    index.jsp
```

Figure 29: Overview of WikiJavadoc's file system structure

6.6 Version Control

The following approach is adopted to the validation and implementation of the system. WikiJavadoc lacks the required version control functionality, so it is replaced with other software, Tortoise CVS (TCVS) and Tortoise SVN (TSVN). They both have a graphical user interface. This approach will grant flexibility as our requirements evolve.

TCVS and TSVN are both F/OSS licensed under the GNU General Public License.

TCVS does away with the command line interface, and instead it has a graphical user interface. It lets users work with files under CVS version control directly from Windows Explorer. That makes version control an enjoyable experience even for novice users. The user simply right clicks on files and folders to access context-sensitive CVS menus (Fig.9)35. TCVS provides a point-and-click interface for the most common CVS commands.

TSVN is similar to TCVS, provides also a point-and-click interface for the most common SVN commands. The difference is that it is a SVN client rather than a CVS client.

7 Testing

This part explains the test methods and technologies used for the WikiJavadoc tool application and their results. It takes into consideration the functional and non-functional requirements mentioned in the Section Requirements. The testing strategy adopted is explained.

7.1 Test Plans and Results

This test plan is aimed at detecting the differences between the expected behavior specified in requirements and the observed behavior of the implemented system. Testing activities for this application include functional and acceptance testing. Functional testing will be carried out by developing test cases. Acceptance testing is performed by the user to verify the proper implementation of the requirements specified by him.

7.2 Functional Testing

7.2.1 Test Methodology Junit

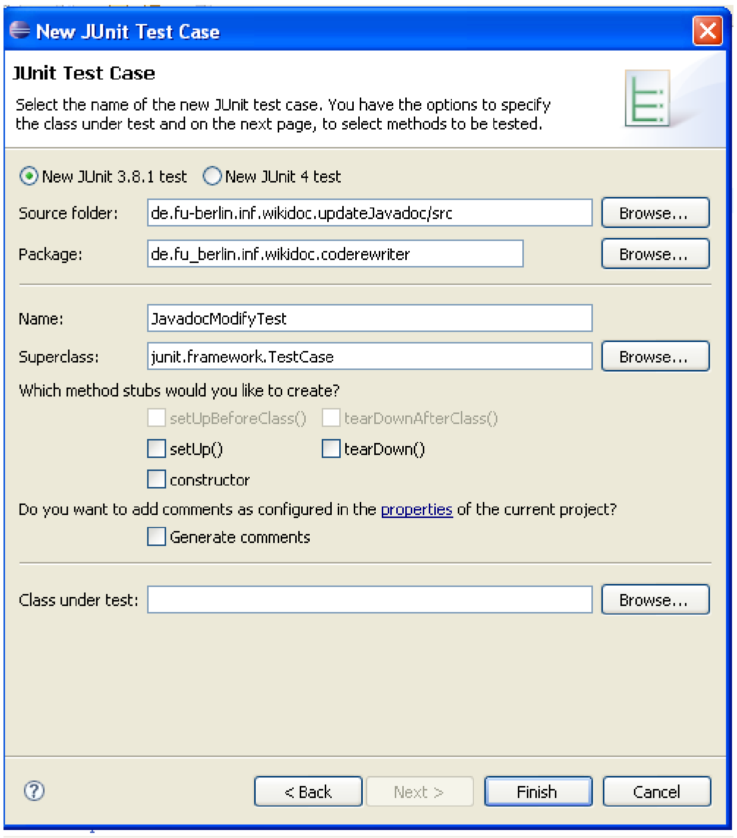
The powerful test methodologies can accelerate testing toolkits, and the development of better testing processes. To test UpdateJavadoc tool we use framework Junit with Eclipse IDE together.

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. It is easy to run many of them at the same time. More information about Junit is available at

`http://junit.sourceforge.net/doc/cookbook/cookbook.htm`.

To create a test for the project updateJavadoc tool, right-click on the project name de.fu-berlin.inf.wikidoc.updateJavadoc, select New -> Other, expand the "Java"selection, and choose JUnit. On the right column of the dialog, choose Test Case, then click Next.

This is illustrated in Fig.30.



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

[✓ New JUnit 3.8.1 test] New JUnit 4 test

Source folder: de.fu-berlin.inf.wikidoc.updateJavadoc/src

Package: de.fu_berlin.inf.wikidoc.coderewriter

Name: JavadocModifyTest

Superclass: junit.framework.TestCase

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()

setUp() tearDown()

constructor

Do you want to add comments as configured in the properties of the current project?

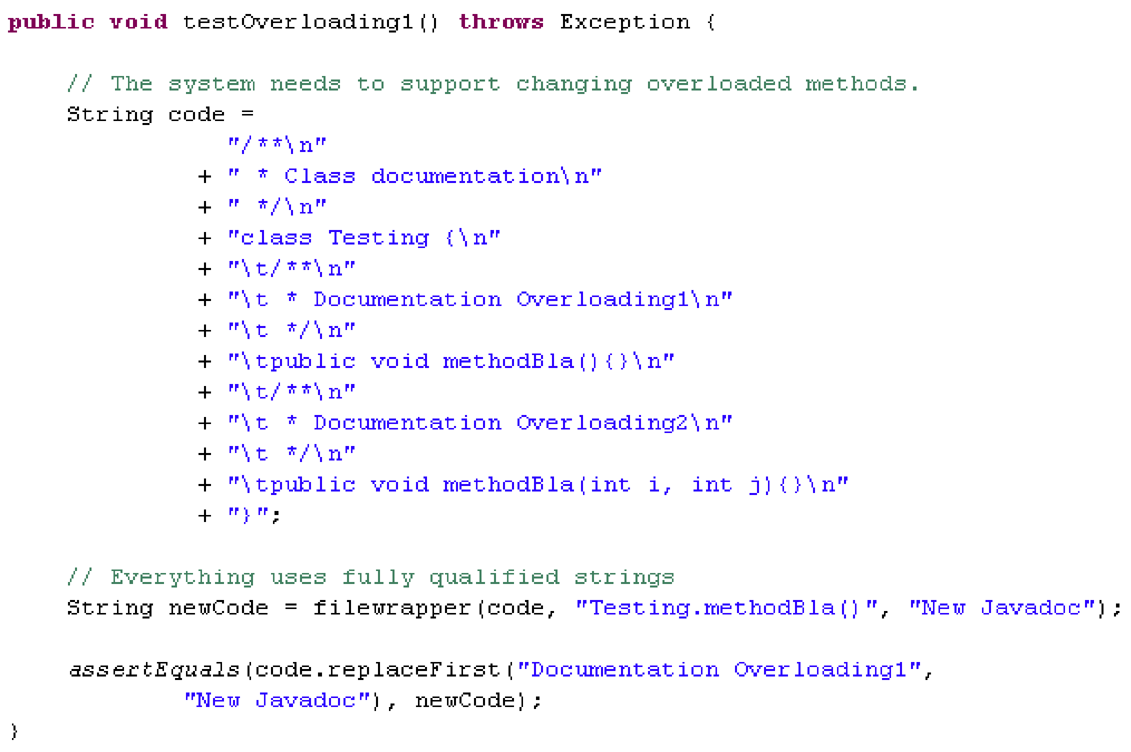
Generate comments

Class under test:

Figure 30: Creating a JUnit test in the Eclipse IDE

Our Test case is named JavadocModifyTest, click on ”Finish”. Java class JavadocModifyTest extends JUnit framework. Junit.framework.TestCase is defined in JUnit’s Javadocs as "a fixture to run multiple tests."In the next step different test cases are defined in JavadocModifyTest. The tests returns void and whose name begins with the string ”test”, such as, testOverloading(), testMultiLineComments() etc..

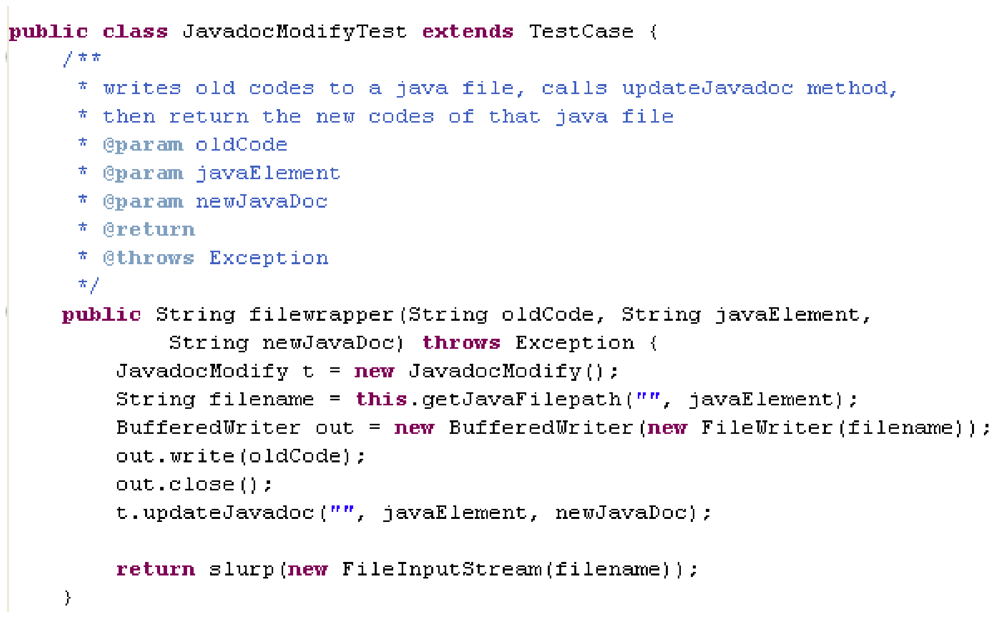
The code for one of test cases is shown in Fig.31:



