Eclipse GWT Setup Notes

Setting up Eclipse with Google Web Toolkit, Hibernate, with SmartGWT/ExtGWT

Robert N. Lacatena

Table of Contents

Introduction

Selected Tools and Toolkits

This document will step through the creation of an environment and project that will make use of :

- Eclipse IDE
- Google Web Toolkit (GWT) for integrated java/javascript/web 2.0 development
- MySQL database functionality
- Hibernate Tools for simplified database access,
- SmartGWT (SWT) toolkit for expanded web 2.0 development (i.e. superior, more functional widgets). *Optional.*
- ExtGWT toolkit for expanded web 2.0 development (i.e. superior, more functional widgets). *Optional.*
- Firebug, for Firefox, is a tremendous HTML/CSS/JavaScript research and debugging too. It, combined with Firefox, is *very highly recommended* for development efforts.

Almost all steps outlined in this document need only be performed once, for environment and project creation.

Eclipse has been selected as a popular free, open source integrated development environment.

MySQL has been selected as a popular, free, open source, and functional database which enjoys a wide install base and range of support.

Google Web Tookit 2.0 (GWT) has been selected because it automatically handles all browser compatibility problems concerning JavaScript functionality, layout issues, and AJAX behavior. In addition, it allows the developer to work almost exclusively in Java and ignore Javascript and, in some cases, HTML and CSS, thus freeing the developer to focus on a single language and tool. GWT, with the support of Google, can be expected to be maintained and to enjoy an adequate implementation and support base for some time.

Hibernate and the accompanying **Hibernate Tools** has been selected because it automatically takes care of synchronizing code with a database, while also simplifying and encapsulating database access, freeing the programmer from the tedious and sometimes error prone task of writing DAO/POJO java classes. In this project, HQL and other Hibernate features will likely not be used; the mere benefit of DAO/POJO creation and update/synchronization will be leveraged.

Eclipse, GWT, Hibernate, SWT Set Up Notes

SmartGWT (SWT) is an optional library that provides a richer toolkit and interface in terms of available widgets, effects and presentation, and it is also offered with an LGPL open source license, permitting its use in commercial projects without obligation. It should be noted that SWT is not always entirely compatible with GWT (i.e. layout issues can be encountered), so whenever possible, if not always, when any of SWT is used, then SWT solutions should be chosen over pure GWT solutions.

Ext GWT is an alternative to SmartGWT which provides a richer toolkit and interface in terms of available widgets, effects and presentation, and it is also offered with an GPL open source license, permitting its use in commercial projects with the obligation to make source code available along with any downloadable binaries, which is not an issue in this case. Ext GWT is more compatible with GWT, making it in some ways a better option than SmartGWT.

Firebug, combined with **Firefox** as a browser, is a tremendous research and debugging tool. It clearly identifies the locations of HTML elements such as divs and table cells, highlighting them both visually and displaying the corresponding HTML source in a split pane, along will all CSS rules from various stylesheets which went into styling the element. It is invaluable in researching layout and visual organization issues.

Jetty will be used as the development web application server/container, but **Tomcat** is available as an option. GWT for Eclipse makes automatic use of Jetty as an web application server. While it is possible to change the environment to use Tomcat, for simplicity in setup and development, Jetty will be retained. A separate section will outline how to deploy a Jetty-developed project to Tomcat as a production application server.

Note: Because support for, licensing and functionality behind SWT can change over time, there are competing products, and in particular GWT itself may improve in functionality and so obviate the need for SWT, all interface logic should be isolated and programmed in a modular way whenever possible, to simplify any future need or desire to transition from SWT to another widget toolkit/library.

Note: SWT also offers server side functionality which cannot be used under the LGPL license, and may be of limited value to this project in any event (i.e. it involves a higher learning curve for relatively simple, straightforward server side tasks).

Installation Steps

The basic steps to be followed will be:

- Installation of the Eclipse IDE
- Installation of required plugins for GWT, MySQL and Hibernate development

Eclipse, GWT, Hibernate, SWT Set Up Notes

- Creation of a GWT Web Application project
- Addition of Hibernate access to a GWT project
- Addition of SWT capability to a GWT project

Lastly, a final section will outline the steps required to:

- Package and deploy the resulting WAR file to a Tomcat server.

Note: Any jars referenced in this document can and should be replaced by newer versions when available, but it should be recognized that in some cases version changes can introduce compatibility issues, and should be the one of the first things considered if unexpected problems are encountered.

This document makes use of a simple MySQL database, named "playground," with a single table named "user", defined below. This table exists only test database access, and may be replaced with anything:

Table: User

Column	Type	Size	Key?
user_name	varchar	32	key
password	varchar	32	
first_name	varchar	32	
last_name	varchar	32	
login_count	int	11	
session_id	int	11	indexed
session_start_time	datetime		
session_expire_time	datetime		

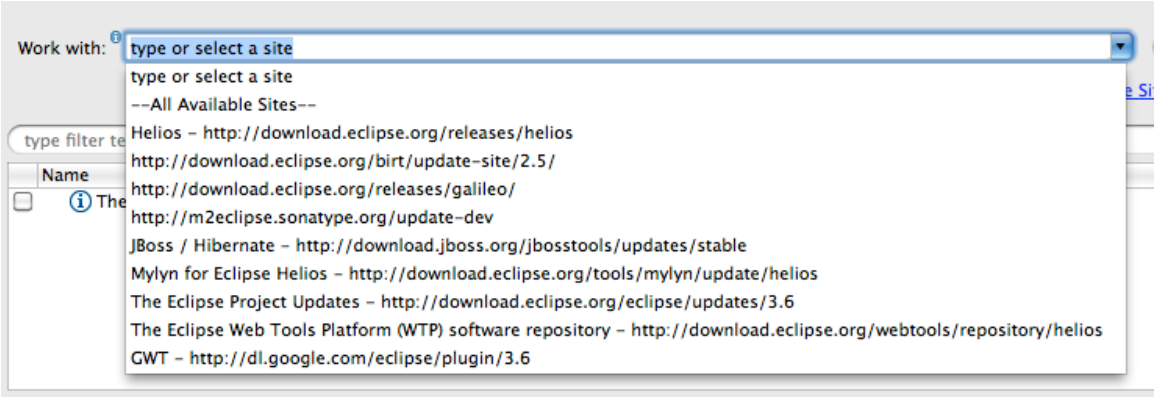
Eclipse GWT Setup

This section will install Eclipse and the minimal required plugins for GWT and Hibernate access.

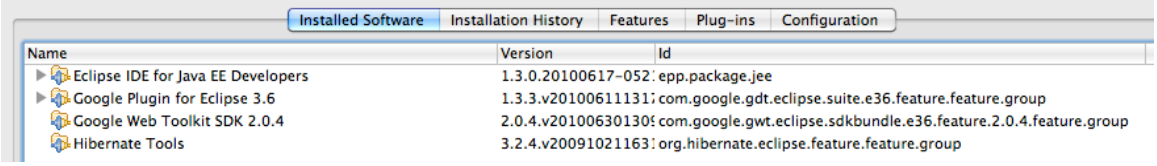
Eclipse installs:

This will install the Eclipse IDE, and the basic features needed for GWT, Hibernate, and SmartGWT development.

