

Exadel Studio Pro 3.5

Getting Started Guide for JSF with Hibernate

In this guide, we will show you how to take a very simple ready-made JSF application and convert it to use a database with the help of Exadel Studio. After downloading, we will first set up and run the application without persistence, and then with persistence.

The application itself is a simple JSF-based application that asks the user to enter a User ID. It tries to locate a record for the entered User ID (entered during the application session). If the record is found, details are displayed. If the record is not found, you are asked to create this record. This application, of course, only runs as long as the Tomcat server is running. Once we stop the server all of the data is lost as all information is saved only in the application session context.

With the help of Exadel Studio, we will convert this application to use the lightweight hsqldb database (included with the downloaded project). We will use Exadel Studio special features for object-relational mapping for this conversion. After the conversion, even if we restart the server, the data we entered will have been saved in a database and thus available to the application.

Before we start, we assume that you have Eclipse, and have installed Exadel Studio Pro with Tomcat server.

Installing the Project

We are first going to download and import this project (ormHibernate3-jsf) into Eclipse.

1. Download:

<http://webdownload.exadel.com/dirdownloads/ormhib/examples/ormHibernate3-jsf.zip>

2. Unzip this file into your Eclipse workspace folder.
3. Launch Eclipse.
4. In Eclipse, select **File/Import/JSF Project**.
5. Click **Next**.
6. Browse to where the project was unzipped.
7. Find the `web.xml` file inside the `WebContent` folder in the `WEB-INF` folder, select it, and Click **Finish**.

Exadel Studio Pro 3.5

The `ormHibernate3-jsf` project should appear in the Package Explorer with a standard Web application structure. As we mentioned before, this is a JSF application. To see the JSF configuration file, browse to **WebContent/WEB-INF/faces-config.xml** .

In the **JavaSource** folder, you will find the following Java source files. We will briefly explain these files and then run the application.

demo/Address.java	holds the user address.
demo/User.java	holds user information.
demo/GetUserIdBean.java	holds user Id and determines navigation (JSF Backing Bean).
demo/UserFormBean.java	holds new user input (JSF Backing Bean).
demo.bundle/Messages.properties	holds label messages used in the application.

Running Without a Database

We are ready to run this project in a Web browser and see how it looks. We don't need to compile these classes, because Eclipse did it for us when we imported the project.

8. Start Tomcat.
9. Click on the running-man-and-blue-butterfly icon from the toolbar.
10. Go ahead and play with the application.

Initially users don't exist, so entering any ID will prompt you to enter user details. Once you have saved a user, you can go back to the main page by clicking on the [Back to Login Page](#) link. If you then enter that user's id again, the application will locate and display the user's details.

Converting for Use With a Database

Now we are ready to convert this application to use with a database with the help of Exadel Studio. To convert the application for use with a database, we need to set things up in three different places:

- The Application
- The Database
- The Application Server

Setting up the Application

Let's start by using Exadel Studio with the application project. First, we create the object/relational mapping from our simple object model to a database schema after adding Hibernate capabilities to our project.

11. Right-click on **ormHibernate3-jsf** in the Package Explorer view and select **Exadel Studio/Add Hibernate Capability...** from the context menu.
12. Click on **Yes** in the the dialog box with **Add Hibernate Jars** selected.
13. In the Configuration Wizard, click twice in the **Value** field for **dialect** and select **org.hibernate.dialect.HSQLDialect** from the pop-up menu.
14. Click **Finish**
15. Select **Object to Schema** for the Mapping Approach.

We are only interested in saving the User class in a database, so we are going to create a mapping for the User class to a database table.

16. In the **Persistent Classes Wizard** dialog that appears next, click on the **Select Classes...** button.
17. In the dialog box, expand **demo**, click the check box for the **User** class, and click **OK**.
18. Back in the wizard, click **Next**.
19. Leave all other values as they are in the next dialog box and click **Finish**.

Edit the Hibernate Configuration

Afterwards, the Hibernate configuration file, `hibernate.cfg.xml`, will appear in an editor window. So, let's adjust this file first.

Note: In code listings, lines may wrap in the PDF page that shouldn't wrap, but these cases should be clear and easy to adjust for.

