Wireless messaging and project management system

Bhrigu Celly
WIRELESS MESSAGING AND PROJECT MANAGEMENT SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Bhriku Celly
June 2002
WIRELESS MESSAGING AND PROJECT MANAGEMENT SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Bhrigu Celly
June 2002

Approved by:

Dr. Ernesto Gomez, Advisor, Computer Science

Dr. David Turner

Dr. Richard Botting

June 3rd, 2002

Date
ABSTRACT

Wireless is an important part of the future and utilizing the wireless technology in project management is a great tool. Almost every large project needs some distant tasks to be performed. The use of wireless technology can help speed up feedback for such projects and hence help in better management of the project. This project is a messaging system designed using WAP and Java Server Pages, as both of them are technologies of the future.

This project uses the above concept to put together a tool to help in project management. This tool has been made with the use of Java Server Pages and MySQL as the backend database. The project uses Wireless Markup Language as the basic language for the cellular phones and the hand held wireless devices. This tool can be used in a number of applications. It is a tool for managers to manage their workers.

This prototype has been modeled on project planning for power plants done by Asea Brown Boveri, Ltd., the power plant manufacturing giants. The design methodology and the reporting of tasks were followed there. The hierarchy of people doing the tasks has also been taken from there.
This is an important tool for the future generations to build on and use. It is not application dependent so can be used in a number of applications.
ACKNOWLEDGMENTS

I would like to thank Dr. Richard Botting and Dr. David Turner for serving on my committee and their insightful comments on the draft of this report. I am particularly grateful to Dr. Ernesto Gomez, my project advisor, for his patience, advice, guidance, detailed suggestions and corrections of the design, implementation, and report of this project.

I would also like to thank Mr. Munish Mehra of e-Promentor for getting me started on this project and Marcia Guerrero for her hardware support for the system.

The support of the National Science Formation under the award 9810708 is gratefully acknowledged.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................. iii

ACKNOWLEDGMENTS .................................................................................................. v

LIST OF TABLES .......................................................................................................... vii

LIST OF FIGURES ........................................................................................................ vi

CHAPTER ONE: SOFTWARE REQUIREMENT SPECIFICATION

1.1 Introduction .............................................................................................................. 1

1.2 Purpose of the Project ............................................................................................ 1

1.3 Context of the Problem .......................................................................................... 2

1.4 Significance of the Project ..................................................................................... 2

1.5 Assumptions ........................................................................................................... 2

1.6 Limitations ............................................................................................................. 5

1.7 Definition of Terms ............................................................................................... 6

1.8 Organization of the Thesis .................................................................................... 15

CHAPTER TWO: SOFTWARE DESIGN

2.1 Introduction ........................................................................................................... 17

2.2 Preliminary Design ............................................................................................... 17

2.3 Detailed Design .................................................................................................... 19

2.4 Setting Up ............................................................................................................. 25

2.5 Summary .............................................................................................................. 26

CHAPTER THREE: SOFTWARE QUALITY ASSURANCE

3.1 Introduction ........................................................................................................... 28

3.2 Unit Test Plan ....................................................................................................... 28

3.3 Integration Test Plan ............................................................................................ 30

3.4 System Test Plan ................................................................................................. 33

vi
LIST OF TABLES

Table 1. Definition of Terms ..................................... 6
LIST OF FIGURES

Figure 1. Assumptions and Limitations .................. 5
Figure 2. Preliminary Design ......................... 17
Figure 3. Bean Deployment .......................... 19
Figure 4. The Structure of the Application .......... 22
Figure 5. Database Server .......................... 23
Figure 6. EmailEngine.java Class Diagram .......... 24
Figure 7. Database Schema .......................... 25
Figure 8. Directory Structure ....................... 39
Figure 9. Login Screen ............................. 43
Figure 10. Registration Page ......................... 44
Figure 11. Project Supervisor Screen .............. 45
Figure 12. Project Administration Screen .......... 46
Figure 13. User Rights Edit Screen ................. 47
Figure 14. Project Create Screen ................... 48
Figure 15. New Project Created ..................... 49
Figure 16. Task Display Screen ..................... 50
Figure 17. Task Create Page ......................... 51
Figure 18. Task Administration Page ............... 52
Figure 19. Node Display Page ....................... 53
Figure 20. Adding Node Page ......................... 54
Figure 21. Project Overview Screen ................. 55
Figure 22. Wireless Login Page ..................... 56
Figure 23. Project Screen .......................... 57
Figure 24. Task Selection Screen ................... 58
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Node List Screen</td>
<td>59</td>
</tr>
<tr>
<td>26</td>
<td>Node Update Screen</td>
<td>60</td>
</tr>
<tr>
<td>27</td>
<td>Nokia Mobile Browser 3.0</td>
<td>70</td>
</tr>
<tr>
<td>28</td>
<td>Nokia Mobile Toolkit Simulator</td>
<td>71</td>
</tr>
<tr>
<td>29</td>
<td>Alcatel Phone Simulator (Alav)</td>
<td>72</td>
</tr>
<tr>
<td>30</td>
<td>Ericsson Phone Simulator</td>
<td>73</td>
</tr>
<tr>
<td>31</td>
<td>Motorola Phone Simulator (Imlk)</td>
<td>74</td>
</tr>
<tr>
<td>32</td>
<td>Motorola Phone Simulator (Mtp1)</td>
<td>75</td>
</tr>
<tr>
<td>33</td>
<td>Openware Simulator Phone (Owgl)</td>
<td>76</td>
</tr>
<tr>
<td>34</td>
<td>Samsung Simulator Phone (Sh)</td>
<td>77</td>
</tr>
<tr>
<td>35</td>
<td>Mitsubishi Simulator Phone (Spol)</td>
<td>78</td>
</tr>
<tr>
<td>36</td>
<td>Mitsubishi Simulator (T250)</td>
<td>79</td>
</tr>
</tbody>
</table>
CHAPTER ONE
SOFTWARE REQUIREMENT SPECIFICATION

1.1 Introduction

The Project Software Requirements Specification (SRS) provides a detailed introduction to the architecture of the messaging system and decision-making system. This is a messaging system designed using WAP and Java Server Pages as both of them are technologies of the future. This tool is to make project management easier and feedback more organized. It is also supposed to help people to perform different tasks which they have not performed before.

1.2 Purpose of the Project

The purpose of the project is to create a prototype of the system. The software product defined in this document is to be known as WMS (Wireless Messaging and Project Management System). This is a tool that would be used for assignment of work to people who are not aware of what to do or basically describing a task list to a worker. This is to make it easier for a project manager to manage his project and to keep track of all the work done.
1.3 Context of the Problem

The problem of lack of coordination has been there for years and this tool is trying to address the problem. This also addresses the problem of lack of instructions too. In a way it acts as a guide tool to accomplish tasks.

1.4 Significance of the Project

The WMS system makes use of the current technologies to increase reliability and efficiency in the creation and execution of a real-life project that involves a group working in various physical locations.

1.5 Assumptions

The following assumptions were made regarding the project:

1. Each Project is composed of Tasks and each Task is composed of Nodes. A Node is a subtask with another task. It will have a database that will store all the information about the person defining the tasks (Maker) and the person using the task list (User). An example for a project is a full power plant turnkey project and tasks in them would be smaller sub parts. These subparts have various stages of making and they would constitute as nodes for the project.
2. There is a GUI that is available in the form of JSP-generated web pages on the Internet, so that the Maker can login and create the task list. The person can only create a task list in an already created project.