```java
public void testOverloading1() throws Exception {

    // The system needs to support changing overloaded methods.
    String code =
                  "/**\n"
                + " * Class documentation\n"
                + " */\n"
                + "class Testing {\n"
                + "\t/**\n"
                + "\t * Documentation Overloading1\n"
                + "\t */\n"
                + "\tpublic void methodBla(){}\n"
                + "\t/**\n"
                + "\t * Documentation Overloading2\n"
                + "\t */\n"
                + "\tpublic void methodBla(int i, int j){}\n"
                + "}";

    // Everything uses fully qualified strings
    String newCode = filewrapper(code, "Testing.methodBla()", "New Javadoc");

    assertEquals(code.replaceFirst("Documentation Overloading1",
            "New Javadoc"), newCode);
}
```

Figure 31: Excerpt of code in JavadocModifyTest class

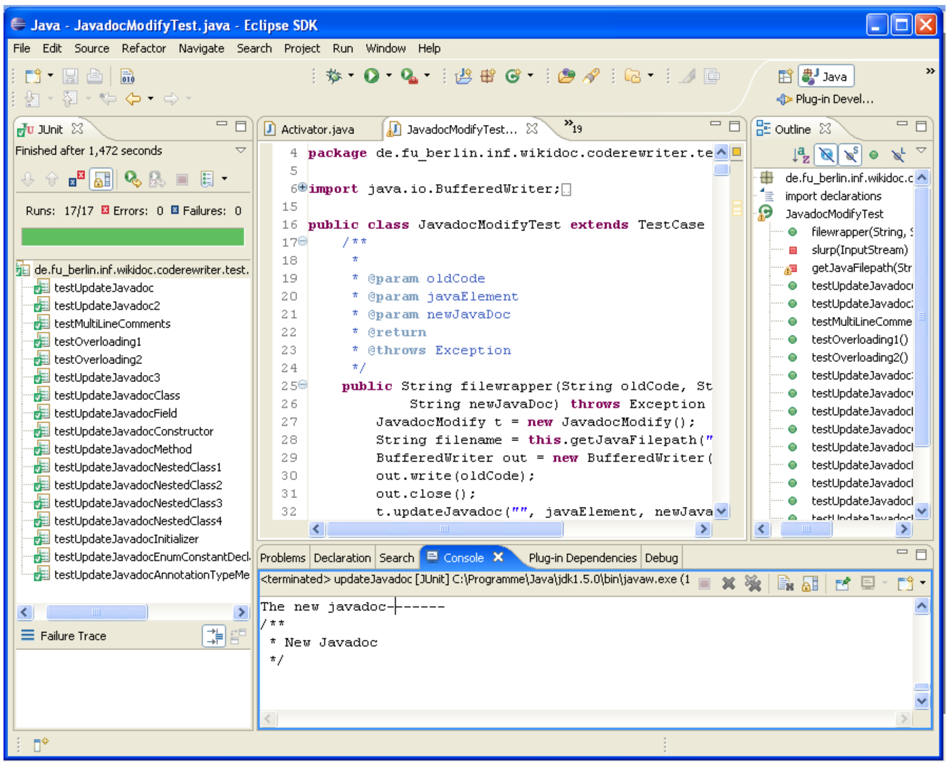
String code contains two methods for overloading. They each have their own Javadoc conmments. The method testOvrloading1() uses an assertEquals() call, which compares the value that we expect to receive against the value returned by filewrapper(¡). filewrapper(¡) is a help method that writes old codes to a java file, calls updateJavadoc method, then returns the new codes of that java file



```
public class JavadocModifyTest extends TestCase {
    /**
     * writes old codes to a java file, calls updateJavadoc method,
     * then return the new codes of that java file
     * @param oldCode
     * @param javaElement
     * @param newJavaDoc
     * @return
     * @throws Exception
     */
    public String filewrapper(String oldCode, String javaElement,
            String newJavaDoc) throws Exception {
        JavadocModify t = new JavadocModify();
        String filename = this.getJavaFilepath("", javaElement);
        BufferedWriter out = new BufferedWriter(new FileWriter(filename));
        out.write(oldCode);
        out.close();
        t.updateJavadoc("", javaElement, newJavaDoc);

        return slurp(new FileInputStream(filename));
    }
```

Figure 32: Excerpt of code in JavadocModifyTest class

Click Run -> Run as -> JUnit Test (remember that JavadocModifyTest.java should be highlighted in Package Explorer). In the left window, instead of Package Explorer, you will see the JUnit window, which shows a green bar, as seen in Fig.33. the tests are successful.


Java - JavadocModifyTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Plug-in Devel...
JUnit
Activator.java
JavadocModifyTest...
Outline
Problems Declaration Search Console Plug-in Dependencies Debug
Failure Trace

```
Finished after 1,472 seconds
Runs: 17/17  Errors: 0  Failures: 0

de.fu_berlin.inf.wikidoc.coderewriter.test.
  testUpdateJavadoc
  testUpdateJavadoc2
  testMultiLineComments
  testOverloading1
  testOverloading2
  testUpdateJavadoc3
  testUpdateJavadocClass
  testUpdateJavadocField
  testUpdateJavadocConstructor
  testUpdateJavadocMethod
  testUpdateJavadocNestedClass1
  testUpdateJavadocNestedClass2
  testUpdateJavadocNestedClass3
  testUpdateJavadocNestedClass4
  testUpdateJavadocInitializer
  testUpdateJavadocEnumConstantDecl
  testUpdateJavadocAnnotationTypeMe
```

```java
package de.fu_berlin.inf.wikidoc.coderewriter.te

import java.io.BufferedWriter;