1. Eclipse IDE for Java EE Developers
   → http://download.eclipse.org/releases/helios
2. Google Plugin for Eclipse 3.6 + Google Web Toolkit SDK 2.0.4
   using (within Eclipse) *Help→Install New Software…* Add site, access, select and use:
   → http:/dl.google.com/eclipse/plugin/3.6
3. Hibernate
   using *Help→Install New Software…* Add site, access, select and use:
   → http://download.jboss.org/bosstools/updates/stable
4. SmartGWT (optional)
   Download and unzip to any work directory
   → http://code.google
   .com/p/smartgwt/downloads/detail?name=smartgwt-2.2.zip&can=2&q=
5. ExtGWT (optional)
   Download and unzip to any work directory
   →
   http://www.sencha.com/products/gwt/download.php?dl=extgwt220gwt2



Eclipse, GWT, Hibernate, SWT Set Up Notes



Basic GWT Test

This will test that GWT is set up and operating.

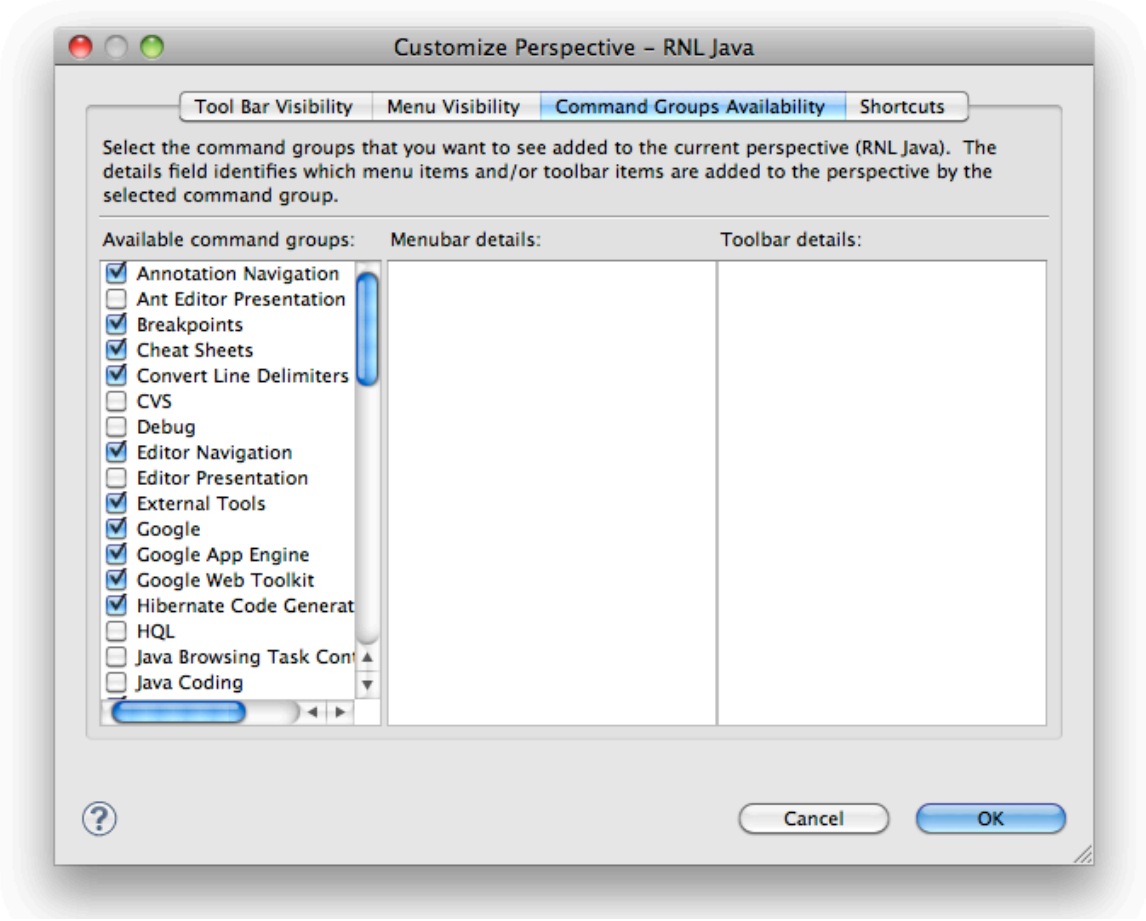
1. Create a new GWT project: Select *File* → *New…* → *Web Application Project*
2. Enter a new project name (e.g. GWT Playground)
3. Uncheck the *Google App Engine* checkbox
4. Click *Finish*
5. Run the project by selecting *Run As…* → *Web Application*
6. Copy the URL from the *Development Mode* window and paste it into a browser window to open the application's web page
7. Click the *Send* button and confirm a server response
8. Terminate the server in the *Development Mode* window

Hibernate Preparation

This will add the *Hibernate Code Generation* dropdown icon  to the toolbar.

1. Add to toolbar
   a. Choose Window → Customize Perspective…
   b. Select the *Command Groups Availability* tab

c. Check the *Hibernate Code Generation* checkbox



New GWT + Hibernate Project Setup

This section will create a project to employ GWT with Hibernate database access, and validate functionality through basic, simple tests. Note that the vast bulk of this set up is for Hibernate, not GWT.

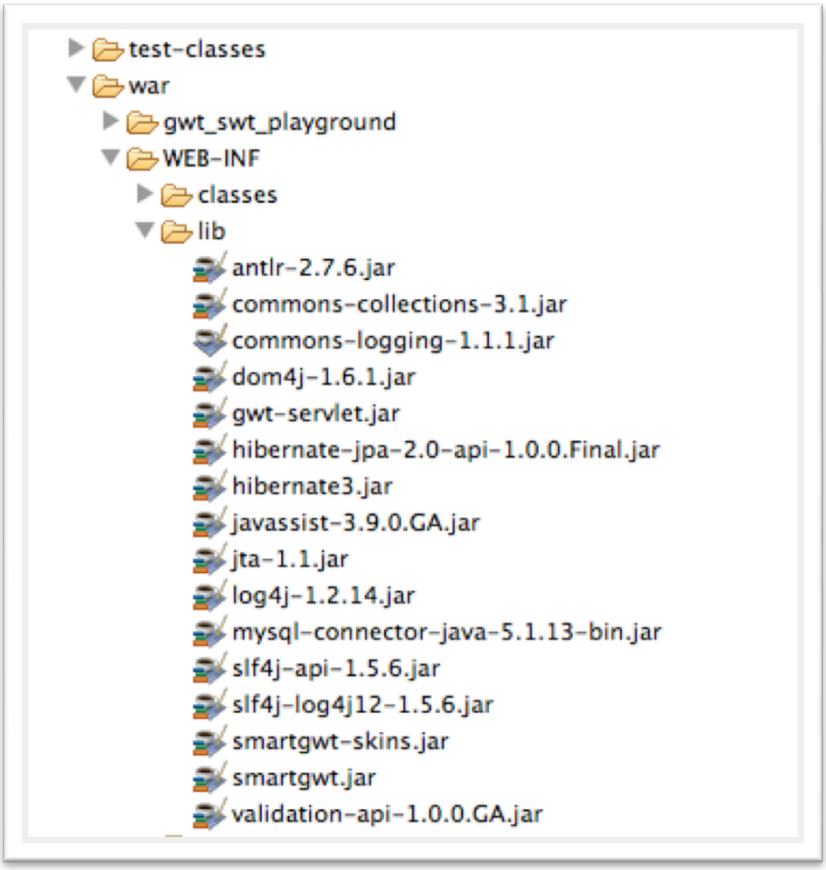
Create Web Application

0. *File → New… → Web Application Project*
   a. Enter project name
   b. Enter path com.company.project
   c. Uncheck *Use Google App Engine*
   d. Click *Finish*
   e. Modify the<title> and <h1> tags of the HTML file to reflect the name of the project