3. The system has three kinds of users; (i) Supervisors (ii) Makers, and (iii) Users. The Supervisor has the maximum rights. Supervisor has all privileges to modify and create all levels in the system. He can create a Project and also edit a project. He is also responsible for assigning the users and makers. The Person who creates a project is the supervisor by default. He can add supervisors or upgrade a user or a maker to the level of a supervisor. A supervisor also has the option to reduce his rights to a maker or a user, and appoint another user as a supervisor. A supervisor has the right to delete projects, tasks and nodes. A maker can delete tasks and nodes. As a project is deleted the corresponding tasks and nodes are deleted with it.

4. A prospective Maker has to enter his data and login over the Internet and make the task list that will be stored by the system. Maker can also use the list with the same functionality as the user.
5. The Maker can define the number of tasks and the attributes associated with it, i.e. the work associated with each task. The supervisor can also do the defining.

6. The Maker, user and supervisor all can use the wireless interface, but they will have the rights of a user only.

7. The Project including its tasks and nodes are automatically saved as they get changed.

8. An Overview can be viewed in the HTML browser. Only the Supervisor can do this. An overview shows the number of tasks completed and the number left to be completed. Also, it shows the comments the user made when he was performing the tasks.

9. There can be more then one maker and user per project. User, maker and supervisor can access and update a project on a wireless device

10. The users have to register before they can be granted access privileges to a project. Any registered user can create a project.
During the development of the project, a number of limitations were noted. The following limitations apply to the project:

1. The project can only be updated on the wireless devices. All the users have the same access rights on the wireless interface.
2. Wireless clients cannot make projects, tasks or nodes.
3. There has to be a minimum of one supervisor per project.
4. No repetition or selection of tasks.
### 1.7 Definition of Terms

The following terms are defined as they apply to the project.

<table>
<thead>
<tr>
<th>TERM</th>
<th>DEFINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAP Device</td>
<td>This term indicates the physical device that you use to access WAP applications and content. It doesn't necessarily have to be a mobile phone. It might be a PDA or a handheld computer. More generally, it's every WAP compliant device.</td>
</tr>
<tr>
<td>WAP Client</td>
<td>In a network environment, a client is typically the logical entity that is operated by the user and communicates with the 'server entity'. In the WAP world, the client is</td>
</tr>
</tbody>
</table>
the entity that receives content from the Internet via a WAP Gateway. This is usually (but not necessarily) the WAP browser. Commonly, 'WAP client' and 'WAP browser' are often used interchangeably.

WAP Browser

This is software running on the WAP device that interprets the WAP content arriving from the Internet and decides how to display it on the screen of the WAP device. WAP browsers are available for all WAP devices, and are frequently referred to as Micro browsers. There are also emulators available for some browsers, which run on PCs.
<table>
<thead>
<tr>
<th>User Agent</th>
<th>An agent is normally the software that deals with protocols, and WAP is no exception to this. The WAP client contains two different agents: the WAE User Agent and the WTA User Agent (each of which will be covered later in the chapter).</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAP Gateway</td>
<td>This is the element that sits (logically) between the WAP device and the origin server. It acts as an 'interpreter' between the two, enabling them to communicate. It usually resides within the operator network, but you can also install your own gateway, as we will see later. Unless otherwise stated, when a gateway is</td>
</tr>
</tbody>
</table>
If discussed, we mean a gateway residing in the operator network, since this is the more common situation that one encounters.

<table>
<thead>
<tr>
<th>Network Operator</th>
<th>This is the company or organization that provides carrier services to its subscribers. As an example, the company you are paying your telephone bills to is your network operator. A network operator enables you to make calls to other phones from your telephone and, in addition, provides you with different services, such as voice mail, call diversion etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bearer Services</td>
<td>These are the different ways that a mobile phone</td>
</tr>
</tbody>
</table>
can communicate with the wireless network. To send and receive data from an application server, mobile phones have to establish some sort of connection with the WAP gateway. A bearer service is the method they use to do this. In GSM networks, for example, we either use SMS (Short Message Service) or CSD (Circuit Switched Data). With the former bearer, the gateway has to divide the information that is to be sent to the phone into a lot of little messages (just like when you send a text message to a friend using your mobile). With CSD, we communicate with the
<table>
<thead>
<tr>
<th><strong>Content/Origin/Application Server</strong></th>
<th>gateway using a data connection, which is not dissimilar to the way the modem in your computer communicates with the Internet Service Provider that you have an account with.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>They denote the element that hosts the Internet content that is sent to clients when they make a request for it. A web server is an origin server, providing HTML content; but also WAP content if properly configured.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>The WAE User Agent</strong></td>
<td><strong>The WAE User Agent</strong> (Wireless Application Environment User Agent) is the micro browser that renders the content for</td>
</tr>
</tbody>
</table>
display. It receives the compiled WML, WMLScript, and any images from the WAP gateway, and executes or displays them on the screen. Even if the implementation details are left to the vendor, the browser must implement all the function provided by WML and WMLScript. It must also manage the interaction with the user, such text input, and error or warning messages.

<p>| WTA User Agent | The WTA User Agent (Wireless Telephony Applications User Agent) receives compiled WT files from the WTA server and executes them. The WTA User Agent includes access to the interface to the |</p>
<table>
<thead>
<tr>
<th>WAP Stack</th>
<th>The WAP Stack implementation allows the phone to connect to the WAP gateway using the WAP protocols.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy</td>
<td>This is an intermediary element, acting both as a client and as a server in the network. It is located between clients and origin servers; the clients send requests to it and it retrieves and caches the information needed by contacting the origin servers.</td>
</tr>
<tr>
<td>Gateway</td>
<td>This is an intermediary element usually used to connect two different types of network. It receives requests directly from the clients as if it actually were the origin server that the clients want to retrieve the information from. The clients are usually unaware that they are speaking to the gateway.</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WBMP</td>
<td>Wireless Bitmap</td>
</tr>
<tr>
<td>WTP</td>
<td>Wireless Transaction Layer</td>
</tr>
<tr>
<td>WSP</td>
<td>Wireless Session Protocol</td>
</tr>
<tr>
<td>WTLS</td>
<td>Wireless Transport Layer Security</td>
</tr>
<tr>
<td>WDP</td>
<td>Wireless Datagram Protocol</td>
</tr>
<tr>
<td>Maker</td>
<td>The person defining the Task List</td>
</tr>
<tr>
<td>User</td>
<td>The Person using the Task List</td>
</tr>
<tr>
<td>List</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td></td>
</tr>
<tr>
<td>The Task List for a</td>
<td></td>
</tr>
<tr>
<td>particular application.</td>
<td></td>
</tr>
<tr>
<td>WAP</td>
<td></td>
</tr>
<tr>
<td>Wireless Application</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>WML</td>
<td></td>
</tr>
<tr>
<td>Wireless Markup Language</td>
<td></td>
</tr>
<tr>
<td>WTAI</td>
<td></td>
</tr>
<tr>
<td>Wireless Telephony</td>
<td></td>
</tr>
<tr>
<td>Application Interface</td>
<td></td>
</tr>
<tr>
<td>Project Manager</td>
<td></td>
</tr>
<tr>
<td>The person controlling the</td>
<td></td>
</tr>
<tr>
<td>project. The supervisor in</td>
<td></td>
</tr>
<tr>
<td>most cases.</td>
<td></td>
</tr>
</tbody>
</table>

1.8 Organization of the Thesis

The thesis portion of the project was divided into six chapters. Chapter One provides the software requirements specification, an introduction to the context of the problem, purpose of the project, significance of the project, limitations and delimitations and definitions of terms. Chapter Two consists of the software design. Chapter Three documents the steps used in testing the project. Chapter Four presents the maintenance required from the project. Chapter Five presents the users manual from the project. Chapter six presents conclusions drawn
from the development of the project. Project references follow Chapter Five. The Appendices for the project consists of: Appendix A Database table creation script; Appendix B Phone Models used in Testing; Appendix C Class Description and finally, the project references.
CHAPTER TWO
SOFTWARE DESIGN

2.1 Introduction

This system is a JSP based system. The front-end is based on JSP, and the backend is a MySQL database. This system has a set of beans in the backend that connect to the database with a native JDBC connection. The beans are responsible for all the data connections. The beans are used with JSP to create the Project Making interface. The WAP interface is written in JSP with WML content type. All the processing is done on the server side.

