**TiGERS**



Tiffany Goldmine Environmental Management Report System

**CSSE7024 System Manual**

Wei Yeap Cheng (Nicholas),  
Jane Harrison,   
Xing-Shu Liu,   
Justin Mancinelli,   
Chih-Hsiang Tang (Sean) and Dell Topel



Tiffany Gold Mine Environmental Management Reporting System – System Manual

[1 System Requirements (minimum) 3](#_Toc274082052)

[1.1 Hardware Interfaces 3](#_Toc274082053)

[1.1.1 Users 3](#_Toc274082054)

[1.1.2 System Administrators 3](#_Toc274082055)

[1.2 Software Interfaces 3](#_Toc274082056)

[1.2.1 Users 3](#_Toc274082057)

[1.2.2 System Administrators 3](#_Toc274082058)

[2 Application Installation 4](#_Toc274082059)

[3 Application Configuration 5](#_Toc274082060)

[3.1 Database Configuration 5](#_Toc274082061)

[3.1.1 File: pom.xml 5](#_Toc274082062)

[3.2 Email Configuration 6](#_Toc274082063)

[3.2.1 File: applicationContext-service.xml 6](#_Toc274082064)

[3.2.2 File: mail.properties 6](#_Toc274082065)

[4 Application Set Up and Maintenance 7](#_Toc274082066)

[4.1 Set Up 7](#_Toc274082067)

[4.2 Maintenance 7](#_Toc274082068)

[4.2.1 AppFuse 7](#_Toc274082069)

[4.2.2 Spring 8](#_Toc274082070)

[4.2.3 Hibernate 8](#_Toc274082071)

[4.2.4 Integration 8](#_Toc274082072)

# System Requirements (minimum)

## Hardware Interfaces

### Users

* 512 MB RAM
* 900 MHz CPU
* 256 colour monitor @ 800x600 resolution
* Network interface
* 20 MB free space on hard drive

### System Administrators

* 1 GB RAM
* 1 GHz CPU
* 256 colour monitor @ 800x600 resolution
* Network interface
* 100GB free space on hard drive

## Software Interfaces

### Users

* Windows XP
* Internet Explorer 8

### System Administrators

* Windows XP (for application)
* Solaris (for database)
* Oracle Database 10g
* Java JDK 1.6
* Apache Maven 2.2.1
* Eclipse IDE 3.3
* Internet Explorer 8
* Subversion 1.6.11 or TortoiseSVN 1.6.11
* SMTP server

# Application Installation

The following assumes that all hardware and software requirements above are available and installed. See relevant documentation for installation instructions of the required software.

1. Check out a working copy from <https://tigers.googlecode.com/svn/trunk/> using the subversion or TortoiseSVN client.
2. Using the command-line, navigate to the working copy’s “tigers” subfolder
3. Generate an Eclipse project by entering “mvn eclipse:eclipse”
4. Open Eclipse and add an existing project located in the “tigers” subfolder of the working copy.
5. Ensure the M2\_REPO classpath variable is defined
   1. From the menu bar, select “Window🡪Preferences”
   2. Select the “Java🡪Build Path🡪Classpath Variables” page
   3. Add the name “M2\_REPO” with the path “[your/maven/repository]” as configured in your maven configuration.
   4. Apply all changes
6. Configure the application as in the section below.
7. Using the command-line, navigate to the working copy’s “tigers” subfolder
8. Start the application for production by entering “mvn jetty:run-war -Pprod”  
   (for development, enter “mvn jetty:run-war”)

# Application Configuration

The following outlines the necessary procedure to configure the system as a whole to integrate with other services. For general maintenance of the application, see the section below.

## Database Configuration

### File: pom.xml

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.[[1]](#footnote-1)

Towards the bottom the the file, find the section labelled “Database settings”. Here you are able to configure database connectivity. Since it is assumed that the application will run on Oracle Database 10g, there are only four fields which must be modified.

* jdbc.version – the version of the installed Oracle Database 10g
  + You may find this information by performing the following query from SQL\*Plus (installed with Oracle Database 10g):   
    “select \* from v$version;”
* jdbc.url – the URL string used by the application to connect to Oracle Database 10g
  + Ask your database administrator for this information as it is configured during installation of Oracle Database 10g which is out of the scope of this document
* jdbc.username – the username used by the application to connect to Oracle Database 10g
  + Ask your database administrator for this information as it is configured during installation of Oracle Database 10g which is out of the scope of this document
* jdbc.password – the password used by the application to connect to Oracle Database 10g
  + Ask your database administrator for this information as it is configured during installation of Oracle Database 10g which is out of the scope of this document

Other aspects of this file are beyond the scope of this document.

### Files: sample-data.xml, default-data.xml

These xml files store the state of the database to be loaded when starting the application. The file sample-data.xml is to be used for testing during development whereas default-data.xml is to be used as the production data. The Maven plugin DBunit is able to read these files to populate the database.

The files are in a strict XML format but it is beyond the scope of this document to detail this format.