public class JavadocModifyTest extends TestCase
    /**
     *
     * @param oldCode
     * @param javaElement
     * @param newJavaDoc
     * @return
     * @throws Exception
     */
    public String filewrapper(String oldCode, St
            String newJavaDoc) throws Exception
        JavadocModify t = new JavadocModify();
        String filename = this.getJavaFilepath("
        BufferedWriter out = new BufferedWriter(
        out.write(oldCode);
        out.close();
        t.updateJavadoc("", javaElement, newJava
```

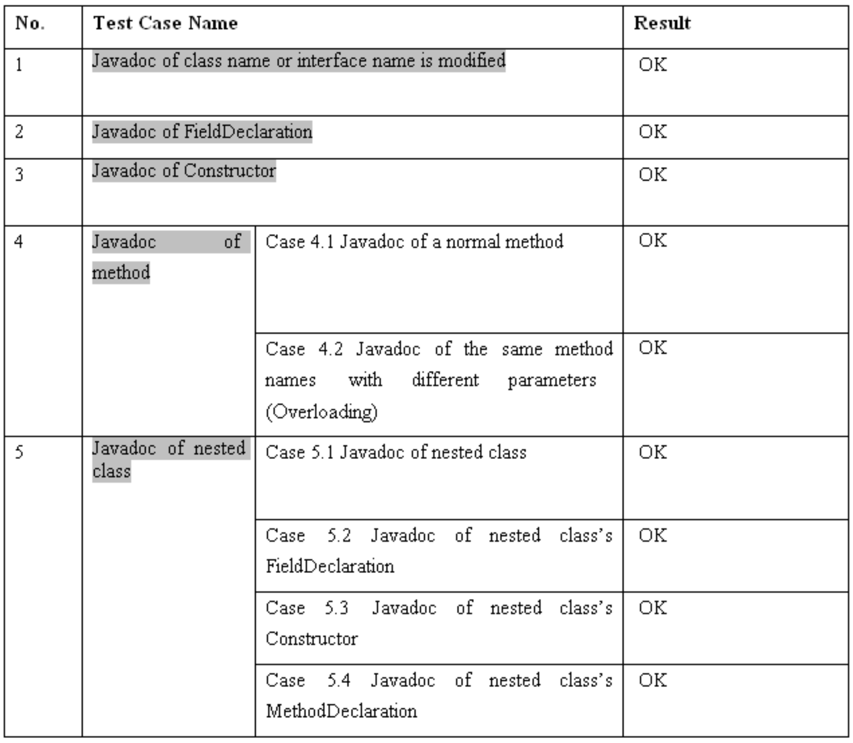
```
de.fu_berlin.inf.wikidoc.c
import declarations
JavadocModifyTest
  filewrapper(String, S
  slurp(InputStream)
  getJavaFilepath(Str
  testUpdateJavadoc
  testUpdateJavadoc2
  testMultiLineComme
  testOverloading1()
  testOverloading2()
  testUpdateJavadoc
  testUpdateJavadoc
  testUpdateJavadocl
  testUpdateJavadoc
  testUpdateJavadocl
  testUpdateJavadocl
  testUpdateJavadocl
  testUpdateJavadocl
```