2.2 Preliminary Design

![Diagram of Preliminary Design](image)

Figure 2. Preliminary Design

User interfaces will be established on the web, and therefore it can use all user interfaces provided by the web browser, including all plug-ins and any added
functionality that the browser [3] may possess (e.g., Adobe Acrobat Reader plug-in for web browsers). A number of different interfaces will be available depending on who the user is. These are detailed as follows. The home page will be the starting page for all people who are going to be using this software product. It will show all required logos, such as that of WMS, CSUSB and the CS Dept. It will enable the user to select from amongst all the options available. This page will also allow the user to LOGIN so that we can fix the access rights and identity of the user as one of the following (in order of increasing access levels):

- Supervisor
- Maker
- User

The login is going to be the same for the supervisor, user and maker but different links will appear for each of them. The Supervisor has extra links for project overview, deleting projects, and granting access rights and adding users.

The wireless interface is only a user interface. All the rights of all are reduced to user level rights when they access the projects through the wireless interface.
The above diagram explains the deployment for the system. In the system the JSP and the bean is used. The EJB concept could not find much use in the project.

2.3 Detailed Design

JSP is now an integral part of developing web based applications using Java. Because of its ability to separate presentation from implementation logic by combining standard markup with scripting elements and object oriented components, JSP provides an excellent front end technology for application that are deployed over the web.

A key element of the J2EE Application Model is the use of the web as a preferred, mechanism for data delivery between the application and the end user, relying on the
web browser as a primary user interface for enterprise software.

A prototypical web application could be composed of:

- Java Runtime Environments) running in the server (required).
- JSP pages, that handle requests and generate dynamic content.
- Servlets, that handle request and generate dynamic content.
- Server-side Java Bean components that encapsulate behavior and state.
- Static HTML, DHTML, XHTML, XML and similar pages.
- Client-side Java Applets, Java Bean components, and arbitrary Java class files.
- Java Runtime Environments downloadable via the plugin (running in clients).

JSP pages can be used in combination with Servlets, HTTP, HTML, XML, Applets, Java Bean components [4] and Enterprise Java Bean components to implement a broad variety of application architectures, or models.

JSP offers several benefits as a system for dynamic content generation. As a Java-based technology, it enjoys
all of the advantages that the Java language provides with respect to development and deployment. As an object-oriented language with strong typing, encapsulation, exception handling, and automatic memory management, use of Java leads to increased programmer productivity and more robust code. Because compiled Java byte code is portable across all platforms that support a JVM, [6] use of JSP does not lock us into using a specific hardware platform, operating system, or server software. If a switch in any of these components becomes necessary, all JSP pages and associated Java classes can be migrated over as is. Because JSP is vendor-neutral, developers and system architects can select best of breed solutions at all stages of JSP deployment. JSP technology is the Java platform technology for building applications containing dynamic web content such as HTML, DHTML, XHTML, and XML. The Java Server Pages technology enables the authoring of web pages that create dynamic content easily but with maximum power and flexibility.

The Java Server Pages technology is platform independent, both in its dynamic Web pages, its Web servers, and its underlying server components. We can author JSP pages on any platform, run them on any Web server or Web enabled application server, and access them
from any web browser. We can also build the server components on any platform and run them on any server.

WML was used as the basic markup language for the wireless phones. It is a standard and has been followed for some time. All the design has been done in WML 1.1 and is compatible with the cellular phones at this present time.

Figure 4. The Structure of the Application

This design allows for a continuing conversation between the client, the main JSP, the main bean, [6] and output JSPs. The idea is that a JSP functioning as an output template contains a form (or a \texttt{<go>} element) whose
action attribute is the URL of the main JSP. The main INDEX.jsp page acts as a switchboard to which all requests are sent, and which forwards them to the right output JSP.

The beans composed of many classes these classes have been shown in the class diagram in Figure 5.

Figure 5. Database Server
Figure 6 is the email server that has been used as the class to send emails. This has the SMTP host name. If the server changes the name also has to be changed.
Figure 7 is the database schema for the backend of the schema. This consists of five tables.

### 2.4 Setting Up

Assuming Java middleware and MySQL as the database layer, the software components we need to perform setup operations are:

- Any database that has a JDBC driver. There is also a JDBC-ODBC bridge that comes with the JDK; you can use any database that has an ODBC driver with it, but not in production environments. In this case the native JDBC downloaded from the mysql website was used.
JDBC is a part of Java 2 Standard Edition (J2SE), JDK 1.2.2

Servlet/JSP engine is the most common combinations are Apache + Tomcat. The one used in this case was Tomcat by itself, as it has a built-in web server. This is what was used for the web application server.

Client browser that was used was UP Phone SDK browser and Nokia toolkit browser.

Nokia Active Server was used for the WAP gateway.

2.5 Summary

The main point of this part has been to bring up several quite general design ideas, and to create a framework for generating both XHTML and WML content from the same data source.

In the actual application we looked at, the data source is a relational database, but it doesn't have to be: the framework is general enough that anything wrapped in a Properties object or described by an XML document can be a source of data. In this case it was the MySQL data source. The web server supplies the web browser HTML content and the wireless browser WML content.
The system is running on java beans that make connection to the database using JDBC. The beans are used in the JSP script and they do all the transaction processing. This section was the description to the architecture for the system.
3.1 Introduction

This document will provide overview, test plan, and results of the integration testing of the Wireless Messaging and Project Management System (WMS). The system has to be tested for working on the wireless as well as working on the browser. This section explains the testing procedures followed.

3.2 Unit Test Plan

The WMS is a web-based system that is designed to provide information to a user to help him perform daily tasks. It also helps people to monitor if the tasks have been done. Users are provided easy access to create and manage projects so that other people can use them. It is also designed for a supervisor (the project creator) to keep a tab on the activities so that he can keep account of the project from his desktop. The project users have the option to use the system by using a WAP Browser or a HTML Browser (Netscape or Internet Explorer). The WAP browser has the option only to view the project and to fill in the status. The WAP browser can be used by any of the Makers, Supervisors or the Users. If the Maker
accesses it through a WAP browser then he just can use the project as user-level rights (only being able to modify status and not being able to build on it or edit it). Any WAP based browser can be used to access the system. The WAP browsers include cell phones and hand held devices.

The web-based system will provide the following functions:

- Project Making
- Project Saving
- Project Editing
- Project Updating
- User Right Editing
- Project Overview for Supervisory Level
- Task Making
- Task Saving
- Task Editing
- Task Updating
- Task Description Updating
- Node Making
- Node Saving
- Node Editing
- Node Updating
The hand held systems that will be used with the cellular phones and other wireless devices will have the following functions:

- Project Browsing
- Task Browsing
- Node Browsing
- Node Updating

3.3 Integration Test Plan

The integration test plan will test how the web sites are linked together. Does each hyperlink of one web page correctly go the intended website? This will test how the various units of WMS web system work together. Test will be conducted as such. The hyperlink will be clicked on and then checked to see if the link goes to the proper web site or if an error has occurred. There will tests of different browsers with a multiple of users. Testing on various hand held and WAP Phone devices will be done using simulators from www.phone.com and from the Nokia website, as they are the two major browser manufacturers for the cellular phone industry. The uniformity of the interface is to be tested too, so that it can be used over various systems and across different cellular and hand held devices.
This is the name of the web site and the test that will be performed.