Since restarting the server will look at one of these files (depending on whether restarting in production or development mode), it is important, especially in production, to back-up the current state of the database before restarting. Execute the command “mvn dbunit:export –Ddest=production.xml” to export the database to an xml file.

SQL scripts containing development data have been included with the source code. They provide an explicit representation of the database as developed. You may execute these under the Oracle environment to populate data or modify them as need be. Details of SQL and Oracle are beyond the scope of this document.

## Email Configuration

### File: applicationContext-service.xml

This file contains configuration of various software services. Since the application must send emails to users, it is important to configure the SMTP server properties here. Find the section labelled “Mail: Sender and Velocity configuration” then ask your SMTP server administration for further guidance as this information is out of the scope of this document.

### File: mail.properties

This file contains properties used to configure mail settings in applicationContext-service.xml. Here, you may configure the default sender, the protocol to use, mail hosts and default templates.

It is assumed that you will use an SMTP server so the property “mail.transport.protocol” must be set as “smtp”

Other aspects of this file are beyond the scope of this document.

# Application Set Up and Maintenance

## Set Up

It is possible to set up the application given the information in the sections above presuming the minimum requirements are satisfied. This is possible because all necessary application code is contained in the Subversion repository and because Apache Maven will automatically find, download, and update all necessary dependencies.

## Maintenance

In order to maintain the system, it is necessary to understand the components which make up the application. AppFuse[[2]](#footnote-2) is the primary framework. The version of AppFuse used comes integrated with Spring[[3]](#footnote-3), Hibernate[[4]](#footnote-4), and various other packages which are beyond the scope of this document.

The following subsections contain general information about the design and purpose of the three main components and their integration into the application as a whole.

### AppFuse

AppFuse organizes files according to purpose and function as dictated by the following package conventions:

* Java Directory
  + model – Entities
  + dao – Dao Interfaces
  + dao.hibernate – Hibernate Implementations
  + service – Service Interfaces
  + service.impl – Service implementation
  + webapp.controller – Controllers
* Resources Directory
  + Application configuration files
  + Application property files
* Webapp Directory
  + View configuration files
  + Dynamic Views
  + Static file resources (e.g. images, styles, scripts, etc)

### Spring

The Spring framework contains several modules for various services[[5]](#footnote-5) but the modules which are most important for maintaining the application are Spring MVC and Spring Security.

Spring MVC implements the Model-View-Controller design architecture. The models are contained in the model package, the controllers are contained in the webapp.controller package, and the views are contained in the Webapp directory.

Spring MVC follows a default naming convention for the three types of files (the default can be overridden but that is beyond the scope of this document). See the Integration subsection below for an example of this convention.

Spring Security allows role-based authentication on URL patterns and also provides a tag library for use in JSP pages for further control. The configuration of URL patterns is specified in security.xml which can be found in the Webapp directory. The tag library can be imported just as any other standard JSP tag library for use in JSP pages. How to use these features is beyond the scope of this document.

### Hibernate

Hibernate is an object-relational mapping (ORM) library. Its primary uses are to provide a standard API for querying databases and encapsulating results as standard Java objects.

This application leverages the Hibernate query language (HQL) extensively for querying the Oracle database. Hibernate provides several means of querying the database but such a discussion is beyond the scope of this document.

### Integration

The following is an example of integrating Spring MVC, Hibernate, and the AppFuse package conventions and configuration.

To complete the MVC stack for viewing and modifying samples, create a model named Sample.java, two controllers named SampleController.java and SampleFormController.java, and two views named samples.jsp and sampleForm.jsp then configure Spring to recognize these files by adding entries in dispatcher-servlet.xml

Several files must be added and others must be modified to integrate the sampler MVC stack with the Hibernate layer. First, create classes which define data access objects (DAO) for the sampler; SamplerDao.java provides an interface to define the access methods available to samplers while SamplerDaoHibernate.java implements those methods using the Hibernate library (e.g. HQL). Second, modify the configuration file applicationContext-dao.xml in the Resources directory to specify the relationship between this DAO and the Model. Third, make Hibernate aware of this model by modifying the configuration file hibernate.cfg.xml in the Resources directory. The previous step gives Hibernate the ability to auto-generate the SQL code necessary for adding and modifying the appropriate table and rows of the database.

Finally, AppFuse must be made aware that samplers should be available as a web service for viewing. Add the file samplerManager.java to the service package and samplerManagerImpl.java to the service.impl package. The methods of the web service are declared in samplerManager.java and implemented in samplerManagerImpl.java. It is now possible to give samplerController.java access to the manager so that the views will be accessible via the web interface. The manager is then given access to the DAO so that the views can be dynamically generated.

Actual implementations are beyond the scope of this document but the source code is available via the Subversion repository. The above example and subsections provide the necessary information to understand the code and design of the application for future maintenance.

1. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html [↑](#footnote-ref-1)
2. http://AppFuse.org [↑](#footnote-ref-2)
3. http://www.springsource.com [↑](#footnote-ref-3)
4. http://www.hibernate.org/ [↑](#footnote-ref-4)
5. http://en.wikipedia.org/wiki/Spring\_Framework [↑](#footnote-ref-5)