Add the Necessary JAR files

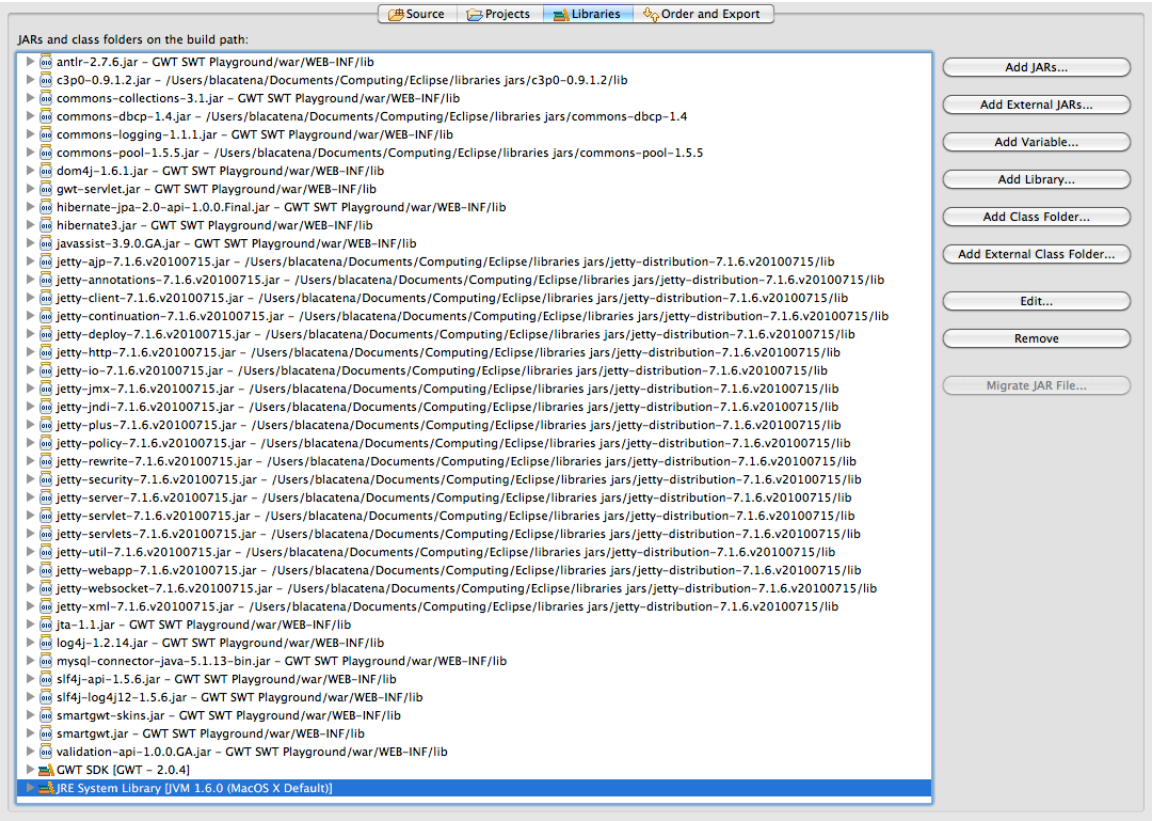
1. Copy the following jar files into the war/WEB-INF/lib folder. These jars are internal to the project (i.e. packaged in the war file) because they will be required no matter how or where the final war file is deployed.
   a. antlr-276.jar
   b. commons-collections-3.1.jar
   c. commons-logging-1.1.1.jar
   d. dom4j-1.6.1.jar
   e. hibernate-jpa-2.0-api-1.0.0.Final.jar
   f. hibernate3.jar
   g. javaassists-3.9.0.GA.jar
   h. jta-1.1.jar
   i. log4j-1.2.14.jar
   j. mysql-connector-ava.5.1.13-bin.jar
   k. slf4j-api-1.5.6.jar
   l. slf4j-log4j12-1.5.6.jar

m. validation-api-1.0.0.GA.jar



2. Add the new jars to the classpath
   a. Select the project, then *Properties*
   b. Select *Java Build Path*
   c. In the *Libraries* tab, Click *Add Jars…*
   d. Navigate to, select and add all of the jars in the war/WEB-INF/lib folder
3. Add external jars to the classpath. These jars are external to the project because they could vary with the production installation (e.g. Jetty versus Tomcat, or c3p0 versus another connection pool system).
   a. Select the project, then *Properties*
   b. Select *Java Build Path*
   c. In the *Libraries* tab, click *Add External Jars…*
   d. Navigate to, select and add each of the following jars:
      i. c3p0-0.9.1.2/lib/c3p0-0.9.1.2.jar
      ii. commons-dbcp-1.4/commons-dbcp-1.4.jar
      iii. commons-pool.1.5.5/commons-pool-1.5.5.jar
      iv. All jars from jetty-distribution-7.1.6.v20100715

4. Confirm that all jars are on the classpath as follows:



5. Set the Jetty InitialContextFactory in the runtime configuration:
   a. Select the Project, then right click to select
      Properties --> Run/Debug Settings
   b. Select the configuration and click *Edit* (this must be done for any new configurations created for new web pages or entry points, as well)
   c. Select the *Arguments* tab
   d. Append the following parameter to the VM Arguments area:

```
-Djava.naming.factory.initial=org.eclipse.jetty.jndi.InitialContextFactory
```