```
<terminated> updateJavadoc [JUnit] C:\Programme\Java\jdk1.5.0\bin\javaw.exe (1
The new javadoc-------
/**
 * New Javadoc
 */
```

Figure 33: The test result for UpdateJavadoc tool with a JUnit

7.2.2 Test Cases

The traceability of Test Cases to UpdateJavadoc Tool is summarized in fig.33. UpdateJavadoc can handle the cases, such as, Javadoc comments of one class or interface, fields, constructors, overloading, nested classes, etc. The required functionalities work well.



No.	Test Case Name		Result
1	Javadoc of class name or interface name is modified		OK
2	Javadoc of FieldDeclaration		OK
3	Javadoc of Constructor		OK
4	Javadoc of method	Case 4.1 Javadoc of a normal method	OK
		Case 4.2 Javadoc of the same method names with different parameters (Overloading)	OK
5	Javadoc of nested class	Case 5.1 Javadoc of nested class	OK
		Case 5.2 Javadoc of nested class's FieldDeclaration	OK
		Case 5.3 Javadoc of nested class's Constructor	OK
		Case 5.4 Javadoc of nested class's MethodDeclaration	OK

Figure 34: Traceability of Test Cases to UpdateJavadoc

Test Cases to WikiJavadoc system:

A simple example is selected and the following steps are performed to test the whole WikiJavadoc system:

1. Create a Java project 'TestSimple' with Eclipse IDE.

2. Create a source folder, then create two packages. In every package one Java file is created. In every file there are some Javadoc comments to the class or class members.

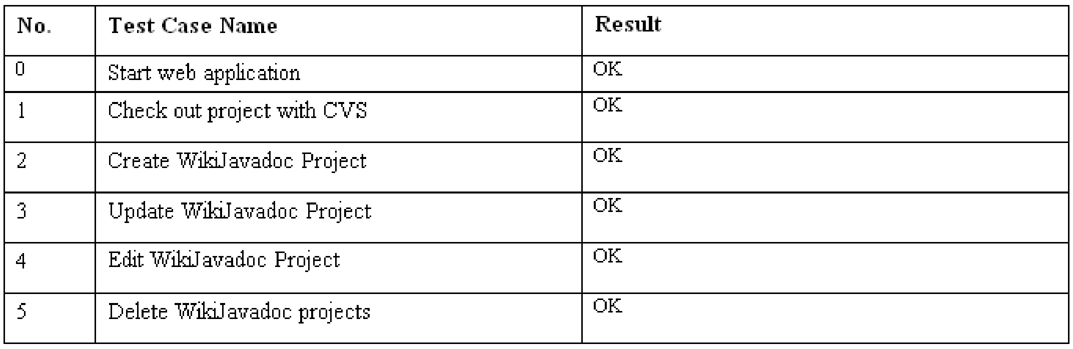
3. Import the project 'TestSimple' to the SVN repository from Sourceforge.net with Tortoies SVN tool.

4. Export the project 'TestSimple' to the locale dictionary WikiJavadoc.

5. Start up WikiJavadoc system, create a WikiJavadoc project named testSimpleWiki from source code in the dictionary WikiJavadoc.
6. Borwse the Javadoc Documentation of testSimpleWiki, then edit comments, click on 'Submit' Button. (Modifications1)
7. Make sure the html Javadoc page, the html source page, and the source files will be updated.
8. Commit the updated WikiJavadoc dictionary to SVN repository from sourceforge.net with Tortoies SVN tool.
9. Update source files from the SVN repository in Eclipse IDE, then make sure if Modifications1 exist.
10. Modify the comments in the source files in Eclipse IDE(Modifications2), then commit to the SVN repositary.
11. Update the source files in the locale dictionary WikiJavadoc fronm SVN repositary with Tortoies SVN tool, make sure Modifications2 exist in the locale dictionary WikiJavadoc.
12. Update the WikiJavadoc project TestSimpleWiki.
13. Browse TestSimple Documentation, make sure if Modifications2 exist in the html Javadoc page and the html source page.

All 13 steps are performed without errors, that shows the WikiJavadoc system works well.

The traceability of test cases to WikiJavadoc is summarized in fig. 34. The WikiJavadoc handles cases, such as creating , updating, editing deleting project. The required functionalities work well.



No.	Test Case Name	Result
0	Start web application	OK
1	Check out project with CVS	OK
2	Create WikiJavadoc Project	OK
3	Update WikiJavadoc Project	OK
4	Edit WikiJavadoc Project	OK
5	Delete WikiJavadoc projects	OK

Figure 35: Traceability of Test Cases to WikiJavadoc

7.3 Acceptance Testing

Of course we want to know whether WikiJavadoc is a truly useful tool for Java developers. On the server of our university, JDK 1.5, Tomcat5.5, Tortoies SVN, Tortoies CVN, WikiJavadoc are installed. 5 open source projects are taken from sourceforge.net. The choice rules is that the projects written in Java and has at least 4 developers. Let WikiJavadoc run on the 5 projects.

The five projects:

- jabref-2.2b[16]: ~~zip file, 3.9M~~
- jasmin-2.3[17]: ~~zip file, 1.24M~~
- jdom-1.0[18]: ~~zip file 3.73M~~
- jnex3.8[19]: ~~zip: 8.95M~~
- jruby-0.9.1[20]:~~zip file 2.76M~~

[16] http://jabref.sourceforge.net/
[17] http://jasmin.sourceforge.net/
[18] http://www.jdom.org//
[19] http://http://jnex.sourceforge.net/
[20] http://jruby.sourceforge.net/

For an initial pilot test, we send mails to c.a 20 students from major computer science of our university and invite them to join in this experiment. We ask them try to improve the documentation of those 5 projects for 30 minutes. For statistical purposes the access to the system will be anonymously logged.

If there are bugs in the programs or something the user wish had been present in the Javadocs of an API they use, they are asked to give us feedback. They report the bugs on the project page on sourceforge.net or write mails directly. Then we'll devote the time and resources to improve WikiJavadoc. If the test-users are interested in receiving the edits performed by users as patches in unified diff format, we will send them.

The following are the test results: There are 26 testers, Every one of them browsed 111 pages in average, and made 13.3 modifications. Fig.36 lists the number of pages pro project the users browse and the number of modifications pro project the users make.

Project	Jabsef		Jasmin		Jdom		Jnex		Jruby		Jabsef	
	brow	edit	brow	edit	brow	edit	brow	edit	brow	edit	brow	edit
Number of pages	940		242		338		762		604		940	
Number of modifica-tions		85		19		21		127		94		85

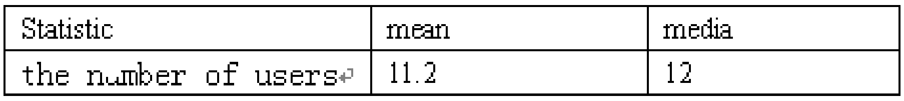
Figure 36: The number of pages and the number of modifications

Fig.37 lists the number of users pro project, most of the testers don't browse all 5 projects, only ca. 2-3 projects.

Project	Jabsef	Jasmin	Jdom	Jnex	Jruby
the number of users	15	7	9	13	12

Figure 37: The number of users pro project

The statistic in Fig.38 shows the arithmetic mean wert is 11.2 and the media wert is 12. That says every projects has 11.2 users in average.



Statistic	mean	media
the number of users	11.2	12

Figure 38: The statitic about the number of users pro project

Testplan2:

Send mails to the mailing list of every project, post messages in the forum of every project, and invite them to use WikiJavadoc for a few weeks. See what will happen? will the developers and users accept this tool? Their feedback is important to improve WikiJavadoc and extend WikiJavadoc's functionalities.

7.4 Conclusion

Testing is a necessary part of the development process. Testing code was always an integral part of any development. But it has been advanced over the last few years. Thanks to powerful methodologies Junit, accelerated testing toolkits, and the development of better testing processes. Thanks to the 20 testers from the university, their feedback are important to find out bugs in the programs or present something the user wish in the API documentation. That improve WikiJavadoc.

8 Future Research

WikiJavadoc is an open source Java inline documentation tool with Wiki functionalities and a JavaDoc enhancement, which makes a normal Javadoc become a collaborative java document.

With the base laid we are planning to improve and add new features to WikiJavadoc.

The improvements will include:

8.1 CVS Portal

CVS is widely used around the world, including in many free and open source software projects. In the current version of WikiJavadoc the source code is loaded with normal CVS or SVN tool, like Tortoises CVN, or Tortoises SVN. A CVS portal can be added to the WikiJavadoc system. It allows users to run basic CVS operations from the web. That will facilitate the work procedure of the administrator.

Up to now no CVS web application in Java exists, that can execute the complete suite of CVS operations. We may consider using an open source CVS client written in Java, such as Jcvsweb. It is an open source CVS client web application written entirely in Java according to the CVS client/server protocol. It involves a com.ice.cvsc package and a web application. The package implements the CVS client/server protocol and allows any Java program to implement the complete suite of CVS operations. The web application presents any CVS repository on the internet just for browsing and checking out source files. Unfortunately, it can not commit, update source files to the CVS repository. We can consider extending Jcvsweb with other functionalities. Another, Jcvsweb uses Struts Tiles template to implement its interface. WikiJavadoc uses technology Equinox Incubator. We need consider integrating Struts Tiles and Equinox Incubator. In other words, we need run Struts Tiles in eclipse plug-ins on server side. It is a new try and surely much work , but it is possible from a technological aspect.