- WMS [Home]
- WMS Login Page (Linked from WMS [Home])
- Projects Display Page (Linked from WMS Login Page)
- Project Administration Page (Linked from Projects Display Page)
- WMS Login Page (Linked from Project Display Page)
- Project Overview Page (Linked from Projects Display Page)
- Project Delete Page (Linked from Projects display Page)
- Project Display Page (Linked from Task Display Page)
- Task Display Page (Linked from Projects Display Page)
- Add User Page (Linked from Project Administration Page)
- Delete User Page (Linked from Project Administration Page)
• Edit User Page (Linked from Project Administration Page)

• Task Administration Page (Linked from Tasks Display Page)

• WMS Login Page (Linked from Task Display Page)

• Task Delete Page (Linked from Task Display Page)

• Node Display Page (Linked from Task Display Page)

• Node Delete Page (Linked from Node Display Page)

• Predecessor Edit Page (Linked from Node Display Page)

• Successor Edit Page (Linked from Node Display Page)

• Status Edit Page (Linked from Node Display Page)

• Comments Edit Page (Linked from Node Display Page)

• Task Display Page (Linked from Node Display Page)

• Project Display Page (Linked from Node Display Page)

• Node Create Page (Linked from Node Display Page)

• WMS Login Page (Linked from Node Display Page)
3.4 System Test Plan

This document will provide overview, test plan, and results of the System testing of the Wireless Messaging and Project Management System (WMS).

The WMS is a web-based system to provide project management facilities to many firms so that they can manage and monitor their projects. For information see the SRS for WMS. The web-based system will provide the following functions:

- Hyperlink
- Fields
- Graphics
- Text - Information
- Print
- Submit
- Wireless Interface

The system test will test for the following criteria as needed:

Hyperlink: The hyperlinks will be checked a number of times to see if links are working consistently.

Content: The web page will be entered in and out a number of times. Each time the web page should be consistent.
Consistent Look and Feel: The web page will be entered in and out a number of times. Each time the web page should be consistent and easy to navigate.

Performance: The web page will be entered in and out a number of times. Each time the web page should download in about five to 10 seconds.

Print: The hardcopy of the web page should fit in a letter sized paper and split in a logical manner. Also all the typed fields should be in there.

Fields: The fields of the web page should do the following. Information should be entered, saved, and cleared. This will be done a number of times to check for consistency.

Submit: Project Planners should be able to save the data and have it entered into the database. This will be done a number of times.

Wireless Interface: the interface should update automatically as the user updates it from the web browser.

3.5 Summary

The testing plan deals with the integration of the web system for the maker, supervisor and the user to the system on the hand held or the cellular phone interfaces.
Also the data storage and retrieval is an important issue that was handled by the testing and integration plan.
CHAPTER FOUR
MAINTENANCE

4.1 Introduction

The Software Maintenance Manual (SMM) provides the procedures for executing the intermediate maintenance for the WMS system. This tells future users how the system is managed and how to make changes to it.

4.2 Instructions for Use

Given below is the directory structure for the Webmaster who will maintain the WMS system in the future. This document will enable the Webmaster to immediately react in case of a system failure. The file locations given in this document would help the Webmaster in not only detecting, but also upgrading specific files.

The WMS main directory is the parent directory and contains the subdirectories for the various forms, information pages for WMS, and other HTML and JavaScript pages. This contains the basic login pages for the system.

The user sub directory contains the user data storage and retrieval pages. Its sub-directories contain documents in JSP, HTML and JavaScript formats. The directory also contains the forms and the action scripts for the storage and retrieval of user information.
The project sub-directory contains the project data storage, project administration and project creation pages. Its sub-directories contain documents in JSP and JavaScript formats. The directory also contains the forms and the action scripts for the storage and retrieval of project creation and administration information.

The task sub directory contains the task data storage, task administration and task creation pages. Its sub-directories contain documents in JSP and JavaScript formats. The directory also contains the forms and the action scripts for the storage and retrieval of task creation and administration information.

The node Sub Directory contains the node data storage, node administration and node creation pages. Its sub-directories contain documents in JSP and JavaScript formats. The directory also contains the forms and the action scripts for the storage and retrieval of node creation and administration information.

The WML Sub Directory contains the cellular phone and the hand held interface scripts. Its sub-directories contain script pages files in JSP and JavaScript formats. The directory also contains the forms and the action scripts for the storage and retrieval of the complete project, task and node information.
In addition to maintaining the files under the system the Webmaster has to maintain the Database Server and the Apache Server that forms the backbone of the system.

The Email server has all the documentation in the java source file, and also the usage of the functions. The code can be changed to accommodate more functionality. The server silicon.csci.csusb.edu should have mail forwarding at all times.

All packages and modules are to be updated in the lib directory of tomcat. They are in the format of *.jar, which can be made using winzip.exe.

4.3 Maintenance Issues

In order to maintain efficiency, it is necessary to put a person, who is familiar with WMS as Webmaster / system administrator, in charge of detecting and reporting maintenance issues frequently. At this point, the maintenance manual will be used to detect and resolve maintenance issues on time.

4.4 Troubleshooting

The Webmaster would follow the file and directory listings to guide him through the troubleshooting process. He would have to go through the pages and upon detecting
any faults; he would make the relevant changes to the source files.

The database that is maintained in the Mysql directory has the name WMSDB. It should be opened in Mysql maker to edit and change it. If the database gets changed, then the module database server also gets changed.

![Directory Structure](image)

Figure 8. Directory Structure

4.5 Contact Information

In case of any irresolvable malfunctions in the system the Webmaster may feel free to contact Bhrigu Celly at:

mailto:bhrigu_celly@lycos.com
4.6 Tools Listing

The tools listing of the various tools used for maintaining and updating the WMS system are:

- Homepage for JavaScript language and HTML.
- Internet explorer 5.5 or Netscape 4.7 for Web browser
- MySql Maker
- MySql database maker
- Dream weaver
- Apache
- Tomcat
- Microsoft Publisher
- JDK 1.3
- WinZip
- Nokia Toolkit
- Phone.com SDK
- Nokia WAP Gateway (Nokia Active Server 2.1)
- Macromedia Fireworks
- Forte Community Edition 3.0

4.7 Summary

This manual will enable future generations of users to use the system. This manual describes all the upgrade
and maintenance features of the system. It also helps them understanding the system so that if there is any upgrade to the system required, it can be done.
CHAPTER FIVE

USERS MANUAL

5.1 Introduction

Wireless messaging and project management system (WMS) helps in managing large-scale projects. This part of the documentation helps the new user to set up projects and to execute and monitor the project well. This manual will help the users to create and execute projects.

5.2 Web Interface

The users have to login using the login screen. The Login ID has to be the email address of the user. This is important to differentiate the users. It is also important if in future developments the user needs to be sent email notifications. In this case it is easy for the system administrator to implement that feature. If the user is a new user then he needs to create an account. He should type his password twice so that the system can verify it. If the user already has an existing account, he can login using the existing users interface.
Figure 9. Login Screen

The user registration is done so that there can be a confirmation email sent to the user. This confirmation email just tells the user that he is registered and can use the system. Registration is important as it allows the user to create projects. Only registered users can create projects or use a project.
After registration the user can use the login page to login into the system. The user will receive a confirmation email from the system so that the user knows he is registered.
The project supervisor sees the screen above. He will have the option to see the project overview and to delete the project. All users that are registered to this system have the option of creating projects. If a user creates the project he is the supervisor by default. Only a supervisor can add and remove users from the project. If the user is not a supervisor for a project, he will not have Project Administration, Project Overview and Delete tags.

Also on the project screen there is the name of the user and all the projects associated with him. If a person
does not have the links of project administration next to it then he is not the supervisor for that project.