   e. Click OK
   f. Click OK

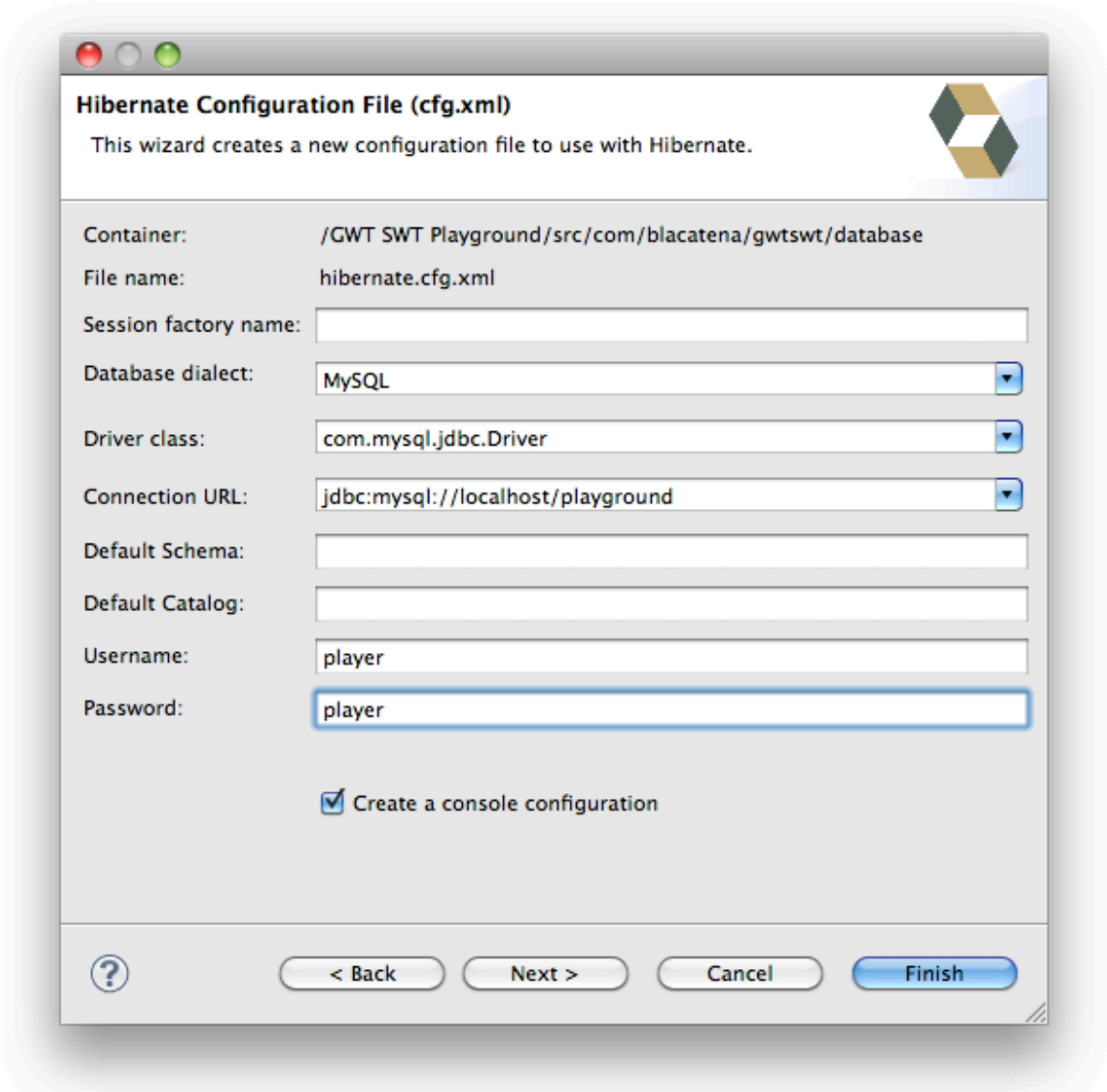
Create Hibernate Configuration and Settings

1. Create database folder (preferred location, under src/com/company/project
2. Either copy and edit a hibernate.cfg.xml file,

... or ...

Create hibernate.cfg.xml:

a. Select src folder
b. Select *New…* → *Other…*
c. In the dialog, select *Hibernate* → *Hibernate Configuration File (cfg.xml)*
d. Click *Next* to accept hibernate.cfg.xml within the database directory
e. Select:
    i. Database Dialect: MYSQL
    ii. Driver Class: com.mysql.jdbc.Driver
    iii. Enter database url (e.g. jdbc:mysql://localhost/dbname)
    iv. Enter user name and password
    v. Check "Create a console configuration"



vi. Click *Finish*

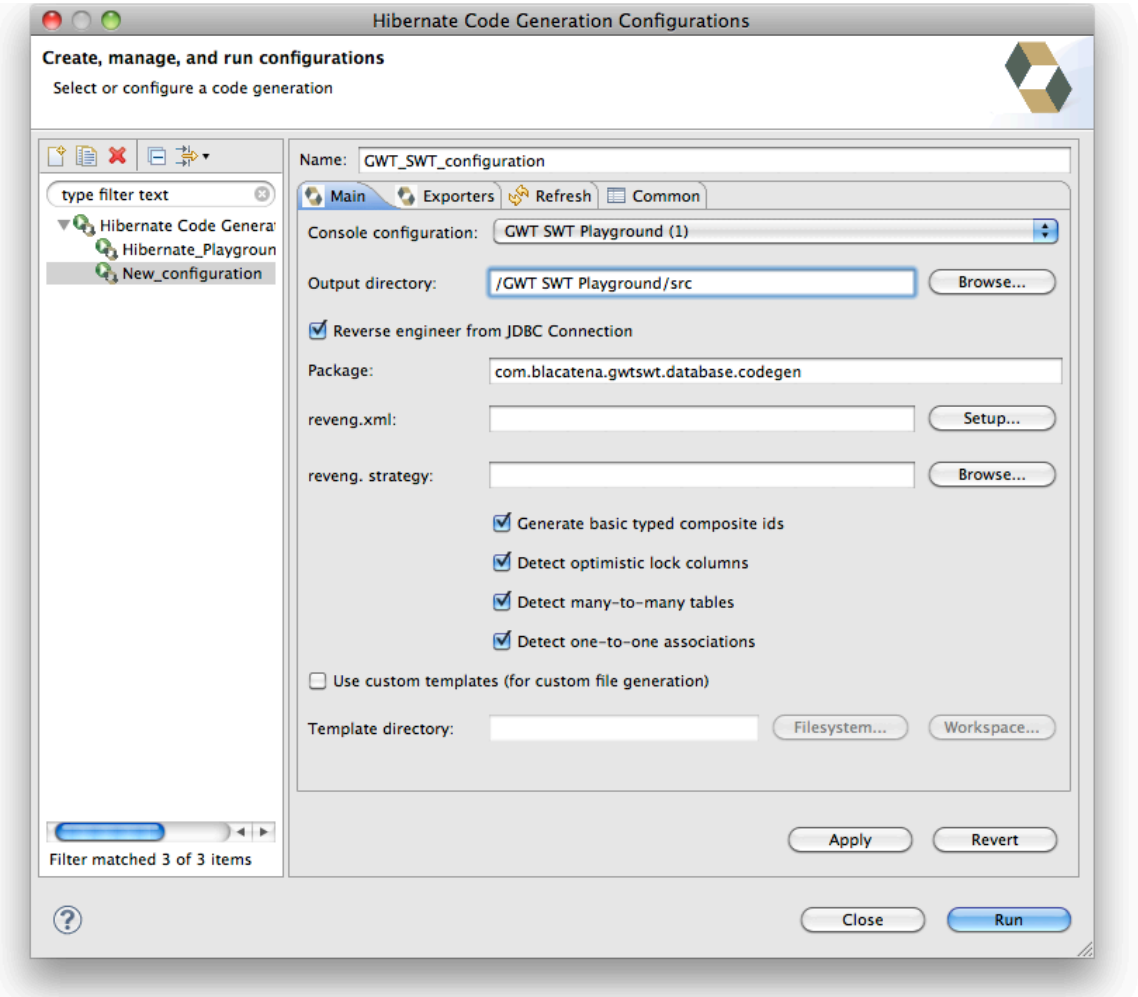
f. Copy / Edit other values, such as c3p0 configuration info

```
<property name="hibernate.bytecode.use_reflection_optimizer">false</property>
<property name="hibernate.c3p0.max_size">50</property>
<property name="hibernate.c3p0.min_size">3</property>
<property name="hibernate.c3p0.timeout">1800</property>
<property
name="hibernate.cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
<property name="hibernate.cache.use_minimal_puts">false</property>
<property name="hibernate.cache.use_query_cache">false</property>
<property name="hibernate.connection.autocommit">true</property>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.password">player</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost/playground</property>
<property name="hibernate.connection.username">player</property>
<property name="hibernate.current_session_context_class">thread</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.max_fetch_depth">3</property>
<property name="hibernate.show_sql">false</property>
```