If the CVS portal is added to WikiJavadoc successfully, in the next step we may consider adding a Subversion web application.

8.2 Authoritative-Versioning

In order to make WikiJavadoc robust, the concept of authoritative-versioning will be introduced. The versioning should have these functions:

1. Users are allowed to see the difference between the authoritative and user modified version.

2. At any time, WikiJavadoc can be rolled back to any version in its history if necessary. The detailed method is to store versions with folders named after modification dates. Only the administrator has the right to restore earlier versions.

8.3 Log Contributions

A sign-on/sign-off mechanism for anonymous users cab be provided. Depending on project size, the sign-on/off mechanism will be activated. For larger projects, such as, GNU ClassPath with about 1000 class files, it is necessary that that changes exceeding a certain size can only be updated by an authorized contributor. In this case the log on/off system will be activated. On the other hand, if a project only requires an easy web-interface for users to modify the inline API documentation, the sign on/off mechanism can be switched off, or the authorization limit can be lowered.

9 Summary

This diploma thesis presented WikiJavadoc, which is a freely available and modifiable Java documentation tool and an JavaDoc enhancement. It extends the standard Javadoc with Wiki functionality and has the following additional features compared to the standard Javadoc:

- Popular Interface: WikiJavadoc provides a proven, known, and popular interface, because it integrates the interface of Javadoc and Twiki respectively.
- Javadoc collaboration: WikiJavadoc makes a normal Javadoc become a collaborative Java document. ~~For a Java project,~~ non-developer users can also document a Java project. WikiJavadoc is effective to put together collaborative Javadoc-style comments in a short time based on its Wiki functionality.
- ~~Linking HTML source page: WikiJavadoc provides also HTML source code pages. That makes the non-developer users can write comments better according to HTML source pages.~~
- Writing back modifications to source: Users can reflect the changes from the Javadoc page back into the source code directly. That makes contributing to favorite APIs much easier.
- Users Collaboration: Users can be asked to collaborate as well. For instance, if a user can send a question to others via a mailing list, others may contribute their solutions to the WikiJavadoc documentation.

Of course, on this field some useful improvements can be made, enpowing WikiJavadoc.

We believe WikiJavadoc is a promising tool to help complete Java documentation, to increase reusability and enhance quality. Thus, it makes the software more accessible to users by bundling a richer documentation. WikiJavadoc will be developed further, possible improvements are sketched in section 8.8

References

[Alm06] Dion Almaer. Web form-based authentication. *Onjava.com, http://www.onjava.com/pub/a/onjava/2001/08/06/webform.html*, 2006.

[Ber00] Erik Berglund. Writing for adaptable documentation. *IEEE*, 2000.

[CHYM03] Davor Cubranic, Reid Holmes, Annie T.T. Ying, and Gail. C. Murphy. Tools for light-weight knowledge sharing in open-source software development. 2003.

[DOC05] Docenhancer's homepage, http://www.alphaworks.ibm.com/tech/docenhancer. *Documentation Enhancer for Java*, 2005.

[Dru00] John G. Drummond. Open source software and documents: A literature and online resource review. *Omar's official Web Site, http://www.omar.org/opensource/litreview/*, 2000.

[EBN00] Henrik Eriksson, Erik Berglund, and Peter Nevalainen. Using knowledge engineering support for a java documentation viewer. *SEKE*, 2000.

[EQU06a] Eclipse Equinox Incubator's homepage, http://www.eclipse.org/equinox/incubator/. *Equinox Incubator*, 2006.

[EQU06b] Eclipse Equinox's homepage, http://www.eclipse.org/equinox/. *Equinox*, 2006.

[EQU06c] Eclipse Equinox's homepage, http://www.eclipse.org/equinox/documents/http_quicl *Equinox Server-side Quickstart*, 2006.

[ESD05] http://help.eclipse.org/help31/index.jsp. *JDT Plug-in Developer Guide*, 2005.

[FAQ06] faq-o-matic.net, http://www.faq-o-matic.net/. *faq-o-matic.net*, 2006.

[Fre05] Jeffrey Fredrick. Headless hello world in eclipse. *Developer Testing, http://www.developertesting.com/archives/month200508/20050823-HeadlessHelloWorldInEclipse.html*, 2005.

[GNU06] GNU Classpath, http://www.gnu.org/software/classpath/. *GNU Classpath*, 2006.

[Her03] Jack Herrington. Is documentation holding open source back? *DevX.com, http://www.devx.com/opensource/Article/11839*, 2003.

[JAV06a] JavadocOnline, http://www.javadoconline.com. *JavadocOnline*, 2006.

[jav06b] Sun, http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/javadoc.html#{@link}. *Javadoc - The Java API Documentation Generator*, 2006.

[jav06c] Wikipedia, http://en.wikipedia.org/wiki/Javadoc. *Javadoc*, 2006.

[JCV06] JCVS's homepage, http://www.jcvs.org/. *JCVS*, 2006.

[JDO05] JDocs' homepage, http://www.jdocs.com/. *Jdocs*, 2005.

[JSO06] JSourcery's homepage, http://jsourcery.com/welcome/home.html. *JSourcery*, 2006.

[Kal06] Riyad Kalla. Developing eclipse/osgi web applications. *Eclipse Zone, http://eclipsezone.com/eclipse/forums/t64085.html*, 2006.

[KIC05] Kick Java's homepage, http://kickjava.com/. *Kick Java*, 2005.

[Lip06] Martin Lippert. Server-side eclipse. *Java Stuttgart Forum, http://www.java-forum-stuttgart.de/folien/D4_Lippert.pdf*, 2006.

[LKW06] Martin Lippert, Bernd Kolb, and Gerd Wuetherich. Next step eclipse. *Eclipse Magazine, http://www.eclipse-magazin.de*, 2006.

[Mar05a] David Patrick Marin. What motivates programmers to comment? *Technical Report No. UCB/EECS-2005-18 http://www.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-18.html*, 2005.

[Mar05b] Manoel Marques. Exploring eclipse's astparser. *IBM developerWorks, http://www-128.ibm.com/developerworks/opensource/library/os-ast/?ca=dgr-lnxw97ASTParser*, 2005.

[PS01] Nic Peeling and Julian Satchell. Analysis of the impact of open source software. *SEKE*, 2001.

[RRM04] A.J. Rostkowycz, V. Rajlich, and A Marcus. A case study on the long-term effects of software redocumentation. *1063-6773/04 20.00 . 2004 IEEE*, 2004.

[Twi06] TWiki, http://twiki.org/. *TWiki*, 2006.

[Whe05] David A. Wheeler. Why open source software / free software? look at the numbers! *David A. Wheeler¡¯s Personal Home Page, http://www.dwheeler.com/oss_fs_why.html*, 2005.

[wik06a] APC.org, http://www.apc.org/english/news/index.shtml?x=5038198. *Wiki part I, Progressive communications, wiki style*, 2006.

[wik06b] APC.org, http://www.apc.org/english/news/index.shtml?x=5038198. *Wiki part II, Wiki part II: Find your wiki at the bazaar*, 2006.

[wik06c] Wikipedia, http://en.wikipedia.org/wiki/Wiki. *Wiki*, 2006.

A Appendix

A.1 User Guides

WikiJavadoc Manual

Contents

---

1. Introduction

   WikiJavadoc is the open source JavaDoc-style in-line documentation systems with the online collaboration paradigm of WikiWiki. It is supplied by:

   Free University Berlin

   Institute of Computer Science

   Make sure that you have downloaded the latest version from the WikiJavadoc website from sourceforge.net:

   `http://sourceforge.net/projects/wikijavadoc`

   WikiJavaDoc may be used under the GNU General Public License, GPL.

2. System requirements

   WikiJavadoc and UpdateJavadoc make use of various JDK 5 features, so you need a JDK 5 or later. You need also a version control tool, e.g. Tortoise CVS or Tortoises SVN. For servlet container Tomcat 5.5 or later is required.

3. Starting

   - Download the pre-built web application archive wikijavadoc.war and copy the wikijavadoc.war file to the webapps directory in the tomcat installation directory.
   - Start Tomcat, your URL is the server's base with the /wikijavadoc/ context. For example, http://localhost:8080/wikijavadoc/.

4. Configuration

   For security reasons, using the administration webapp od WikiJavadoc is restricted to users with a role "wikidocAdmin". Users are defined in $CATALINA_HOME/conf/tomcat-users.xml. In tomcat-users.xml file you should add the following lines:

   <role rolename="wikidocAdmin"/>

   <user username="yourName" password="yourPassword" roles="wikidocAdmin"/>

   Of course the administrator of Tomcat is the default Administrator of WikiJavadoc.

5. Problems

   If you encounter any problems, you can ask in WikiJavadoc's forum from sourceforge.net. or if you find out bugs, please report it to WikiJavadoc's bugs tracker page. Of course you can also write mails to zhao@inf.fu-berlin.de