Figure 12. Project Administration Screen

The above screen is only visible to the project supervisor. It gives the option to the project supervisor to add any registered user to the project. He can give him the rights of either a maker or a user. He can also delete and edit the rights of the added user. User Info gives information about the user he has added.

Project create option allows to create a new project and by default the user creating it is the project supervisor. Email Notify allows selected users to receive
email as project are updated and when projects get finished. This option can be activated from the edit menu.

![User Rights Edit Screen](image)

**Figure 13. User Rights Edit Screen**

Project Create option allows the user to select any registered user to the project as the maker. By default the project does not have its supervisor in the list. One registered user can have only one right.

Create project allows the user to create project and will allow the user to know if the project was created successfully or if there was a problem.
Figure 14. Project Create Screen

After creating the project, the project display screen shows the list of projects with the addition of a new project. The person creating the project will not have his name displayed, as he is already the supervisor of the project.
As we click on the project we get a new screen. This screen is called the task display screen. The screen is different for a user and for a maker.
Figure 16. Task Display Screen

If the user has maker or supervisor rights, then the task administration link is visible. The links create task and delete are only visible to the supervisors and makers.
Create task allows user to add more tasks to a project. This has the user name and project by default and they cannot be changed.
Figure 18. Task Administration Page

This allows the maker and supervisor to edit task descriptions and users assigned to the task. It also displays last update and last accessed users for information.
Figure 19. Node Display Page

This page allows the maker and supervisor to create nodes and delete nodes. If a user accesses the page, he does not get the delete and the create node options. He can only modify status and comments.
Node Description

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Structural Design calculations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>NTPC Kymore</td>
</tr>
<tr>
<td>Task Name</td>
<td>Tender</td>
</tr>
<tr>
<td>User Name</td>
<td><a href="mailto:briigu_cally@hotmail.com">briigu_cally@hotmail.com</a></td>
</tr>
<tr>
<td>Predecessor</td>
<td>Demolition of the tender</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td></td>
</tr>
</tbody>
</table>

Figure 20. Adding Node Page

A new node can be added on this page. This allows the maker and the supervisor to assign predecessors and successors to the newly created node. If it is the first node in the project then null will appear for successor and predecessor. This could be later edited to give the right predecessor and successor.
This screen allows the supervisor to see the project completion status, and also allows him to edit any of the project components.

5.3 Wireless Interface

The wireless interfaces have the same functionality for all the different kinds of users. They all are able to use the project, but they are not able to edit the project. They have the option to change the status of the project to yes or no.
Figure 22. Wireless Login Page

Use options on the left button to enter login and password. This screen may vary for different cellular phones.
This screen shows the projects the user has, and allows him or her to select one. The user can use the back button to go to the previous screen.
Figure 24. Task Selection Screen

This screen allows the user to select the task he wants to update. This leads to the node screen.
This screen gives the node list for the selected task. This will allow one to update status of the current node.
This screen allows one to change the status of the node. This will show an acknowledgement message if the node gets updated.

5.4 Summary

This section was the explanation of how the user should use the software, and what he should expect from the management system. It also showed the user the
different sections in the system and introduces him to how to create new projects. Also, it taught the user to use the wireless interface, and to update from a wireless device.
CHAPTER SIX
CONCLUSIONS

6.1 Introduction
The wireless messaging and management system is a system for project management. It is project independent, so it can be used for various different industries. This system uses JSP, Java, MySQL and WML as its essential components. The User Interface of the System was designed to accommodate a variety of projects. The projects get saved in the database, and the maker and the supervisor can access the project from any terminal having a web connection. The WAP browser can be any Cellular phone or hand held device.

6.2 Conclusions
Project Management is an important component in industry, and needs to be as real time as possible. This system uses the new wireless WAP technology to get real-time access to the project manager.

The management system provides a graphical user interface to display and manage project information. The supervisor can monitor how the project is proceeding and can make judgments on that in real time. The projects are
saved in a central database, so can be accessed from any Internet connecting terminal.

There are a few limitations on this system. First, the system can process only three levels of schedules. Secondly, there is no local console application. This requires the user to always logon to the system to create projects. This may reduce the efficiency of the system.

Future developments can be achieved in WAP Client browsers that can process java instead of only WML. Secondly, the making of a local console application could help users work when they are offline, and synchronize their work when they are connected to the Internet. Thirdly, Also this can be used to integrate with palm OS To Do list. Finally, it can be used for structured scheduling for use for MS students Project/Thesis.

6.3 Summary

Chapter Six reviewed the conclusions extracted from the project. Lastly, this project would be a great help to project managers all over the world. This system can be used with any industry, as it is not catering to any specific application.
APPENDIX A

DATABASE TABLE CREATION SCRIPT
CREATE TABLE Node_Attribute ( 
    Task_Name varchar(20),
    Node_Name varchar(20) NOT NULL,
    Project_Name varchar(20),
    Predecessor varchar(20),
    Successor varchar(20),
    Comments varchar(20),
    Status varchar(20)
)

Execution Successful

CREATE UNIQUE INDEX XPKNode_Attribute ON Node_Attribute ( Node_Name )

Execution Successful

CREATE TABLE Project_Database ( 
    Project_Name varchar(20) NOT NULL,
    Supervisor varchar(20),
    ... )
Project_Maker varchar(20),
Last_Update date NOT NULL,
Last_Update_User varchar(20),
Last_Accessed varchar(20)
)

Execution Successful

CREATE UNIQUE INDEX XPKProject_Database ON
Project_Database (
    Project_Name
)

Execution Successful

CREATE TABLE Task_Name (
    Task_Name varchar(20) NOT NULL,
    Project_Name varchar(20),
    User_Name varchar(20),
    Task_Description varchar(20)
)
Execution Successful

CREATE UNIQUE INDEX XPKTask_Name ON Task_Name
(
    Task_Name
)

Execution Successful

CREATE TABLE User ( 
    User_ID varchar(20) NOT NULL, 
    Password varchar(20), 
    Name_First varchar(20), 
    Name_Middle varchar(20), 
    Name_Last varchar(20), 
    Name_Title varchar(20), 
    Home_Address varchar(20), 
    Home_City varchar(20), 
    Home_State varchar(20), 
    Home_Zip varchar(20), 
    Home_County varchar(20), 
    Home_Country varchar(20), 
    Home_Phone varchar(20), 
)
CREATE UNIQUE INDEX XPKUser ON User
(
    User_ID
)

Execution Successful

CREATE TABLE User_Relation (
    Project_Name     varchar(20),
    User_ID          varchar(20),
    User_Type        varchar(20),
    Email_Notify     varchar(20)
)
APPENDIX B

PHONE MODELS USED IN TESTING
Figure 27. Nokia Mobile Browser 3.0
Figure 28. Nokia Mobile Toolkit Simulator
Figure 29. Alcatel Phone Simulator (Alav)
Figure 30. Ericsson Phone Simulator
Figure 31. Motorola Phone Simulator (Im1k)
Figure 32. Motorola Phone Simulator (Mtp1)
Figure 33. Openware Simulator Phone (Owg1)
Figure 34. Samsung Simulator Phone (Sh)
Figure 35. Mitsubishi Simulator Phone (Spol)
Figure 36. Mitsubishi Simulator (T250)
Class DBOBJECT

java.lang.Object

|--wms.modules.database_server.DBOBJECT

public abstract class DBOBJECT
extends java.lang.Object

This class creates all the database connections.

Field Detail

m_RowSet
protected java.sql.ResultSet m_RowSet

m_Stmt
protected java.sql.Statement m_Stmt

s_Conn
protected static java.sql.Connection s_Conn

Constructor Detail

DBOBJECT

public DBOBJECT()

The default constructor in the class
Method Detail

Refresh

public void Refresh()
Refreshes the database connection.