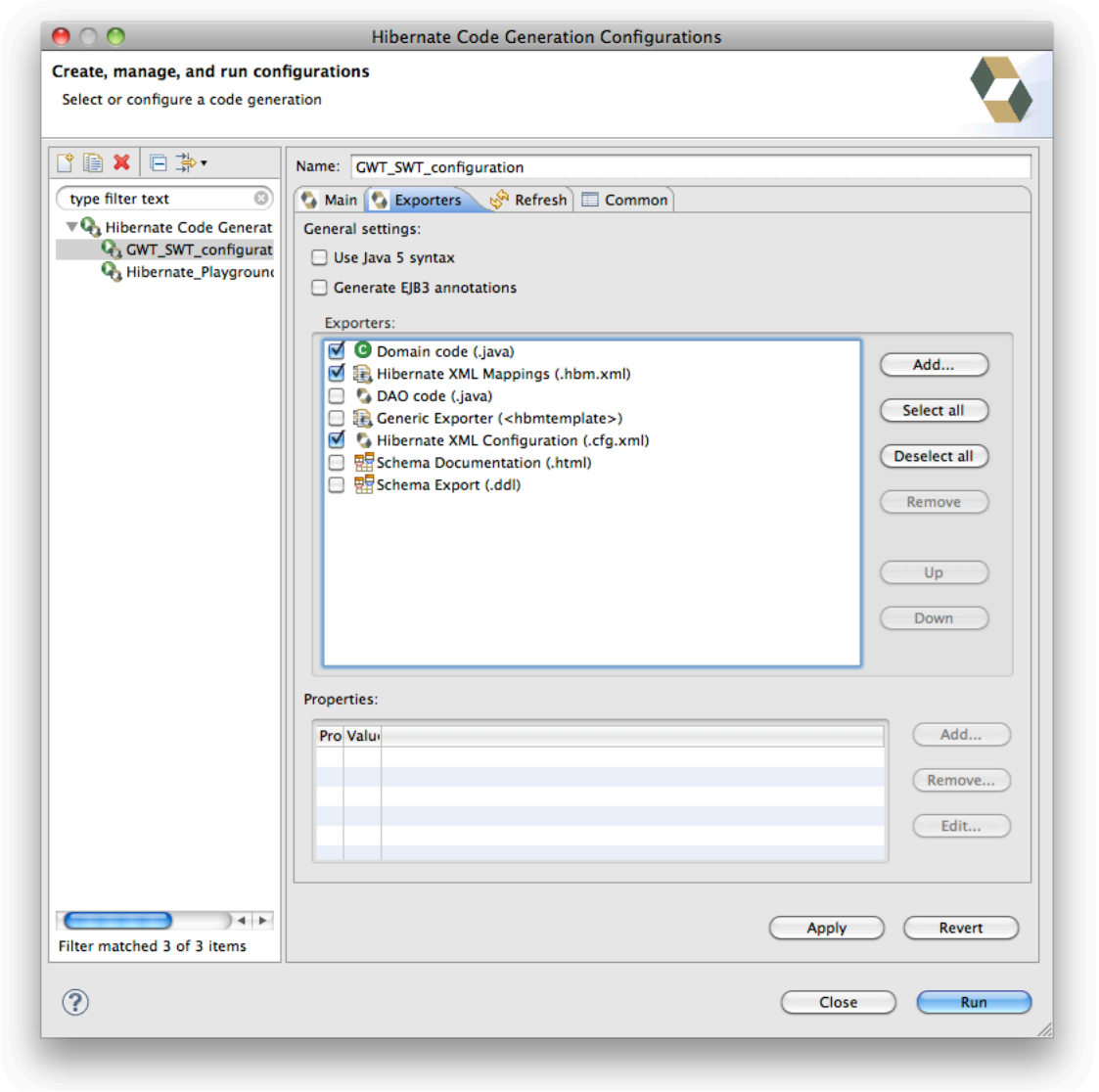
3. Edit the Hibernate Settings
   a. Select the project
   b. Right click, select *Properties*
   c. Select *Hibernate Settings*
      i. Check the "Enable Hibernate 3" checkbox
      ii. Select the just created console configuration
4. Create a Hibernate Code Generation Configuration
   a. Select either *Run* → *Hibernate Code Generation…* → *Hibernate Code Generation Configurations*

      … or …

      Use the equivalent toolbar drop down selection 
   b. Click the icon to create a new configuration
      i. Main tab
         1. Set the name
         2. Browse to the src directory in the project
         3. Check the "Reverse Engineer" checkbox



4. Enter the package name, usually as com.company.project.database.codegen



ii. Exporters tab
    1. Check *Hibernate XML Mappings (.hbm.xml)*
    2. Check *Domain code (.java)*
    3. Check *Hibernate XML Configuration (.cfg.xml)*
    4. Optionally check *DAO Code (.java)*
iii. Click *Apply* to save the changes
iv. Click *Run* to test the code generation (and database access)
v. Check for the necessary and valid and correctly compiling .java, .hbm.xml and .cfg.xml files

5. Copy in the two base Java Hibernate utility classes into src/company/project/database/util
    a. HibernateUtil.java
    b. HibernateAccessor.java
6. Create or copy into the src directory the log4j.properties file (change the log name, location, or settings, as desired):

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
                                                                %m%n

log4j.appender.file=org.apache.log4j.FileAppender
#log4j.appender.file.DatePattern='.'yyyy-MM-dd
#log4j.appender.file.File=logs/sli.log
#log4j.appender.file.File ${logs.modifier}/gwtswtplayground.log
log4j.appender.file.File=/var/tmp/gwtswtlog.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%x %p %d{yyyy/MM/dd HH:mm:ss} %c
- %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

#TURN OFF log4j.rootLogger=warn
log4j.rootLogger=info, stdout, file

#DEFAULT log4j.logger.org.hibernate=info
log4j.logger.org.hibernate=error

### log HQL query parser activity
#log4j.logger.org.hibernate.hql.ast.AST=debug

### log just the SQL
#DEFAULT log4j.logger.org.hibernate.SQL=debug

### log JDBC bind parameters ###
#DEFAULT log4j.logger.org.hibernate.type=info

### log schema export/update ###
#DEFAULT log4j.logger.org.hibernate.tool.hbm2ddl=info

### log HQL parse trees
#log4j.logger.org.hibernate.hql=debug

### log cache activity ###
#DEFAULT log4j.logger.org.hibernate.cache=info

### log transaction activity
#log4j.logger.org.hibernate.transaction=debug

### log JDBC resource acquisition
#log4j.logger.org.hibernate.jdbc=debug

### enable the following line if you want to track down connection ###
### leakages when using DriverManagerConnectionProvider ###
#log4j.logger.org.hibernate.connection.DriverManagerConnectionProvider=trace

### log SLI upload manager activity
#log4j.logger.sliThreads.SliUploadManager=info
```

7. Create a jetty-web.xml file to define the datasource for the database:
   a. Copy or create a jetty-web.xml file into the war/WEB-INF folder
   b. Add or edit the datasource reference as follows (changing the database name, user and password as needed):

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure//EN"
"http://jetty.mortbay.org/configure.dtd">
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
   <New id="DSPlayground" class="org.eclipse.jetty.plus.jndi.Resource">
      <Arg></Arg>
      <Arg>jdbc/DSPlayground</Arg>
      <Arg>
         <New class="com.mchange.v2.c3p0.ComboPooledDataSource">
            <Set name="driverClass">com.mysql.jdbc.Driver</Set>
            <Set
name="jdbcUrl">jdbc:mysql://localhost:3306/playground</Set>
            <Set name="user">player</Set>
            <Set name="password">player</Set>
         </New>
      </Arg>
   </New>
</Configure>
```

