Developing user interface with JSF

This tutorial illustrates the following aspects of JSF:

- Various UI components of JSF (Text box, Combo box, Select box, Radio button, submit button).
- Associating validations to UI components.
- Configuring navigation rules using eclipse wizard.
- Configuring JSF managed beans.
- Associating managed bean actions to events.
- Using JPA to persist the data captured in JSPs through managed beans.

The tutorial is divided into the following sections.

- Setting up the Eclipse environment
- Creating WEB application with entities
  - Creating a dynamic Web project and defining JSF capabilities
  - Implementing persistence for the application
  - Defining UI components
  - Configuring JSF managed beans and associating managed bean actions to events
- Setting up the database tables and the Datasource
- Deploying the (war) application
- Running the application

In order to develop, deploy and run the application, the following environment is required.

- Sun JDK 5.0+ (J2SE 1.5)
- Eclipse 3.3.1.1 (Eclipse Classic package of Europa distribution), which is platform specific
- Web Tools Platform (WTP) 2.0.1
- Data Tools Platform (DTP) 1.5.1
- Eclipse Modeling Framework (EMF) 2.3.1
- Graphical Editing Framework (GEF) 3.3.1

The entire application can be downloaded from this link.

Setting up the Eclipse environment

1. Download Apache Geronimo and install it on the server. Look into Getting and installing Geronimo for instructions.
2. Install the eclipse IDE and download geronimo eclipse plugin and install it on top of eclipse. Look into Installing Geronimo Eclipse Plugin for instructions.
3. Create a runtime environment for Apache Geronimo in Eclipse. Look into Defining Geronimo server runtimes and servers for instructions to install a runtime for Apache Geronimo.

CreatingWEB application with entities

This section is organized in the following steps:

Creating a dynamic Web project and defining JSF capabilities
1. Start the eclipse wizard and right click on the *Project Explorer* and click on the **New -> Dynamic Web Project**
2. Enter the project name as *EmployeeWEB* and click on the **Next** button.

![New Dynamic Web Project](image1.png)

3. On the *New Dynamic Web Project* wizard, check on the **Project Facet** checkboxes and select the version values as below screen shot, and click on the **Next** button.

![New Dynamic Web Project](image2.png)
4. On the **Web Module** wizard, make sure that **Generate Deployment Descriptor** is checked as below and click on the **Next** button.

5. On the **Geronimo Deployment Plan** wizard, provide the moduleId values as below screen shot and click on the **Next** button.

6. On the **JSF Capabilities** wizard, check the second radio button and click on the **New** button by the side of the combo box. This will open **Create JSF Implementation Library** wizard.

7. Provide the **Library Name** as **MyJSFLibrary** and add the following jars by clicking on the **Add** button. Finally, click on the **Finish** button.
8. Now, on the **JSF Capabilities** wizard, add the **URL Mapping Patterns** as *.jsf* and click on the **Finish** button.
9. This will create `EmployeeWEB` application in the `Project Explorer` as below screen shot.
10. Right click on the WEB-INF folder and navigate to New -> Folder and create a folder by name *td* as given in the screen shot below.

![Project Explorer](image)

11. Copy myfaces_core.tld and myfaces_html.tld files into the *td* folder. These *.tld* files are available in the myfaces-impl-1.2.6.jar file.
12. We are going to use JPA to connect to EmployeeDB database created in the embedded Derby database. JPA uses `persistence.xml` file for configuration. Create a `META-INF` folder in the `EmployeeWEB -> build -> classes` folder in the Project Explorer. The contents of the `persistence.xml` are as follows.

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <persistence-unit name="Employee" transaction-type="JTA">
        <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
        <non-jta-data-source>EmployeeDS</non-jta-data-source>
    </persistence-unit>
</persistence>
```

Implementing persistence for the application
1. Right click on the EmployeeWEB and navigate to New -> Class and create Employee.java as given in the below screen shot. Click on the Finish button after providing the values.

2. The contents of the com.jpa.sample.Employee are as follows. This is the entity class used with JPA for persistence.