20. Replace this line:

```
<property  
name="hibernate.connection.url">jdbc:hsqldb:hsq1:[hostname]</  
property>
```

With this:

```
<property name="hibernate.connection.url">jdbc:hsqldb:hsq1://  
localhost</property>
```

Your file should now look like this:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration  
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
  
<hibernate-configuration>  
  <session-factory>
```

Exadel Studio Pro 3.5

```
<property
name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</
property>
  <property name="hibernate.connection.url">jdbc:hsqldb:hsql://localhost</
property>
  <property name="hibernate.dialect">org.hibernate.dialect.HSQLDialect</
property>
  <property name="hibernate.connection.username">sa</property>
  <property name="hibernate.connection.password"></property>
  <property name="hibernate.connection.datasource">java:comp/env/jdbc/
kickstart</property>
  <mapping resource="demo/User.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

21. Save the file.

Edit the Mapping File

Next, we need to make on slight change to the mapping file, `User.hbm.xml`.

22. In the ORM Explorer view, reveal the **ormHibernate3-jsf/JavaSource/hibernate.cfg.xml/demo/User -> user** node, right-click it, and select Open Mapping from the context menu.
23. In the editor that opens up for the mapping file, just change the `class` attribute for generator to a value of `assigned` and you're done with this file.

Here is what the edited `User.hbm.xml` file should now look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="demo">
  <class name="User" table="user" optimistic-lock="none">
    <id name="id" type="string" unsaved-value="null" column="id">
      <generator class="assigned"/>
    </id>
    <property name="firstName" type="string" column="first_name"/>
    <property name="lastName" type="string" column="last_name"/>
    <property name="email" type="string" column="email"/>
    <component name="address" update="true" insert="true" class="demo.Address">
      <property name="city" type="string" column="address_city"/>
      <property name="state" type="string" column="address_state"/>
      <property name="street" type="string" column="address_street"/>
      <property name="zip" type="string" column="address_zip"/>
    </component>
  </class>
</hibernate-mapping>
```

24. Save the file.

Add a General Class for Incorporating Hibernate

Next, we will need to create a special Java class for incorporating Hibernate into our application.

Exadel Studio Pro 3.5

25. Switch to the Package Explorer view and create the class, `HibernateHelper.java`, in **JavaSource/demo** with this content and save it.

```
package demo;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateHelper {
    /**
     * Reference to SessionFactory.
     */
    private static SessionFactory sf;
    public static final ThreadLocal session = new ThreadLocal();
    public static synchronized void init() {
        if (sf != null) return;
        System.out.println("Initializing Hibernate");
        try {
            Configuration cfg = new Configuration().configure();
            sf = cfg.buildSessionFactory();
        } catch (Exception he) {
            System.err.println("Unable to create session factory from
configuration");
            he.printStackTrace();
            throw new RuntimeException("Unable to create session factory from
configuration", he);
        }
        System.out.println("Hibernate initialized");
    }
    /**
     * Return the SessionFactory.
     * @return The SessionFactory for this application session
     */
    public static SessionFactory sessionFactory() {
        if (sf == null) init();
        return sf;
    }
    public static void destroy() {
        if (sf != null) {
            try {
                sf.close();
            } catch (HibernateException he) {
                he.printStackTrace();
            }
        }
        sf = null;
        System.out.println("Hibernate resources released");
    }
    /**
     * Closes an hibernate {@link Session}, releasing its resources.
     * @throws HibernateException if an hibernate error occurs
     */
    public static void closeSession() throws HibernateException {
        Session s = (Session)session.get();
        session.set(null);
        if (s != null) {
            s.close();
        }
    }
}
```

Exadel Studio Pro 3.5

```
    }
}
/**
 * Returns an hibernate {@link Session} from the session factory.
 * @return an hibernate {@link Session}
 * @throws HibernateException if an error occurs
 */
public static Session openSession() throws HibernateException {
    if (sf == null) init();
    Session s = (Session)session.get();
    if (s == null) {
        s = sf.openSession();
        session.set(s);
    }
    return (s);
}
}
```

Edit the Two Bean Classes

We also need to modify the two bean classes in our application to “Hibernate-ize” them.

26. Modify the **GetUserIdBean.java** class in **JavaSource/demo** by adding these imports:

```
import org.hibernate.Session;
import javax.faces.application.FacesMessage;
```

27. Then, replace the **action()** method with this code and save.