GetCount

public int GetCount()
Gets count of the recordset.

getSQLQuery

public java.lang.String getSQLQuery()
Gets the SQL Query

setSQLQuery

public void setSQLQuery(java.lang.String s)
Sets a SQL query in a recordset.

HandleException

public void HandleException(java.sql.SQLException sqlexception)
Exception when there is an error in the SQL command.

MoveNext

public boolean MoveNext()
Moves the pointer to the next record in the recordset.

IsEmpty
public boolean isEmpty()

Returns true is the recordset does not have records in it.

Open

public boolean open(java.lang.String s)

Opens a database connection to the database.

Execute

public boolean execute(java.lang.String s)

Executes a SQL Query string.

Init

protected abstract void init()

Initializes the recordset.

Move

public boolean move(int i)

Moves to record i in the present recordset.
wms.modules.database_server

Class Node

java.lang.Object

|-- wms.modules.database_server.DBObj

|-- wms.modules.database_server.Node

Constructor Detail

Node

public Node(java.lang.String TaskName,
            java.lang.String ProjectName,
            java.lang.String NodeName)

Constructor which takes in the argument of taskname, project name and node name to get the exact required node.

Node

public Node(java.lang.String TaskName,
            java.lang.String ProjectName)

Constructor, which takes in the argument of taskname, project name and node name to get the recordset of nodes with the project name and task name.
public Node()

Constructor, which takes in the argument of project name and node name to get the recordset of nodes with the project name.

Method Detail

setProjectName

public void setProjectName(java.lang.String s)
Sets the project name to the string s.

getProjectName

public java.lang.String getProjectName()
Gets the project name from the object node.

setTaskName

public void setTaskName(java.lang.String s)
Sets the task name to the string s.

getTaskName

public java.lang.String getTaskName()
Gets the task name from the object node.

getNodeName

public java.lang.String getNodeName()
Gets the node name from the object node.

setNodeName

public void setNodeName(java.lang.String s)
Sets the node name to the string s.

**getPredessor**

```java
public java.lang.String getPredessor()
```

Gets the predessor name from the object node.

**setPredessor**

```java
public void setPredessor(java.lang.String s)
```

Sets the predessor name to the string s.

**getSuccessor**

```java
public java.lang.String getSuccessor()
```

Gets the successor name from the object node.

**setSuccessor**

```java
public void setSuccessor(java.lang.String s)
```

Sets the successor name to the string s.

**getComments**

```java
public java.lang.String getComments()
```

Gets the comments from the object node.

**setComments**

```java
public void setComments(java.lang.String s)
```

Sets the comments to the string s.

**getStatus**

```java
public java.lang.String getStatus()
```

Gets the status from the object node.

**setStatus**
public void setStatus(java.lang.String s)
Sets the status to the string s.

clear
public void clear()
Clears the node, resets all the values to null.

GetAll
public void GetAll(java.lang.String s)
Gets all the nodes in the database.

GetNode
public Node GetNode()
Returns the current node object.

Update
public boolean Update()
Updates node attributes in the database

Submit
public boolean Submit()
Submits the data to the database.

GetSpecificByTaskName
public void GetSpecificByTaskName(java.lang.String t)
Gets all the nodes with the task name s.

GetSpecificByProjectName
public void GetSpecificByProjectName(java.lang.String s)
Gets all the nodes with the project name s.

**GetSpecificByNodeName**

```java
public void GetSpecificByNodeName(java.lang.String s)
```

Gets all the nodes with the node name s.

**DeleteAllProjectNameTaskName**

```java
public boolean DeleteAllProjectNameTaskName()
```

Deletes all with the current project and task name.

**DeleteAllProjectNameTaskName**

```java
public boolean
DeleteAllProjectNameTaskName(java.lang.String p,
java.lang.String t)
```

Deletes all with the project name p and task name t.

**DeleteAllProjectNameTaskNameNodeName**

```java
public boolean DeleteAllProjectNameTaskNameNodeName()
```

Deletes all with the current project name, task name and node name.

**DeleteAllProjectNameTaskNameNodeName**

```java
public boolean
DeleteAllProjectNameTaskNameNodeName(java.lang.String p,
java.lang.String t, java.lang.String n)
```

Deletes all with the project name p, task name t and node name n.

**DeleteAllProjectName**
public boolean DeleteAllProjectName()

Deletes all nodes with the current project name.

**DeleteAllProjectName**

public boolean DeleteAllProjectName(java.lang.String p)

Deletes all nodes with the project name p.

**GetAll**

public void GetAll()

Gets all the nodes in the node database.

**GetSpecificByProjectNameTaskName**

public void GetSpecificByProjectNameTaskName(java.lang.String s, java.lang.String t)

Gets all nodes with the project name s and task name t.

**GetSpecificByProjectNameTaskNameNodeName**

public void GetSpecificByProjectNameTaskNameNodeName(java.lang.String s, java.lang.String t, java.lang.String n)

Gets all nodes with the project name s and task name t and node name n.

**Insert**

public boolean Insert()

Inserts a node in the database.
**Init**

protected void Init() Overrides:

Init in class DBObject

Overrides the method in DBOject and initializes the node object.

**Save**

public boolean Save()

Saves the current node in the node database.
wms.modules.database_server

Class NodeList

java.lang.Object

|--wms.modules.database_server.NodeList

Constructor Detail

NodeList

public NodeList(java.lang.String ProjectName,
                  java.lang.String TaskName)

Creates a node list for ProjectName and TaskName.

NodeList

public NodeList()

Default constructor.

Method Detail

OpenList

public java.util.Vector OpenList()

Opens a nodelist and retrieves data from the node database.

getNodeList

public java.util.Vector getNodeList()
Gets a node list for the current task in a vector.

**appendNode**

```java
public void appendNode(Node n)
```

Appends node n.

**removeNode**

```java
public void removeNode(Node node)
```

Removes node object called node from the temporary vector.

**removeNode**

```java
public void removeNode(int i)
```

Removes node from the temporary node vector.

**insertNode**

```java
public void insertNode(Node n, int i)
```

Inserts node n in position i in the temporary node vector.

**sortList**

```java
public void sortList()
```

Sorts the temporary vector.

**saveList**

```java
public void saveList()
```

Saves the temporary vector in the node database.

**clearList**

```java
public void clearList()
```
Clears the temporary node list vector.

**deleteList**

```java
class NodeList {
    public void deleteList()
        Deletes the temporary node list vector.
}
```

**setTaskName**

```java
class NodeList {
    public void setTaskName(java.lang.String s)
        Sets the task name of the temporary node list to s.
}
```

**getTaskName**

```java
class NodeList {
    public java.lang.String getTaskName()
        Gets the task name of the node list.
}
```

**updateNode**

```java
class NodeList {
    public void updateNode(int i, java.lang.String ProjectName,
            java.lang.String TaskName,
            java.lang.String NodeName,
            java.lang.String Predessor,
            java.lang.String Successor,
            java.lang.String Status,
            java.lang.String Comment)
        Updates all attributes of a particular node to the node database.
}
```

**getNode**

```java
class NodeList {
    public Node getNode()
        Returns the node object.
}
```
**setNode**

```java
public void setNode(java.lang.String TaskName, 
java.lang.String ProjectName, 
java.lang.String NodeName, 
java.lang.String Predessor, 
java.lang.String Successor, 
java.lang.String Status, 
java.lang.String String Comment)
Sets the value of the present node to the input terms.
```
wms.modules.database_server
Class Project
java.lang.Object

|--wms.modules.database_server.DBObject
   |  
   |--wms.modules.database_server.Project

Constructor Detail

Project
public Project()
Default constructor.