Your WikiJavadoc team.

A.2 For potential Developers

If you are interested in WikiJavadoc and want to join our developers team. You are very welcome. You can write mails to

oezbek@inf.fu-berlin.de. or zhao@inf.fu-berlin.de

We will add you as a number in our project in sourceforge.net.

A.2.1 File Release

You can download our newest release from

`http://sourceforge.net/project/showfiles.php?group_id=166216`

If you want to release a file, You have to login to sourge.net and go to the WikiJavadoc page.

Select ″Admin->Publicity->File Release″, then you come into the file release system of WikiJavadoc.

Under the file release system two packages named ″javawikidoc″ and ″updateJavadoc″ are created. One is for the project WikiJavadoc, the other is for the project UpdateJavadoc. If you want to release a new file for the project WikiJavadoc, you should generate a WAR file. If you want to realse a new file for the project UpdateJavadoc, you should generate a zip archive or tar file. And you can place the new file at the root of your hard drive, for easy access. Then follow these instructions to perform the upload:

1. FTP to upload.sourceforge.net
2. Login as "anonymous"
3. Use your e-mail address as the password for this login
4. Set your FTP client to binary mode.
5. Change your current directory to /incoming .

6. Upload the desired files for the release.

A.2.2 Code Management

A.2.2.1 With SVN You can also get source code of our project with SVN or CVS.

WikiJavadoc Subversion repository can be checked out through SVN with the following instruction set:

svn co https://svn.sourceforge.net/svnroot/wikijavadoc wikijavadoc

As a developer, if you want to execute other SVN operations, you need to enter your site password when prompted.

A.2.2.2 With CVS

A.2.2.2.1 Anonymous CVS Access This project's SourceForge.net CVS repository can be checked out through anonymous (pserver) CVS. When prompted for a password for anonymous, simply press the Enter key.

The repositorise:

cvs -d:pserver:anonymous@wikijavadoc.cvs.sourceforge.net:/cvsroot/wikijavadoc login

cvs -z3 -d:pserver:anonymous@wikijavadoc.cvs.sourceforge.net:/cvsroot/wikijavadoc co -P modulename

If you've never used SVN or CVS, you should read some documentation about them, useful URLs:

SVN: `http://sourceforge.net/docs/E09`

CVS: `http://sourceforge.net/docs/E04/`

A.2.2.2.2 Developer CVS Access via SSH Only project developers can access the CVS tree via this method. A SSH client must be installed on your client machine. Substitute modulename and developername with the proper values. Enter your site password when prompted.

The repository:

export CVS_RSH=ssh

cvs -z3 -d:ext:developername@wikijavadoc.cvs.sourceforge.net:/cvsroot/wikijavadoc co -P modulename

A.3 Writing Javadoc Comments

The commenting styles and Javadoc tags should be used when documenting source code. A Javadoc comment block always starts with /** and ends with */ . A Javadoc tag begins with an "@". An example of using Javadoc to document a method is given in Fig.33. In WikiJavadoc edit window the comments delimiters are avoided. Fig.44 illustrates an example of documenting in WikiJavadoc edit window.

There are two types of tags:

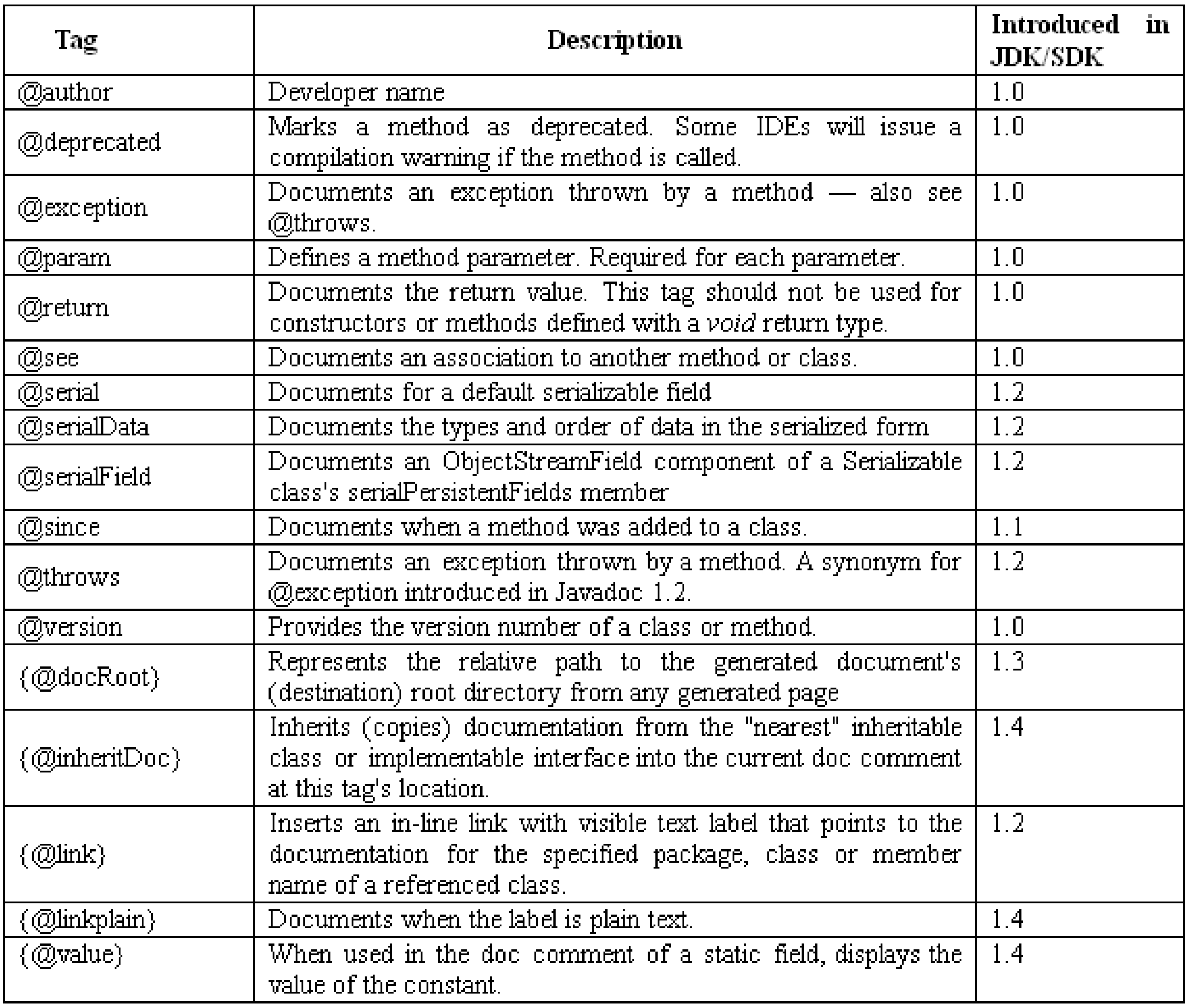
- Block tags

  Block tags can be placed only in the tag section that follows the main description.

- Inline tags

  Inline tags are denoted by curly braces, and can be placed anywhere in the main description or in the comments for block tags. [jav06b]

The following table provides the Javadoc tags, which are introduced in JDK/SDK 1.0-1.4, and are supported for by WikiJavadoc.



Tag	Description	Introduced in JDK/SDK
@author	Developer name	1.0
@deprecated	Marks a method as deprecated. Some IDEs will issue a compilation warning if the method is called.	1.0
@exception	Documents an exception thrown by a method — also see @throws.	1.0
@param	Defines a method parameter. Required for each parameter.	1.0
@return	Documents the return value. This tag should not be used for constructors or methods defined with a *void* return type.	1.0
@see	Documents an association to another method or class.	1.0
@serial	Documents for a default serializable field	1.2
@serialData	Documents the types and order of data in the serialized form	1.2
@serialField	Documents an ObjectStreamField component of a Serializable class's serialPersistentFields member	1.2
@since	Documents when a method was added to a class.	1.1
@throws	Documents an exception thrown by a method. A synonym for @exception introduced in Javadoc 1.2.	1.2
@version	Provides the version number of a class or method.	1.0
{@docRoot}	Represents the relative path to the generated document's (destination) root directory from any generated page	1.3
{@inheritDoc}	Inherits (copies) documentation from the "nearest" inheritable class or implementable interface into the current doc comment at this tag's location.	1.4
{@link}	Inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class.	1.2
{@linkplain}	Documents when the label is plain text.	1.4
{@value}	When used in the doc comment of a static field, displays the value of the constant.	1.4

Figure 39: Javadoc Block Tags in JDK/SDK 1.0-1.4

A.4 Creating a WAR-File

The WikiJavadoc project is a web application. To ease the work of users, it should be exported into a ”runnable” WAR file. There are a few ways to create WAR files.

- With Ant script

  You can write an ant script to help you.

  Sun Microsystems has a technical article on how to create a WAR file.

  `http://access1.sun.com/techarticles/simple.WAR.html`

- With Lomboz Plug-in

  If you do not know Ant or have not time to learn how to write Ant script, you can use tools to do that. If you is using Eclipse there are several plug-ins available. One of them called Lomboz is a nice plug-in, which you can download from

  `http://sourceforge.net/projects/lomboz/`

- With Sysdeo Tomcat Plug-in

  If your web application is not a J2EE-Web project, you can use Sysdeo Eclipse Tomcat Launcher plug-in. WikiJavadoc is such a Tomcat plug-in project and is exported into a WAR file with this method. And it’s also the easiest way.

  1. Install Sysdeo Tomcat plugin

     You can download Sysdeo Eclipse Tomcat Launcher plug-in from

     `http://www.sysdeo.com/eclipse/tomcatplugin`.

     Extract zip file to a temporary directory and copy the directory com.sysdeo.eclipse.tomcat_ to Exlipse’s plugins dictionary or directly extract zip file to Exlipse’s plugins dictionary.

2. Configure Eclipse and set the Tomcat plug-in preferences. Select ″Window -> Preferences -> Tomcat″, then check Version 5.x, then click Browse to select Tomcat home.

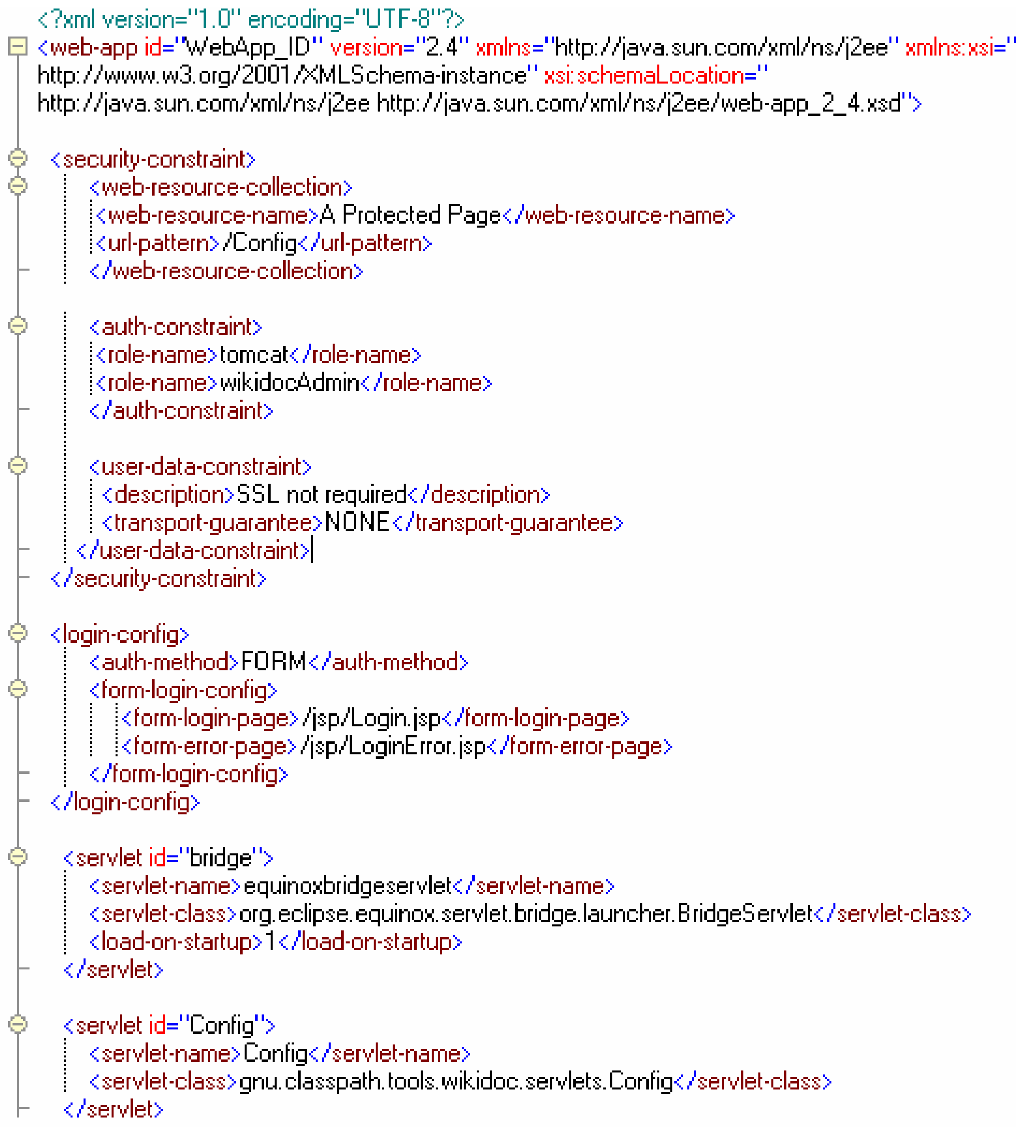
3. Create Tomcat Project

   With Eclipse a new project with a WAR structure is created in the workspace. Make sure your project has no problem to deploy on the local Tomcat. Then Select the Tomcat project name, right-click mouse, then select ″Properties ->Tomcat-> Attitude for an Export in WAR-File″ , and give the war file local path and name.

4. Export a WAR file. Select Tomcat project name, right-click mouse, then select ″Tomcat Project->Export...War-File″

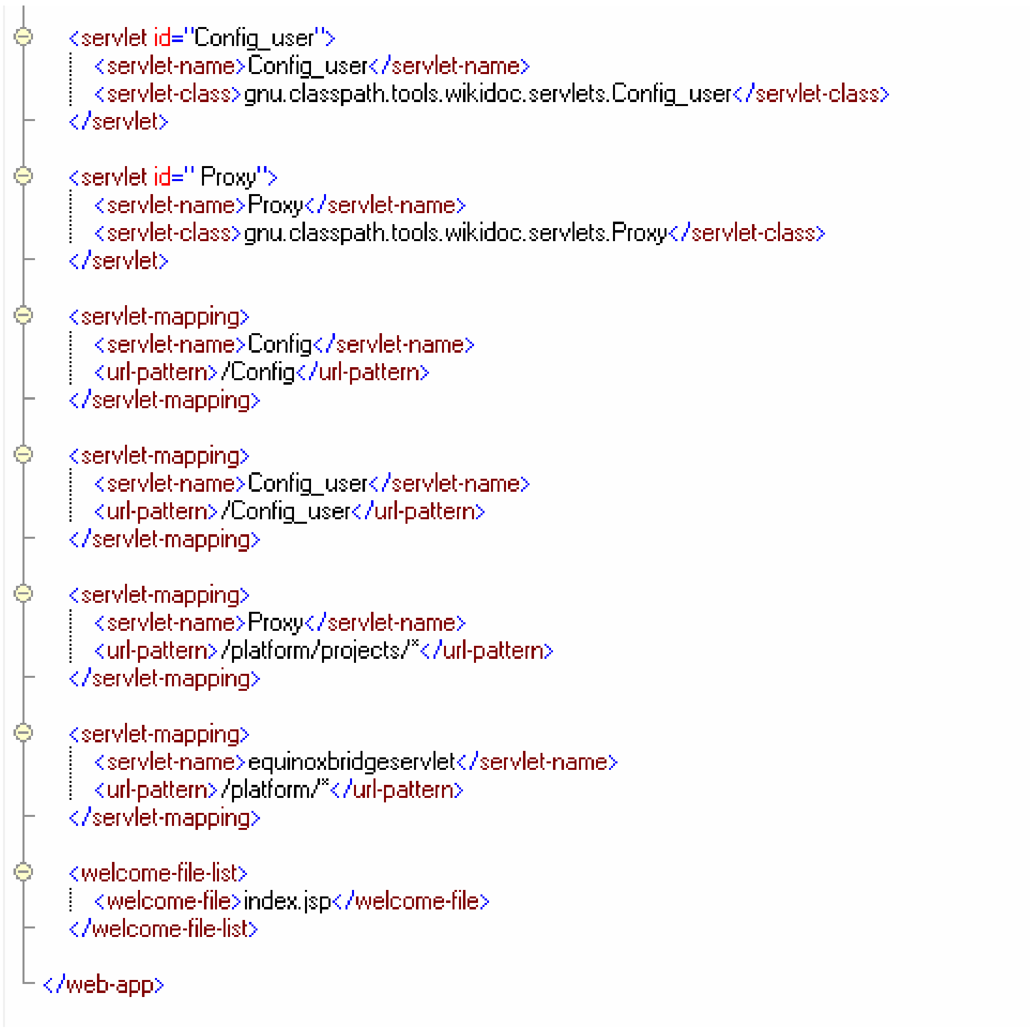
A.5 Important Files of WikiJavadoc

The Deployment Descriptor, namely Web.xml in the WikiJavadoc project.



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

 <security-constraint>
    <web-resource-collection>
     <web-resource-name>A Protected Page</web-resource-name>
     <url-pattern>/Config</url-pattern>
    </web-resource-collection>

    <auth-constraint>
     <role-name>tomcat</role-name>
     <role-name>wikidocAdmin</role-name>
    </auth-constraint>

    <user-data-constraint>
     <description>SSL not required</description>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
 </security-constraint>

 <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/jsp/Login.jsp</form-login-page>
      <form-error-page>/jsp/LoginError.jsp</form-error-page>
    </form-login-config>
 </login-config>

  <servlet id="bridge">
    <servlet-name>equinoxbridgeservlet</servlet-name>
    <servlet-class>org.eclipse.equinox.servlet.bridge.launcher.BridgeServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet id="Config">
    <servlet-name>Config</servlet-name>
    <servlet-class>gnu.classpath.tools.wikidoc.servlets.Config</servlet-class>
  </servlet>
```