8. Edit the web.xml file in war/WEB-INF to add a resource reference for the datasource by inserting (after the servlet-mapping entries):

```
<resource-ref>
     <description>Playground DataSource Reference</description>
     <res-ref-name>jdbc/DSPlayground</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
</resource-ref>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <!-- Servlets -->
  <servlet>
    <servlet-name>greetServlet</servlet-name>
    <servlet-class>com.blacatena.gwtswt.server.GreetingServiceImpl</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>greetServlet</servlet-name>
    <url-pattern>/gwt_swt_playground/greet</url-pattern>
  </servlet-mapping>

  <resource-ref>
     <description>Playground DataSource Reference</description>
     <res-ref-name>jdbc/DSPlayground</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
  </resource-ref>

  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>GWT_SWT_Playground.html</welcome-file>
  </welcome-file-list>

</web-app>
```

9. Create the Run configuration
   a. Select the project
   b. Right click and select *Run As → Web Application*
   c. If necessary terminate the run, and modify the created configuration by choosing *Run As → Run Configurations…*
   d. Add the following VM parameter:
   -Djava.naming.factory.initial=org.eclipse.jetty.jndi.InitialContextFactory

Test The Environment

1. Test Jetty startup and raw GWT and Hibernate access. Run the project as a Web Application, check the console for proper startup including Hibernate access (may include warnings, but no errors), and test the supplied URL for Ajax access.
2. Test Hibernate access by building basic database functionality into the default GWT server class.
   a. Create a src/company/project/database/util folder
   b. Copy the HibernateUtil.java and HibernateAccessor.java classes into the new util folder, and correct the package names so that they compile cleanly.

i. Modify the HibernateAccessor.java class to reference GEN_BASE string value "com.company.project.database.codegen".

c. Create a src/company/project/database/accessors (or helpers or tables or objects) folder

d. Create an accessor helper class for a database table such as:

```
public class DbUser extends HibernateAccessor {

    static String objectName = User.class.getSimpleName();

    public static User getByUserName(String userName) {
        return (User) getByField(objectName, "userName", userName,
"lastName");
    }

    public static List<User> findByLastName(String lastName) {
        List<Object> results = findByField(objectName, "lastName",
lastName, "firstName");
        List<User> users = new ArrayList<User>();
        for (int i = 0; i < results.size(); i++)
            users.add((User) results.get(i));
        return users;
    }
}
```

e. Modify the default GreetingServiceImpl.java class file to perform some simple database access, such as:

```
private int getUserCounter(String userName) {
    HibernateUtil.openSession();
    HibernateUtil.startTransaction();
    User user = DbUser.getByUserName(userName);
    if (user == null) {
         System.out.println(userName + " not found.");
         user = new User();
         user.setUserName(userName);
         user.setPassword(userName + "0");
         user.setFirstName("Poindexter");
         user.setLastName("Twain");
         user.setLoginCount(0);
    }
    user.setLoginCount(user.getLoginCount() + 1);
    DbUser.persist(user);
//  DbUserHelper.persist(user);
    HibernateUtil.endTransaction();
    HibernateUtil.closeSession();

    return user.getLoginCount();
}
```

f. Run the web application and confirm database access works

Hibernate Access Notes

The following conventions are used for Hibernate database access.

First, Hibernate is configured to use the c3p0 pooling system.

The Hibername code generation tools are used to generate the Hibernate mapping files, as well as appropriate data objects to represent table row data instances. When database changes are made, all that need be done is to rerun the project's Hibernate Code Generation Configuration from the Hibernate dropdown icon  , or from the Run → Hibernate Code Generation menu.

A modified HibernateUtil class is used to construct hibernate sessions, and a HibernateAccessor class is used to provide for the most common database functions (persist, refresh, getById, getByExample).

The HibernateAccessor class may and should also be extended to work with particular tables, and so perform access on specific tables without requiring type casting. It can, in simple cases, be used without extension (and with inline type casting).

Note that HibernateAccessor differs very slightly from the Home methods created by Hibernate Code Generation in that it uses the HibernateUtil class to properly create sessions (as required by Tomcat),

HibernateUtil and HibernateAccessor should both be placed in the com.company.project.database.util package.

To start any database session, call HibernateUtil.openSession() to start a session, and always call HibernateUtil.closeSession() to release the session to the pool.

Also call HibernateUtil.startTransaction() to start a transaction, and HibernateUtil.endTransaction() to end the transaction.

For simple access, the HibernateAccessor can be extended, primarily to extend any get or find methods, and to return either an object or list of the appropriate type, by type casting. New finder methods can also be written with specific Hibernate functionality.

There is no need to extend the persist or other methods, as these access Object parameters, however they can be extended to provide for some level of type validation/checking.

Set Up and Test a SmartGWT Project

This section will set up (and test the installation of) SmartGWT into a project.

1. Create a project, or use an existing project (in this example named GWT_SWT_Playground).
2. Add the SmartGWT jars to the project.
    a. Copy the following jar files into the war/WEB-INF/lib folder.
        i. smartgwt.jar
        ii. smartgwt-skins.jar
    b. Add these two jars to the classpath.
        i. Select the project, then *Properties*
        ii. Select *Java Build Path*
        iii. In the *Libraries* tab, Click *Add Jars…*
        iv. Navigate to, select and add the two smartgwt jars in the war/WEB-INF/lib folder
3. Configure SWT into the GWT module.
    a. Open and edit the GWT_SWT_Playground.gwt.xml file
    b. Add the following two lines after the first inherits element:

```
<!-- Inherit the SmartGWT Toolkit library configuration.        -->
<inherits name='com.smartgwt.SmartGwt'/>
```

4. Test the SWT functionality by modifying the sample GWT_SWT_Playground.java file.
    a. Comment out the existing onModuleLoad() method
    b. Comment out the existing GWT library imports
    c. Add the following imports:

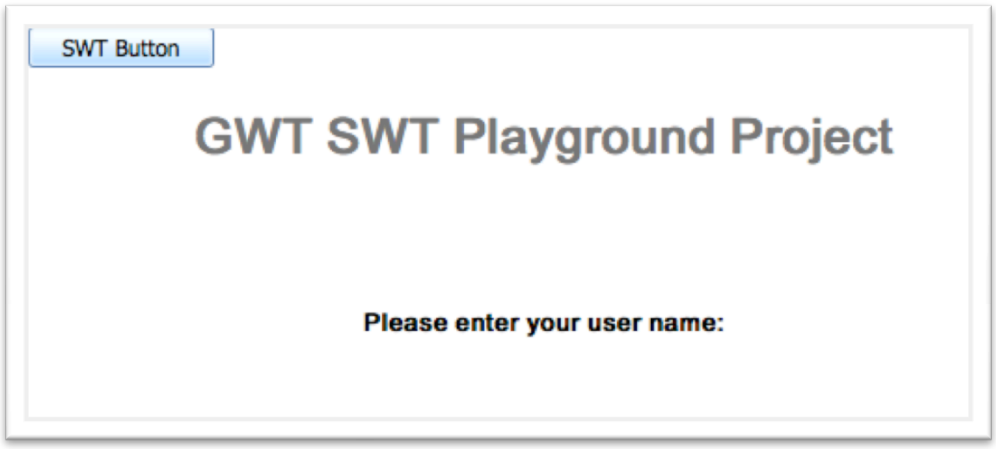
```
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.smartgwt.client.util.SC;
import com.smartgwt.client.widgets.IButton;
import com.smartgwt.client.widgets.events.ClickEvent;
import com.smartgwt.client.widgets.events.ClickHandler;
```

    d. Add the following onModuleLoad() method:

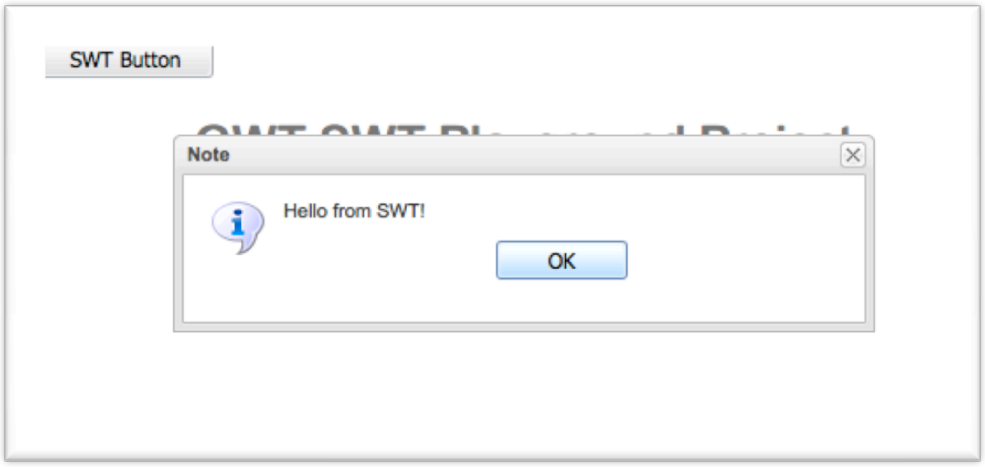
```
public void onModuleLoad() {
        IButton button = new IButton();
        button.setTitle("SWT Button");
        button.addClickHandler(new ClickHandler(){

            public void onClick(ClickEvent event) {
                SC.say("Hello from SWT!");
            }

        });
        button.draw();
```

```
}
```

e. Run the web application and test the result (note new SWT button in the upper left corner, and no GWT field or button):



Click the SWT button to see:



Set Up and Test an ExtGWT Project

This section will set up (and test the installation of) ExtGWT into a project.

1. Create a project or use an existing project (in this example, named GWT_Ext_Playground).
2. Add the extGWT jar to the project.
    a. Copy the following jar file into the war/WEB-INF/lib folder.
        i. gxt.jar
    b. Add that jar to the classpath.
        i. Select the project, then *Properties*
        ii. Select *Java Build Path*
        iii. In the *Libraries* tab, Click *Add Jars…*
        iv. Navigate to, select and add the gxt jar in the war/WEB-INF/lib folder
3. Configure SWT into the GWT module.
    a. Open and edit the GWT_Ext_Playground.gwt.xml file
    b. Add the following two lines after the first inherits element:

```
<!-- Inherit the ExtGWT Toolkit library configuration.        -->
<inherits name='com.extjs.gxt.ui.GXT'/>
```

4. Create a folder in the war directory named resources, and import the resources contents from the Ext GWT download into that folder.
5. Test the ExtGWT functionality by modifying the sample GWT_Ext_Playground.java file.
    a. Comment out the existing onModuleLoad() method
    b. Comment out the existing GWT library imports
    c. Add the following imports:

```
import com.extjs.gxt.ui.client.util.Params;
import com.extjs.gxt.ui.client.widget.button.Button;
import com.extjs.gxt.ui.client.event.*;
import com.extjs.gxt.ui.client.widget.Info;
import com.extjs.gxt.ui.client.widget.MessageBox;
import com.extjs.gxt.ui.client.event.MessageBoxEvent;
```

d. Add the following onModuleLoad() method:

```
public void onModuleLoad() {
    Button button = new Button("Ext GWT Button",
            new SelectionListener<ButtonEvent>() {
                public void componentSelected(ButtonEvent ce) {
                    final MessageBox box = MessageBox.prompt(
                            "Comment", "Please enter a comment:");
                    box.addCallback(new Listener<MessageBoxEvent>() {
                            public void handleEvent(MessageBoxEvent be) {
                                Info.display("MessageBox",
                                    "You entered '{0}'",
                                    new Params(be.getValue()));
                            }
                    });
                    }
            });

    RootPanel.get("myButtonContainer").add(button);
}
```

e. Add the following line to the GWT_Ext_Playground.html file:

```
<link rel="stylesheet" type="text/css" href="resources/css/gxt-all.css" />
```

f. Rename the page (in the H1 tag), remove all of the table declarations from GWT_Ext_Playground.html, and insert a single, empty DIV with the id "myButtonContainer":

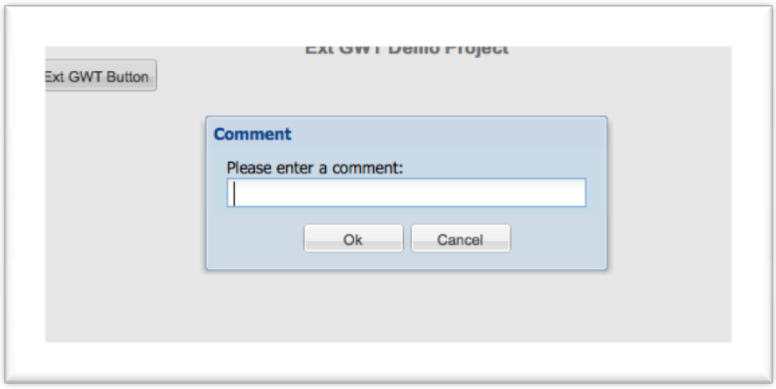
```
<div id="myButtonContainer"></div>
```

g. Optionally, to use charts (not done in this sample), add the following line to the HTML:

```
<script language="javascript" src="resources/flash/swfobject.js"></script>
```

h. Run the web application and test the result (note new SWT button in the upper left corner, and no GWT field or button):

Click the Ext button to see:



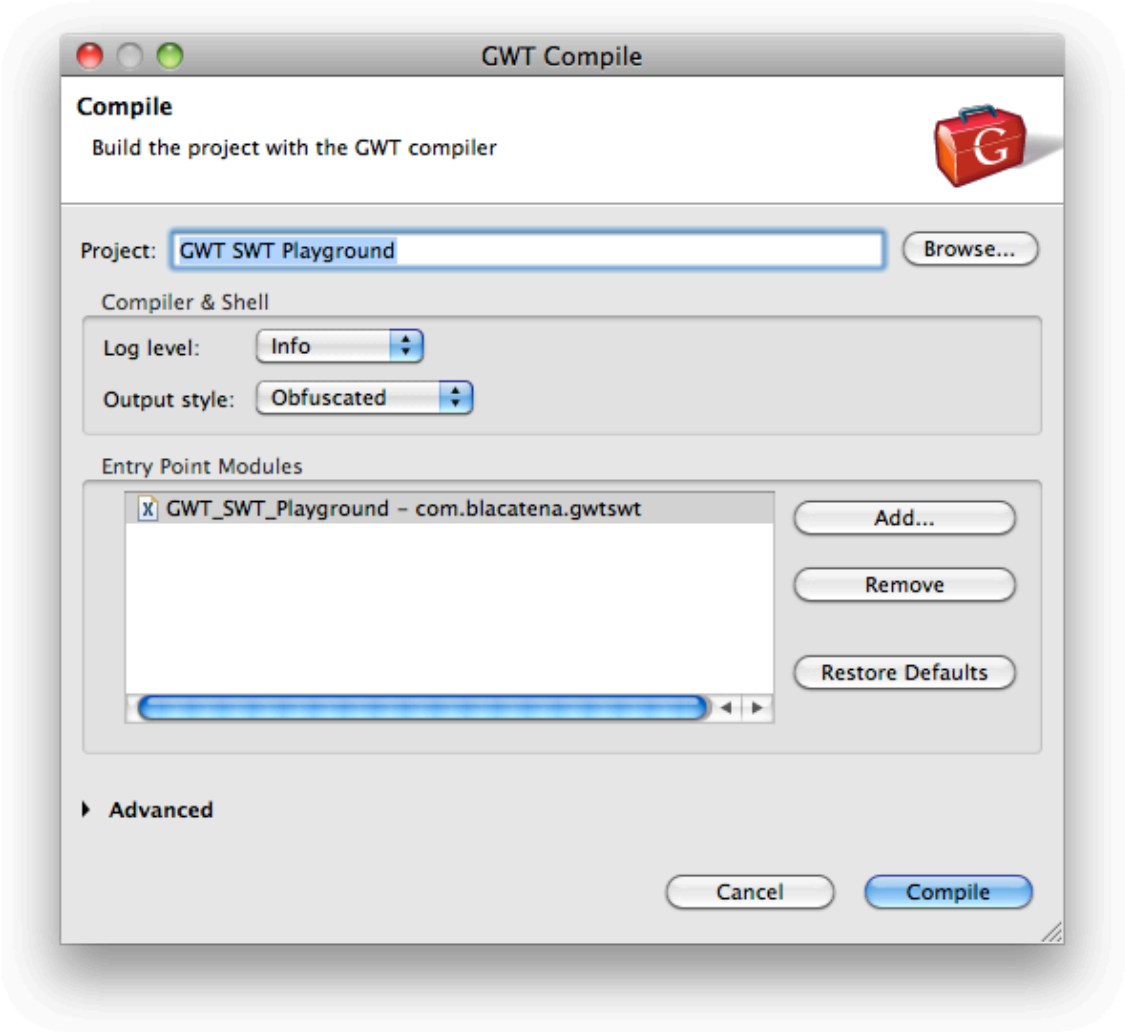
Enter a string and click the OK button.

Deploying a Project to Tomcat

1. Compile the production code:
   a. Select the project, right click *Google* → *GWT Compile*

      … or …

      Select the project and click the red *GWT* icon in the toolbar
   b. Click the *Compile* button



2. Construct the war file
   a. Copy the project war file to another location and rename if/as desired.

   b. Either deploy any external jars (c3p0, commons-dbcp, commons-pool) to the shared library directory in the Tomcat setup, or else copy them to the WEB-INF/lib directory within the copy of the war folder.
   c. Compress the contents (but not the folder itself) of the copy of the war folder into a zip file.
   d. Rename the zip file with the name of the intended webapp and a .war extension (e.g. "mywebapp.war").
3. Add the data source resource reference to the Tomcat server.xml file, such as:

```
<!-- Added for GWT SWT Playground  -->
  <Resource name="jdbc/playground" auth="Container" type="javax.sql.DataSource"
        maxActive="100" maxIdle="30" maxWait="10000"
            removeAbandoned="true"
            removeAbandonedTimeout="60"
            logAbandoned="true"
            username="player" password="player"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/player?autoReconnect=true"/>
   <!-- END Added for ScholasticLibraryInsight-->
```

4. Deploy the war file to Tomcat, HOWEVER…

> *Note: The web application expects to run on port 80, while Tomcat runs on port 8080. To run directly on port 80, Tomcat must be started as root, which is a security risk. Alternately, varying strategies can be used to redirect servlet access, or servlet access and other static HTTP requests to tomcat, using Apache and mod_proxy or mod_jk. As the exact solution depends on the production environment requirements, the steps to do so will not be listed here.*

Redirecting port 80 to Tomcat

One simple way to redirect port 80 requests for a particular web app (or all web apps) is to configure httpd.conf, or a supplemental conf file in "other" as follows:

```
<IfModule proxy_module>
    ProxyPass /<webappname> http://localhost:8080/<webappname>/
    ProxyPassReverse /<webappname> http://localhost:8080/<webappname>/
</IfModule>
```

or, to redirect everything to Tomcat (not just one particular web application):

```
<IfModule proxy_module>
    ProxyPass / http://localhost:8080/
    ProxyPassReverse / http://localhost:8080/
</IfModule>
```

On Max OS X 10.6, for this to wok one must start both Tomcat and Apache. Tomcat is started by navigating to the Tomcat directory ($CATALINA_HOME, usually /Library/Tomcat/Home) and executing bin/startup.sh. Apache can be started by using the *Sharing* panel in *System Preferences*, and checking *Web Sharing*.

On Mac OS X 10.6, this can be done by creating a file named /etc/apache2/other/proxy.conf with these lines. There do seem to be occasional, random problems serving the css and other files associated with this. This appears to be rectified either by restarting the web server, or by accessing things first through port 8080, then port 80.

Redirecting Servlets to Tomcat

It's possible to allow Apache to more efficiently serve static resources (HTML, images, js) by deploying the project to both Tomcat and a location from which Apache can find the static files, and redirecting only servlet request, such as:

```
<IfModule proxy_module>
    ProxyPass /webapp/servlet http://localhost:8080/webapp/servlet
    ProxyPassReverse /webapp/servlet http://localhost:8080/webapp/servlet
</IfModule>
```

Documentation References

See the following useful links that played some role in developing this document/approach:

Open Source Libraries

Apache Commons Logging	http://commons.apache.org/logging/
Apache Commons DBCP	http://commons.apache.org/dbcp/
Apache Commons Pool	http://commons.apache.org/pool/
Simple Logging Facade for Java (SLF4J)	http://www.slf4j.org/

Tools and Jars

Eclipse	http://eclipse.org
GWT	http://code.google.com/webtoolkit
SWT	http://www.smartclient.com/
Ext GWT	http://www.sencha.com/products/gwt/
Hibernate	http://hibernate.org
Jetty	http://eclipse.org/jetty/
Tomcat	http://tomcat.apache.org/

How To Pages

Link Apache and Tomcat
Apache Mod Proxy
Apache Tomcat Connectors How To
Apache Tomcat-6 with Eclipse
Internet Security for GWT
Deploying GWT Apps
GWT Tutorial
Communicate with a Server
Jetty Tutorial
Setting up SmartGWT
Jetty Through Eclipse

Assorted Useful Things to Do

Check apache version	httpd –v
Apache modules location on Mac OS X 10.6	/usr/libexec/apache2
Apache conf location on Max OS X 10.6	/etc/apache2
Apache additional conf loc on Max OS X 10.6	/etc/apache2/other