```
public String action()
    throws Exception
{
    UserFormBean ufb;
    User user;
    String actionResult = "inputuser"; // new user by default

    Map sessionMap =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
    if (sessionMap != null) {

        ufb = new UserFormBean();
        ufb.setId(id);

        try {
            Session hSession = HibernateHelper.openSession();
            user = (User)hSession.get(User.class, id);
            HibernateHelper.closeSession();
        } catch (Exception e) {
            FacesContext context = FacesContext.getCurrentInstance();
            context.addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_ERROR,e.toString(), null));
            return "failed";
        }

        if (user==null || !user.getId().equals(id)) {
            user = new User();
            user.setId(id);
            sessionMap.put("user", user);
        } else {
```

Exadel Studio Pro 3.5

```
// fill UserFormBean with user information
loadUser(ufb,user);
actionResult="greeting";
}
sessionMap.put("UserFormBean", ufb);
}
return actionResult;
}

private void loadUser(UserFormBean userForm,User user) {

userForm.setId(user.getId());
userForm.setFirstName(user.getFirstName());
userForm.setLastName(user.getLastName());
userForm.setEmail(user.getEmail());
userForm.setStreet(user.getAddress().getStreet());
userForm.setCity(user.getAddress().getCity());
userForm.setState(user.getAddress().getState());
userForm.setZip(user.getAddress().getZip());
}
```

28. Modify the **UserFormBean.java** class in **JavaSource/demo** by adding these imports:

```
import org.hibernate.Session;
import org.hibernate.Transaction;
```

29. Then, replace the **save()** method with this code and save the class.

```
public String save() throws Exception {
    Map sessionMap =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
    if (sessionMap != null) {
        UserFormBean ufb=(UserFormBean) sessionMap.get("UserFormBean");

        try {

            Session hSession = HibernateHelper.openSession();
            Transaction tran = hSession.beginTransaction();
            User user = (User)hSession.get(User.class, id);
            if (user == null) {
                user = new User();
                user.setId(id);
                hSession.save(user);
            }
            saveUser(ufb, user);

            tran.commit();
            HibernateHelper.closeSession();
        } catch (Exception e) {
            return "failed";
        }
    }
    return "greeting";
}
```

Setting up the Database

To set up the database end, we need to use Exadel Studio and our HSQL database engine to create a database table corresponding to the class we are trying to persist and make the database available.

Creating the Database Table

Let's first create the script for our database table in Exadel Studio.

30. In the ORM Explorer view, right-click on **JavaSource/hibernate.cfg.xml** and select **Generate DDL Wizard...**
31. Select **HSQL** as the **Dialect** and leave **Location** as is.
32. Click **Finish**.

A DDL file called `schema.sql` will be created in the root of the project and will be opened in an editor window.

Making the Database Available for the Application

The database server, HSQLDB, is provided with the project. It's located in the `ormHibernate3-jsf/hsqldb` folder.

33. Start the database server:
`.../ormHibernate3-jsf/hsqldb/bin/server.bat`

34. In a separate window, start the admin tool:
`.../ormHibernate3-jsf/hsqldb/bin/dbadmin.bat`

This will launch a small GUI application, HSQL Database Manager.

35. Leave all values as they are, only change **URL:** to the following:
`jdbc:hsqldb:hsql://localhost`
36. Click **OK**.
37. Select **File/Open Script...** from the menu bar of HSQL Database Manager.
38. Find and open the the DDL file we just created.
39. Click **Execute** back in the main screen of the Database Manager.
40. Select **View/Refresh Tree** from the menu bar.

The User database should now appear in the expand/collapse tree to the left.

41. Select **File/Exit** from the menu bar.
42. Stop the database server:
`.../ormHibernate3-jsf/hsqldb/bin/shutdown.bat`

Setting up the Application Server

Finally, we need to set up the application server before we can run the database-enabled application in a Web browser.

43. Stop the Tomcat server, if it's running.
44. Then, copy `.../ormHibernate3-jsf/hsqldb/lib/hsqldb.jar` to your Tomcat `.../common/lib` folder. (If you are using the Tomcat server included with Exadel Studio Pro, the Tomcat folder will be under the Exadel Studio Pro install folder.)

Running Our New Application

45. Start the database server:
`.../ormHibernate3-jsf/hsqldb/bin/server.bat`
46. Start the Tomcat server.
47. Run the application.

Play with the application. Restart Tomcat and the database server. If you run the application again and enter a user that you already saved, the application should retrieve it from the database and display its details.