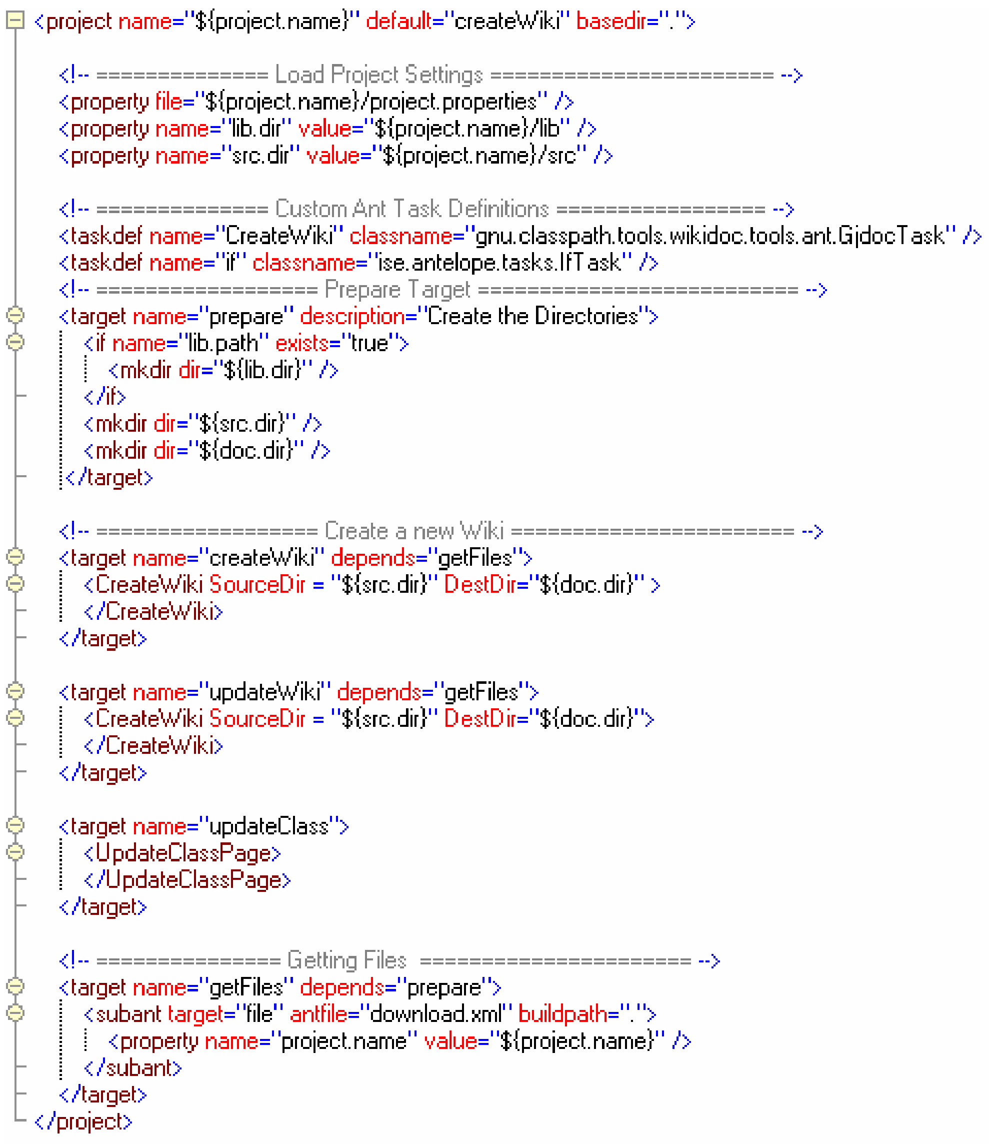
Figure 40: web.xml File of Wikijavadoc project I



```
<servlet id="Config_user">
    <servlet-name>Config_user</servlet-name>
    <servlet-class>gnu.classpath.tools.wikidoc.servlets.Config_user</servlet-class>
</servlet>

<servlet id=" Proxy">
    <servlet-name>Proxy</servlet-name>
    <servlet-class>gnu.classpath.tools.wikidoc.servlets.Proxy</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Config</servlet-name>
    <url-pattern>/Config</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Config_user</servlet-name>
    <url-pattern>/Config_user</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Proxy</servlet-name>
    <url-pattern>/platform/projects/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>equinoxbridgeservlet</servlet-name>
    <url-pattern>/platform/*</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

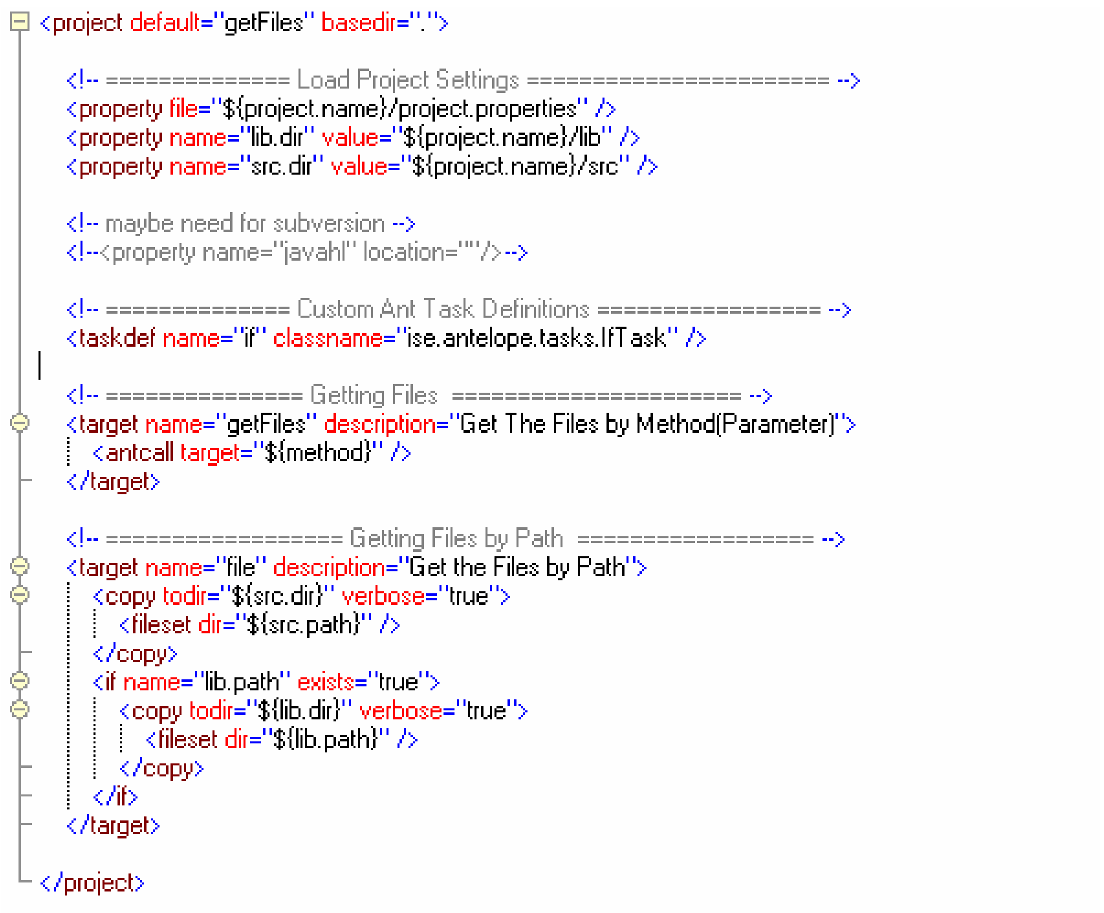
Figure 41: web.xml File of Wikijavadoc project II

The build.xml and downloa d.xml files are used in Ant script:



```
<project name="${project.name}" default="createWiki" basedir=".">

    <!-- ============== Load Project Settings ======================= -->
    <property file="${project.name}/project.properties" />
    <property name="lib.dir" value="${project.name}/lib" />
    <property name="src.dir" value="${project.name}/src" />

    <!-- ============== Custom Ant Task Definitions ================= -->
    <taskdef name="CreateWiki" classname="gnu.classpath.tools.wikidoc.tools.ant.GjdocTask" />
    <taskdef name="if" classname="ise.antelope.tasks.IfTask" />
    <!-- ================== Prepare Target ========================== -->
    <target name="prepare" description="Create the Directories">
        <if name="lib.path" exists="true">
            <mkdir dir="${lib.dir}" />
        </if>
        <mkdir dir="${src.dir}" />
        <mkdir dir="${doc.dir}" />
    </target>

    <!-- ================== Create a new Wiki ======================= -->
    <target name="createWiki" depends="getFiles">
        <CreateWiki SourceDir = "${src.dir}" DestDir="${doc.dir}" >
        </CreateWiki>
    </target>

    <target name="updateWiki" depends="getFiles">
        <CreateWiki SourceDir = "${src.dir}" DestDir="${doc.dir}">
        </CreateWiki>
    </target>

    <target name="updateClass">
        <UpdateClassPage>
        </UpdateClassPage>
    </target>

    <!-- =============== Getting Files  ====================== -->
    <target name="getFiles" depends="prepare">
        <subant target="file" antfile="download.xml" buildpath=".">
            <property name="project.name" value="${project.name}" />
        </subant>
    </target>
</project>
```

Figure 42: build.xml File of Ant Script



```
<project default="getFiles" basedir=".">

    <!-- ============== Load Project Settings ======================== -->
    <property file="${project.name}/project.properties" />
    <property name="lib.dir" value="${project.name}/lib" />
    <property name="src.dir" value="${project.name}/src" />

    <!-- maybe need for subversion -->
    <!--<property name="javahl" location=""/>-->

    <!-- ============== Custom Ant Task Definitions ================= -->
    <taskdef name="if" classname="ise.antelope.tasks.IfTask" />

    <!-- =============== Getting Files  ====================== -->
    <target name="getFiles" description="Get The Files by Method(Parameter)">
        <antcall target="${method}" />
    </target>

    <!-- ================== Getting Files by Path  ================== -->
    <target name="file" description="Get the Files by Path">
        <copy todir="${src.dir}" verbose="true">
            <fileset dir="${src.path}" />
        </copy>
        <if name="lib.path" exists="true">
            <copy todir="${lib.dir}" verbose="true">
                <fileset dir="${lib.path}" />
            </copy>
        </if>
    </target>

</project>
```

Figure 43: download.xml File of Ant Script