```java
package com.jpa.sample;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {

@Id
public int empNo;
public String empName;
public String deptName;
public String sex;
public double salary;
public String band;

public int getEmpNo() {
```
```java
public void setEmpNo(int empNo) {
    this.empNo = empNo;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}

public String getDeptName() {
    return deptName;
}

public void setDeptName(String deptName) {
    this.deptName = deptName;
}

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}

public String getBand() {
    return band;
}

public void setBand(String band) {
    this.band = band;
}
```

3. Similarly, create `sample.jsf.Employee`. The contents of the class are as follows. This is the managed bean used by JSF.

```java
package sample.jsf;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javax.faces.event.ActionEvent;
import javax.faces.model.SelectItem;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

public class Employee {
    private int empNo;
    private String empName;
    private String deptName;
    private String sex;
    private double salary;
    private String band;

    public int getEmpNo() {
        return empNo;
    }

    public void setEmpNo(int empNo) {
        this.empNo = empNo;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public String getDeptName() {
        return deptName;
    }

    public void setDeptName(String deptName) {
        this.deptName = deptName;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String getBand() {
        return band;
    }

    public void setBand(String band) {
        this.band = band;
    }
}
```
public String empName;
public String deptName;
public String sex;
public double salary;
public String band;
ArrayList<SelectItem> deptOptions;

@PersistenceContext
private EntityManager em;

public Employee(){
    deptOptions = new ArrayList<SelectItem>();
    SelectItem option = new SelectItem("Inventory", "Inventory");
    deptOptions.add(option);
    option = new SelectItem("Production", "Production");
    deptOptions.add(option);
    option = new SelectItem("Accounts", "Accounts");
    deptOptions.add(option);
    option = new SelectItem("Finance", "Finance");
    deptOptions.add(option);
    option = new SelectItem("Marketing", "Marketing");
    deptOptions.add(option);
    option = new SelectItem("IncomeTax", "IncomeTax");
    deptOptions.add(option);
    option = new SelectItem("Engineering", "Engineering");
    deptOptions.add(option);
}

public int getEmpNo() {
    return empNo;
}

public void setEmpNo(int empNo) {
    this.empNo = empNo;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}

public String getDeptName() {
    return deptName;
}

public void setDeptName(String deptName) {
    this.deptName = deptName;
}

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
}

public double getSalary() {
    return salary;
}
public void setSalary(double salary) {
    this.salary = salary;
}

public String getBand() {
    return band;
}

public void setBand(String band) {
    this.band = band;
}

public void addEmployee(ActionEvent event){
    com.jpa.sample.Employee employeeDup =
    em.find(com.jpa.sample.Employee.class, this.empNo);
    if(employeeDup != null)
        throw new IllegalArgumentException
        ("Message : sample.jsf.Employee :
            Employee Already Exists ("+this.empNo+")");

    com.jpa.sample.Employee employee =
    new com.jpa.sample.Employee();
    employee.setBand(this.band);
    employee.setDeptName(this.deptName);
    employee.setEmpName(this.empName);
    employee.setEmpNo(this.empNo);
    employee.setSalary(this.salary);
    employee.setSex(this.sex);
    try{
        Context ctx = new InitialContext();
        UserTransaction ut =
            (UserTransaction)ctx.lookup("java:comp/UserTransaction");
        ut.begin();
        em.persist(employee);
        ut.commit();
    }catch (Exception e){
        throw new IllegalArgumentException
        ("Message : sample.jsf.Employee : Exception :"+e);
    }
}

public void editEmployee(ActionEvent event){
    try{
        Context ctx = new InitialContext();
        UserTransaction ut =
            (UserTransaction)ctx.lookup("java:comp/UserTransaction");