Project
public Project(java.lang.String ProjectName)
Constructor that takes in the project name.

Method Detail

setProjectName
public void setProjectName(java.lang.String s)
Sets project name to string p.

getProjectName
public java.lang.String getProjectName()
Gets project name from the project object.

setProjectSupervisor

public void setProjectSupervisor(java.lang.String s)

Sets project supervisor to string s.

getProjectSupervisor

public java.lang.String getProjectSupervisor()

Gets project supervisor from the project object.

setProjectMaker

public void setProjectMaker(java.lang.String s)

Sets project maker to string s.

getProjectMaker

public java.lang.String getProjectMaker()

Gets project maker from the project object.

setLastUpdate

public void setLastUpdate(java.lang.String s)

Sets last update to string s.

getLastUpdate

public java.lang.String getLastUpdate()

Gets last update the project object.

setLastUpdateUser

public void setLastUpdateUser(java.lang.String s)

Sets last update user to string s.

getLastUpdateUser
public java.lang.String getLastUpdateUser()

Gets last update user from the project object.

setLastAccessed

public void setLastAccessed(java.lang.String s)

Sets last accessed to string s.

g getLastAccessed

public java.lang.String getLastAccessed()

Gets last accessed from the project object.

addTask

public void addTask(java.lang.String s)

Adds Task Name s to the current project object.

SaveTask

public void SaveTask()

Saves the current task to the current project object.

GetTask

public Task GetTask(java.lang.String s)

Gets the current task from the project object.

clear

public void clear()

Clears the task list in the project.

GetAll

public void GetAll(java.lang.String s)
Gets all the projects in the project database.

**GetProject**

```java
public Project GetProject()
```

Gets the current project object.

**Update**

```java
public boolean Update()
```

Updates the current project object and updates it to the project database.

**Submit**

```java
public boolean Submit()
```

Calls submit function in the form.

**GetSpecificByProjectName**

```java
public void GetSpecificByProjectName(java.lang.String s)
```

 Gets a specific project from the project database.

**DeleteAllProjectName**

```java
public boolean DeleteAllProjectName()
```

Deletes all project with the current project name from the project database.

**DeleteAllProjectName**

```java
public boolean DeleteAllProjectName(java.lang.String p)
```
Deletes all project with the project name p from the project database.

**GetAll**

```java
public void GetAll()
```

Gets all projects from the project database.

**Insert**

```java
public boolean Insert()
```

Inserts the current project if there does not exist a project with the same name in the project database.

**Init**

```java
protected void Init() Overrides:
Init in class DBObject
```

Overrides the init function in the database object and initialize the project object.

**Save**

```java
public boolean Save()
```

Saves the current project to the project database.
wms.modules.database_server

Class Task

java.lang.Object

|--wms.modules.database_server.DBObj ect

|--wms.modules.database_server.Task

Constructor Detail

Task

public Task()
Default constructor.

Task

public Task(java.lang.String ProjectName,
java.lang.String TaskName)
Creates a Task object with ProjectName and TaskName.

Method Detail

setProjectName

public void setProjectName(java.lang.String s)
Sets project name in the task object to s.
getProjectName

public java.lang.String getProjectName()

Gets project name form the task object.

setTaskName

public void setTaskName(java.lang.String s)

Sets task name to s in the task object.

getTaskName

public java.lang.String getTaskName()

Gets task name form the task object.

getUserName

public java.lang.String getUserName()

Gets user name form the task object.

setUserName

public void setUserName(java.lang.String s)

Sets user name to s in the task object.

getTaskDescription

public java.lang.String getTaskDescription()

Gets task description form the task object.

setTaskDescription

public void setTaskDescription(java.lang.String s)

Sets task description to s in the task object.

clear

public void clear()
Clears the task object attributes.

GetAll

public void GetAll(java.lang.String s)

Gets all the tasks from the task database.

GetTask

public Task GetTask()

Returns with the task object.

addTask

public void addTask()

Adds task to the current project.

Update

public boolean Update()

Updates the current task in the task database

Submit

public boolean Submit()

Calls the submit button on the form.

GetSpecificByTaskName

public void GetSpecificByTaskName(java.lang.String t)

Gets task name by the name t from the task database.

GetSpecificByProjectName

public void GetSpecificByProjectName(java.lang.String s)
Gets all the tasks in the project s from the task database.

**DeleteAllProjectNameTaskName**

```java
public boolean DeleteAllProjectNameTaskName()
```

Deletes all the tasks with the current project name and task name.

**DeleteAllProjectNameTaskName**

```java
public boolean DeleteAllProjectNameTaskName(java.lang.String p, java.lang.String t)
```

Deletes all the tasks with the task name t and project name p.

**DeleteAllProjectName**

```java
public boolean DeleteAllProjectName()
```

Deletes all tasks with the current project name.

**DeleteAllProjectName**

```java
public boolean DeleteAllProjectName(java.lang.String p)
```

Deletes all tasks with the project name p.

**GetAll**

```java
public void GetAll()
```

Gets all tasks from the task database.

**GetSpecificByProjectNameTaskName**
public void
GetSpecificByProjectNameTaskName(java.lang.String s, java.lang.String t)
Gets all tasks with the task name t and project name s

Insert

public boolean Insert()
Inserts the task if it does not already exist.

Init

protected void Init()Overrides:
Init in class DBObject
Initializes the task object by getting the task value from the task database.

Save

public boolean Save()
Saves the task values to the task database.
wms.modules.database_server

Class User

java.lang.Object
   |
   +--wms.modules.database_server.DBObj
      |
      +--wms.modules.database_server.User

Constructor Detail

User

public User()
Default constructor.

Method Detail

setFirstName

public void setFirstName(java.lang.String s)
Sets first name in the user object.

Update

public boolean Update()
Updates the user fields in the user table.

Submit
public boolean Submit()
Calls the submit button on the form.

getHomeState

public java.lang.String getHomeState()
Gets home state from the user object.

setHomeState

public void setHomeState(java.lang.String s)
Sets home state to s in the user object.

getOfficeCountry

public java.lang.String getOfficeCountry()
Gets office country from the user object.

setOfficeCountry

public void setOfficeCountry(java.lang.String s)
Sets office country to s in the user object.

GetSpecificByLastName

public void GetSpecificByLastName(java.lang.String s)
Gets a user by the last name s.

GetAll

public void GetAll()
Gets all users.

getOfficeZip

public java.lang.String getOfficeZip()
Gets the office zip from the user object.
setOfficeZip
public void setOfficeZip(java.lang.String s)
Sets office zip to s in the user object.

getHomeCity
public java.lang.String getHomeCity()
Gets home city from the user object.

setHomeCity
public void setHomeCity(java.lang.String s)
Sets home city to s in the user object.

getHomePhone
public java.lang.String getHomePhone()
Gets home phone from the user object.

setHomePhone
public void setHomePhone(java.lang.String s)
Sets home phone to s in the user object.

getHomeAddress
public java.lang.String getHomeAddress()
Gets home address from the user object.

setHomeAddress
public void setHomeAddress(java.lang.String s)
Sets home address to s in the user object.

GetSpecific
public void GetSpecific(java.lang.String s)
Gets specific user with the LoginID s from the user database.

**getOfficeState**

public java.lang.String getOfficeState()

Gets office state from the user object.

**setOfficeState**

public void setOfficeState(java.lang.String s)

Sets office state to s in the user object.

**Insert**

public boolean Insert()

Inserts the current user in the user database.

**GetSpecificByFirstName**

public void GetSpecificByFirstName(java.lang.String s)

Gets a specific user from the user database by the last name s.

**getLoginID**

public java.lang.String getLoginID()

Gets LoginID of the user object.