        ut.begin();
        com.jpa.sample.Employee employee =
        em.find(com.jpa.sample.Employee.class, this.empNo);
        employee.setBand(this.band);
        employee.setDeptName(this.deptName);
        employee.setEmpName(this.empName);
        employee.setEmpNo(this.empNo);
        employee.setSalary(this.salary);
        employee.setSex(this.sex);
        ut.commit();
    }catch (Exception e){
        throw new IllegalArgumentException
        ("Message : sample.jsf.Employee : Exception :"+e);
public ArrayList<SelectItem> getDeptOptions() {
    return deptOptions;
}

public void setDeptOptions(ArrayList<SelectItem> deptOptions) {
    this.deptOptions = deptOptions;
}

public void retrieveEmployee(ActionEvent event) {
    com.jpa.sample.Employee employeeDup =
    em.find(com.jpa.sample.Employee.class, this.empNo);
    if(employeeDup == null)
        throw new IllegalArgumentException
        ("Message : sample.jsf.Employee : Employee does not exit ("+this.empNo+")");
    this.setBand(employeeDup.getBand());
    this.setDeptName(employeeDup.getDeptName());
    this.setEmpName(employeeDup.getEmpName());
    this.setEmpNo(employeeDup.getEmpNo());
    this.setSalary(employeeDup.getSalary());
    this.setSex(employeeDup.getSex());
}

Defining UI components
1. Right click on the **EmployeeWEB** project and create **index.jsp** as follows.

   ```html
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
   <html>
   <body>
   <a href="/EmployeeWEB/addEmployee.jsf">
   <font size=4 color='blue'>
   Add employee details
   </font></a> 
   <br/>
   <a href="/EmployeeWEB/RetrieveEmployee.jsf">
   <font size=4 color='blue'>
   Edit employee details
   </font></a> 
   </body>
   </html>
   ```

   The contents of the **index.jsp** are as follows.

   **index.jsp**

   ```html
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
   <html>
   <body>
   <a href="/EmployeeWEB/addEmployee.jsf">
   <font size=4 color='blue'>
   Add employee details
   </font></a> 
   <br/>
   <a href="/EmployeeWEB/RetrieveEmployee.jsf">
   <font size=4 color='blue'>
   Edit employee details
   </font></a> 
   </body>
   </html>
   ```

2. Similarly, create **addEmployee.jsp** and the contents are as follows.

   **addEmployee.jsp**

   ```jsp
   <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
   <%@ taglib uri="WEB-INF/tld/myfaces_html.tld" prefix="h" %>
   <%@ taglib uri="WEB-INF/tld/myfaces_core.tld" prefix="f" %>
   ```
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + "" + request.getServerPort() + path + ";";

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<base href="" + basePath + ">
<script type="text/javascript"></script>
<title>Add employee details</title>
</head>
<body>
<f:view>
<h:form>
<h:inputHidden id="id" value="#{Employee.empNo}"/>
<h:panelGrid columns="2" border="0">
<h:outputText value="Employee No : ">
<h:inputText id="empNo" value="#{Employee.empNo}" required="true">
<f:validateLongRange maximum="100000" minimum="1"/>
</h:inputText>
<h:outputText value="Employee Name : ">
<h:inputText id="empName" value="#{Employee.empName}" required="true">
<f:validateLength maximum="100" minimum="2"/>
</h:inputText>
<h:outputText value="Department Name : ">
<h:selectOneMenu id="deptName" value="#{Employee.deptName}" required="true">
<f:selectItems value="#{Employee.deptOptions}"/>
</h:selectOneMenu>
<h:outputText value="Sex : ">
<h:selectOneRadio id="sex" value="#{Employee.sex}" required="true">
<f:selectItem id="male" itemLabel="Male" itemValue="male" />
<f:selectItem id="female" itemLabel="Female" itemValue="female"/>
</h:selectOneRadio>
<h:outputText value="Salary : ">
<h:inputText id="salary" value="#{Employee.salary}" required="true">
<f:validateDoubleRange minimum="1000.00" maximum="10000000.00"/>
</h:inputText>
<h:outputText value="Employee Band : ">
<h:selectOneListbox id="band" value="#{Employee.band}" size="3" required="true">
<f:selectItem id="bandA" itemLabel="Band A" itemValue="A" />
<f:selectItem id="bandB" itemLabel="Band B" itemValue="B" />
<f:selectItem id="bandC" itemLabel="Band C" itemValue="C" />
<f:selectItem id="bandD" itemLabel="Band D" itemValue="D"/>
</body>
</f:view>
3. Similarly, create editEmployee.jsp and the contents are as follows.

```html
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="/WEB-INF/tld/myfaces_html.tld" prefix="h" %>
<%@ taglib uri="/WEB-INF/tld/myfaces_core.tld" prefix="f" %>

String path = request.getContextPath();
String basePath = request.getScheme()+":"+request.getServerName()+":"+request.getServerPort()+path+"/";

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<base href=""/>
<title>Edit employee details</title>
</head>
<body>

<f:view>
<h:form>

<h:inputHidden id="id" value="#{Employee.empNo}"/>
<h:panelGrid columns="2" border="0">
<h:outputText value="Employee No :"/>
<h:inputText id="empNo" disabled="true" value="#{Employee.empNo}" required="true">
<f:validateLongRange maximum="100000" minimum="1"/>
</h:inputText>
<h:outputText value="Employee Name :"/>
</h:panelGrid>
<h:messages id="errors" style="color:red;font-weight:bold" layout="table"/>
<h:commandButton value="Save" action="saveEmployee" actionListener="#{Employee.addEmployee}"/>

</h:form>
</f:view>
</body>
</html>
```
4. Similarly, create RetrieveEmployee.jsp and the contents are as follows.

RetrieveEmployee.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>
<%@ taglib uri="/WEB-INF/tld/myfaces_html.tld" prefix="h" %>
<%@ taglib uri="/WEB-INF/tld/myfaces_core.tld" prefix="f" %>

String path = request.getContextPath();
```
5. Open the `geronimo-web.xml` and replace the existing contents with the following contents. We explain the contents later when creating the datasource.

```
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"
         xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"
         xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
         xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
    <sys:environment>
        <sys:moduleId>
            <sys:groupId>EmployeeJSF</sys:groupId>
            <sys:artifactId>WEB</sys:artifactId>
            <sys:version>1.0</sys:version>
            <sys:type>car</sys:type>
        </sys:moduleId>
        <sys:dependencies>
            <sys:dependency>
                <sys:groupId>console.dbpool</sys:groupId>
                <sys:artifactId>EmployeeDS</sys:artifactId>
            </sys:dependency>
        </sys:dependencies>
    </sys:environment>
</web-app>
```
Configuring JSF managed beans and associating managed bean actions to events

1. Click on the faces-config.xml to open Faces Configuration Editor.

   ![Faces Configuration Editor](image)

   The Faces Configuration Editor helps you to complete the JSF web application development process by allowing you to edit configuration. Use the pages in this editor to define and edit page navigations, managed beans, component, converters, renderkit, and other element configurations.

   - **Start**
     - Start working with the editor by selecting PageFlow and defining page navigations.

   - **Tutorial**
     - Launch the cheat sheet for guidance in working with the editor.

2. Click on the **ManagedBean** tab below and Click on the **Add** button. This will open New Managed Bean Wizard.

   ![New Managed Bean Wizard](image)

   - **Java Class Selection**
     - Search for an existing class or generate a new one:

       - Using an existing Java class
         - Qualified class name: [Browse...]
         - (This option will use an existing java class as managed bean's type.)

       - Create a new Java class
         - (This option will create a new java class in the next wizard page.)
3. Click on the **Browse** button and search for `sample.jsf.Employee` class and click on the **OK** button.
4. Click on the **Next** button on the *New Managed Bean Wizard*. On the next screen, select *Scope* as *request* and click on the **Next** button.

![New Managed Bean Wizard](image)

**Managed Bean Configuration**

Set managed bean's name and scope, and create or update Managed Bean's properties, their types and default values.

**General**
- **Name:** employee1
- **Scope:** request
- **Description:**

5. Click on the **Next** button and finally, click on the **Finish** button. This will bring up the following screen shot as below.