**setLoginID**

public void setLoginID(java.lang.String s)

Sets LoginID to s for the current user.

**Init**

108
protected void Init() Overrides:
Init in class DBOBJECT
Initializes the user object by taking values from the database.

getHomeCounty
public java.lang.String getHomeCounty()
Gets home country from the user object.

setHomeCounty
public void setHomeCounty(java.lang.String s)
Sets home country to s in the users object.

getOfficeCity
public java.lang.String getOfficeCity()
Gets office city from the user object.

setOfficeCity
public void setOfficeCity(java.lang.String s)
Sets office city to s in the object user.

getOfficePhone
public java.lang.String getOfficePhone()
Gets office phone from the user object.

setOfficePhone
public void setOfficePhone(java.lang.String s)
Sets office phone to s in the object user.

getOfficeAddress
public java.lang.String getOfficeAddress()
Gets office address from the user object.

setOfficeAddress

public void setOfficeAddress(java.lang.String s)
Sets office phone to s in the object user.

getTitle

public java.lang.String getTitle()
Gets title from the user object.

setTitle

public void setTitle(java.lang.String s)
Sets title to s in the object user.

getMiddleName

public java.lang.String getMiddleName()
Gets middle name from the user object.

setMiddleName

public void setMiddleName(java.lang.String s)
Sets middle name to s in the object user.

getLastName

public java.lang.String getLastName()
Gets last name from the user object.

setLastName

public void setLastName(java.lang.String s)
Sets last name to s in the object user.
getHomeCountry
public java.lang.String getHomeCountry()
Gets home country from the user object.

setHomeCountry
public void setHomeCountry(java.lang.String s)
Sets home country to s in the object user.

Save
public boolean Save()
Save object user to the user database.

getHomeZip
public java.lang.String getHomeZip()
Gets home zip code from the user object.

setHomeZip
public void setHomeZip(java.lang.String s)
Sets home zip to s in the object user.

Login
public static boolean Login(java.lang.String s,
        java.lang.String s1)
Gets loginID s and password to check in the user
database and allow user to login.

getPassword
public java.lang.String getPassword()
Gets password from the user database.
setPassword

public void setPassword(java.lang.String s)
Sets password to s in the users database.

getFirstName

public java.lang.String getFirstName()
Gets first name from the object user.
wms.modules.database_server

Class UserRelation

java.lang.Object

|--wms.modules.database_serverDBObject

|--wms.modules.database_server.UserRelation

Constructor Detail

UserRelation

public UserRelation()
Default constructor

Method Detail

setProjectName

public void setProjectName(java.lang.String s)
Sets project name to s in the object user relation.

getProjectName

public java.lang.String getProjectName()
Gets project name from the object user relation.

getUserID
public java.lang.String getUserID()

Gets userID from the object user relation.

setUserID

public void setUserID(java.lang.String s)

Sets userID to s in the object user relation.

getUserType

public java.lang.String getUserType()

Gets user type from the object user relation.

setUserType

public void setUserType(java.lang.String s)

Sets user type to s in the object user relation.

GetAll

public void GetAll(java.lang.String s)

Gets all the user from the user database.

GetUser

public User GetUser()

Returns the current user object.

GetUserRelation

public UserRelation GetUserRelation()

Gets the current user relation back.

Update

public boolean Update()
Updates the user relation in the user relation database.

Submit

public boolean Submit()

Is called when the user submits a form.

GetSpecificByProjectName

public void GetSpecificByProjectName(java.lang.String s)

Gets all the relations with project name s from the user relation database.

GetSpecificByProjectNameUserName

public void GetSpecificByProjectNameUserName(java.lang.String s, java.lang.String u)

Gets all the relations with project name s and user id u from the user relation database.

DeleteByProjectNameUserName

public void DeleteByProjectNameUserName(java.lang.String s, java.lang.String u)

Deletes all records with the project name p and the user name u from the user database.

GetAll

public void GetAll()
Gets all user relations from the user relation database.

**GetSpecific**

```java
public void GetSpecific(java.lang.String s)
```

Gets the user relations for the project name s from the user relation database.

**Insert**

```java
public boolean Insert()
```

Inserts a user relation in the user relation database.

**DeleteAllProjectName**

```java
public boolean DeleteAllProjectName(java.lang.String P)
```

Deletes all records with project name s from the user relation database.

**Init**

```java
protected void Init() Overrides:
Init in class DBObject
```

Initializes the object user relation.

**Save**

```java
public boolean Save()
```

Saves the object user relation to the user relation database.
wms.modules.email_server

Class EmailEngine

java.lang.Object

|--wms.modules.email_server>EmailEngine

Constructor Detail

EmailEngine

public EmailEngine(java.lang.String s,
    java.lang.String s1,
    java.lang.String s2,
    java.lang.String s3)

Constructor requiring different parameters for email transmission.

EmailEngine

public EmailEngine(java.lang.String s,
    java.lang.String s1,
    java.lang.String s2,
    java.lang.String s3,
    java.lang.String s4,
    java.lang.String s5)
Constructor requiring different parameters for email transmission.

EmailEngine

custom EmailEngine(java.util.Properties properties)
Default Constructor.

Method Detail

getSubject

custom java.lang.String getSubject()
Gets subject from the email engine object.

isDebug

custom boolean isDebug()
Checks for errors in the email.

getCc

custom java.lang.String getCc()
Gets cc from the email engine object.

setCc

custom void setCc(java.lang.String s)
Sets cc to s in the email engine object.

getUser

custom java.lang.String getUser()
Gets user from the email engine object.
**setUser**

public void setUser(java.lang.String s)

Sets user to s in the email engine object.

**getTo**

public java.lang.String getTo()

Gets to address from the email engine object.

**setTo**

public void setTo(java.lang.String s)

Sets to address to s in the email engine object.

**debug**

protected void debug(java.lang.String s)

Checking for errors sub function.

**getProtocol**

public java.lang.String getProtocol()

Gets protocol from the email engine object.

**setProtocol**

public void setProtocol(java.lang.String s)

Sets the protocol to s in the email engine object.

**setDebug**

public void setDebug(boolean flag)

Sets the debug option in the email engine object.

**getUrl**

public java.lang.String getUrl()
setUrl

public void setUrl(java.lang.String s)
Sets the URL in the email engine object.

getBcc

public java.lang.String getBcc()
Gets Bcc from the email engine object.

setBcc

public void setBcc(java.lang.String s)
Sets Bcc to s in the email engine object.

getMailhost

public java.lang.String getMailhost()
Gets mail host from the email engine object.

setMailhost

public void setMailhost(java.lang.String s)
Sets mail host to s in the email engine object.

getBody

public java.lang.String getBody()
Gets body of the email from the email engine object.

setBody

public void setBody(java.lang.String s)
Sets email body to s in the email engine object.

getPassword

120
public java.lang.String getPassword()

Gets password from the user object.

setPassword

public void setPassword(java.lang.String s)

Sets password to s for the user object.

getFilename

public java.util.Vector getFilename()

Gets filename from the email engine object.

setFilename

public void setFilename(java.lang.String s)

Sets filename to s in the email engine object.

getFrom

public java.lang.String getFrom()

Gets the senders email address from the email engine object.

setFrom

public void setFrom(java.lang.String s)

Sets the senders email address to s in the email engine object.

getHost

public java.lang.String getHost()

Gets the host name from the email engine object.

setHost
public void setHost(java.lang.String s)
Sets the host name to s in the email engine object.

getRecord

public boolean getRecord()
Gets record.

setRecord

public void setRecord(boolean flag)
Sets record.

send

public int send()
    throws java.lang.Exception
Sends email to the recipients.

setSubject

public void setSubject(java.lang.String s)
Sets the subject of the email to s.
REFERENCES