![Managed Bean](image)

**Managed Bean**

- **Managed Bean Elements:** The following managed beans are defined.
  - **request**
    - Employee
    - application
  - **Employee**
- **Initialization**
  - You can initialize the managed bean's properties or itself if it is a subclass of java.util.Map or java.util.List

**Managed Bean Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Click on the Navigation Rule tab and move the mouse over Palette to bring it to the front.

7. Click on the Page and move the mouse over main area and click once. This will bring up Select JSP File wizard. Select the index.jsp and click on the OK button.
8. Similarly, add other JSPs also to the main area as follows.

9. Bring the **Palette** to the front and click on the **Link**.
   - Place the mouse on `addEmployee.jsp` on the main area and click once and move it to `index.jsp` and click once.
   - Similarly, click the mouse over `RetrieveEmployee.jsp` and move the mouse to `index.jsp` and click once.
   - Click the mouse over `RetrieveEmployee.jsp` and move the mouse over `editEmployee.jsp` and click once.
   - Click the mouse over `editEmployee.jsp` and move the mouse over `RetrieveEmployee.jsp` and click once.

The above actions create navigation rules and look like the screen shot below.
10. Press the escape button once and click on the link between addEmployee.jsp and index.jsp. On the Properties window at the bottom, go to the From Outcome textbox and provide the value as saveEmployee.

11. Similarly, Click on the link between RetrieveEmployee.jsp and index.jsp, and on the Properties window at the bottom, go to the From Outcome textbox and provide the value as cancel.

12. Similarly, Click on the link between RetrieveEmployee.jsp and editEmployee.jsp, and on the Properties window at the bottom, go to the From Outcome textbox and provide the value as retrieveEmployee.

13. Similarly, Click on the link between editEmployee.jsp and RetrieveEmployee.jsp, and on the Properties window at the bottom, go to the From Outcome textbox and provide the value as saveEmployee.

14. Click on Source to view the XML content generated out of these navigations rules and configuration. The contents should be as follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
 xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
 version="1.2">
 <navigation-rule>
<description>Add Employee</description>
<from-view-id>/addEmployee.jsp</from-view-id>
<navigation-case>
<from-outcome>saveEmployee</from-outcome>
<to-view-id>/index.jsp</to-view-id>
</navigation-case>
</navigation-rule>

```

<navigation-rule>
    <description>Retrieve Employee</description>
    <from-view-id>/RetrieveEmployee.jsp</from-view-id>
    <navigation-case>
        <from-outcome>retrieveEmployee</from-outcome>
        <to-view-id>/editEmployee.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

<navigation-rule>
    <description>Retrieve Employee</description>
    <from-view-id>/RetrieveEmployee.jsp</from-view-id>
    <navigation-case>
        <from-outcome>cancel</from-outcome>
        <to-view-id>/index.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

<navigation-rule>
    <description>Edit Employee</description>
    <from-view-id>/editEmployee.jsp</from-view-id>
    <navigation-case>
        <from-outcome>saveEmployee</from-outcome>
        <to-view-id>/RetrieveEmployee.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

<managed-bean>
    <description>
        Employee Bean
    </description>
    <managed-bean-name>Employee</managed-bean-name>
    <managed-bean-class>sample.jsf.Employee</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

</faces-config>

This finishes the WEB application creation and JSF configuration. Export the EmployeeWEB to a EmployeeWEB.war file.

Setting up the database tables and the Datasource
1. Start the geronimo server and click on the **Console Navigation -> Embedded DB -> DB manager**. Enter EmployeeDB in the **Create DB** textbox as shown in the below screen shot and click on the **Create** button.

![DB Viewer](image)

**Run SQL**

Create DB: EmployeeDB
Delete DB: AccountDB
Use DB: AccountDB

SQL Command/s:

```
create table EMPLOYEE (EMPNO integer, ENAME varchar(255), DEPTNO varchar(255), SEX varchar(10), SALARY decimal(20,2), BAND varchar(10));
```

2. Create the **Employee** table by entering the below SQL statement and clicking on the **Run SQL** button as shown in the below screen shot.

![Run SQL](image)

**Run SQL**

Create DB:
Delete DB: AccountDB
Use DB: EmployeeDB

SQL Command/s:

```
create table EMPLOYEE (EMPNO integer, ENAME varchar(255), DEPTNO varchar(255), SEX varchar(10), SALARY decimal(20,2), BAND varchar(10));
```
3. Click on the Console Navigation -> Services -> Database Pools, and click on the Using the Geronimo database pool wizard link to create a new datasource.

Database Pools

This page lists all the available database pools.
For each pool listed, you can click the usage link to see examples of how to use the pool.

<table>
<thead>
<tr>
<th>Name</th>
<th>Deployed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>CurrencyDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>MonitoringClientDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>NoTxDataSource</td>
<td>Server-wide</td>
</tr>
<tr>
<td>PhaniDBPool</td>
<td>Server-wide</td>
</tr>
<tr>
<td>ProductDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>ProductDS-nonJTA</td>
<td>Server-wide</td>
</tr>
<tr>
<td>SystemDataSource</td>
<td>Server-wide</td>
</tr>
<tr>
<td>jdbc/ActiveDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>jdbc/ArchiveDS</td>
<td>Server-wide</td>
</tr>
<tr>
<td>jdbc/juddiDB</td>
<td>org.apache.geronimo.configs/udci-tomcat/2.1/car</td>
</tr>
</tbody>
</table>

Create a new database pool:
- Using the Geronimo database pool wizard
- Import from JBoss 4
- Import from WebLogic 8.1

4. On the next screen, enter EmployeeDS in the Name of Database Pool textbox and select Derby embedded in the Database Type combo box. Click on the Next button.

Database Pools

Create Database Pool -- Step 1: Select Name and Database

Name of Database Pool: EmployeeDS
A name that is different than the name for any other database pools in the system please.
Database Type: Derby embedded
The type of database the pool will connect to.
Next
Cancel
5. On the next screen, select the configuration in the Driver JAR select box and enter EmployeeDB in the Database Name textbox, and click on the Deploy button at the bottom.

This page edits a new or existing database pool.

![Database Pool Configuration](image1)

5.6. This will deploy the EmployeeDS database pool. The moduleId of the database pool is `console.dbpool/EmployeeDS/1.0/rar` which is declared as a dependency in the `geronimo-web.xml`. This is because, JPA uses this database source to persist entities.

### Deploying the (war) application

Deploy the EmployeeWEB.war file using the command prompt as follows.

```
C:\Geronimo-2.1\bin>deploy.bat --user system --password manager deploy EmployeeWEB.war
Using GERONIMO_BASE:  C:\Geronimo-2.1
Using GERONIMO_HOME:  C:\Geronimo-2.1
Using GERONIMO_TMPDIR: var\temp
Using JRE_HOME:        C:\SDK-May-31-2007\jre
Deployed EmployeeJSF/WEB/1.0/car @ /EmployeeWEB
```

### Running the application

1. Open a browser window and hit the URL `http://localhost:8080/EmployeeWEB/`. This will bring up the below screen.

![Application Running](image2)
2. Click on the **Add employee details** link to get the below screen to add employees into the database. Just click on the **Save** button without entering any values. The JSF complains with validation errors as below.

3. Enter the values as given in the screen below and click on the **Save** button. This will take back to the main page.

4. Click on the **Edit employee details** link to get the below screen. Enter **Employee No** value as 10 and click on the **Retrieve** button.
5. This will bring up the details of the employee whose employee number is 10. We can edit the values as required and click on the **Save** button.

![Employee details form]

<table>
<thead>
<tr>
<th>Employee No</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee Name</td>
<td>John Rodgers</td>
</tr>
<tr>
<td>Department Name</td>
<td>Marketing</td>
</tr>
<tr>
<td>Sex</td>
<td>Male</td>
</tr>
<tr>
<td>Salary</td>
<td>1000000.0</td>
</tr>
<tr>
<td>Employee Band</td>
<td>Band A</td>
</tr>
<tr>
<td></td>
<td>Band B</td>
</tr>
<tr>
<td></td>
<td>Band C</td>
</tr>
</tbody>
</table>

**Save**