

Accessing JDBC in Web applications

[← Accessing EJB in Web applications](#)

[↑ Developing Web applications](#)

[Accessing JMS in Web applications →](#)

This application will deal with populating an image on a database and later retrieving the same from the database. This tutorial will help you understand how to deal with data of type Blob. We will be using Derby database in this application. Please use images of smaller sizes to upload on to database. This is because there are some limitations associated with the derby database.

To run this tutorial, as a minimum you will be required to have installed the following prerequisite software:

1. Sun JDK 6.0+ (J2SE 1.6)
2. Eclipse IDE for Java EE Developers, which is platform specific
3. Apache Geronimo Eclipse Plugin 2.1.x
4. Apache Geronimo Server 2.1.x



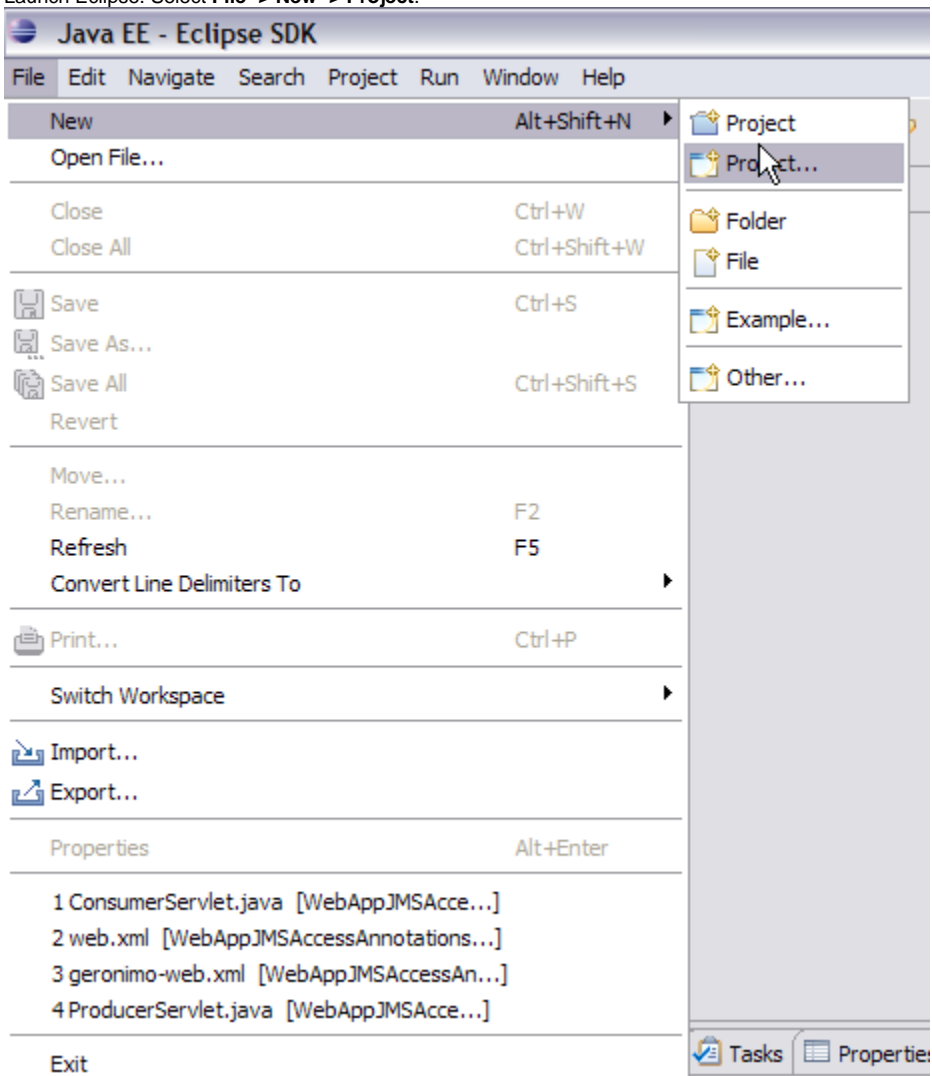
Geronimo version 2.1.x, Java 1.5 runtime, and Eclipse Ganymede are used in this tutorial but other versions can be used instead (e.g., Geronimo version 2.2, Java 1.6, Eclipse Europa)

Details on installing eclipse are provided in the [Development environment](#) section. This tutorial is organized in the following sections:

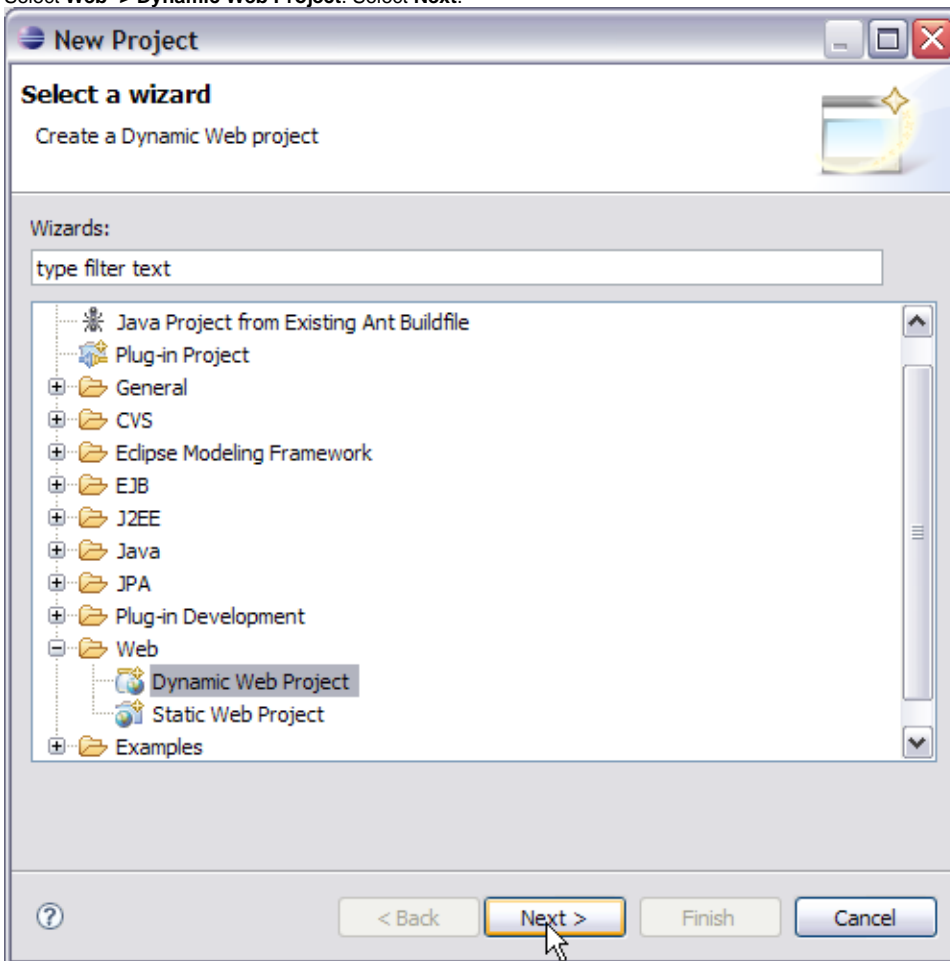
- [Creating a dynamic Web project](#)
- [Creating a database using the administrative console](#)
- [Creating a datasource using the administrative console](#)
- [Adding code for image upload to derby database](#)
- [Code to retrieve the image from derby database](#)
- [Modifying deployment plan](#)
- [Deploy and run](#)

Creating a dynamic Web project

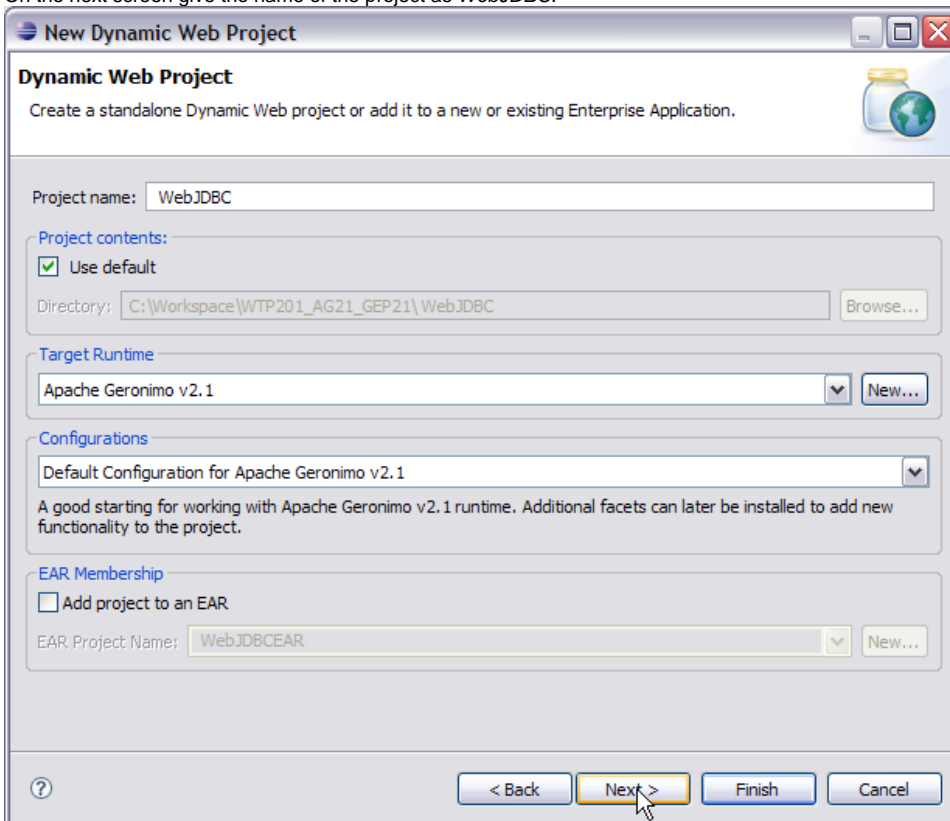
1. Launch Eclipse. Select **File -> New -> Project**.



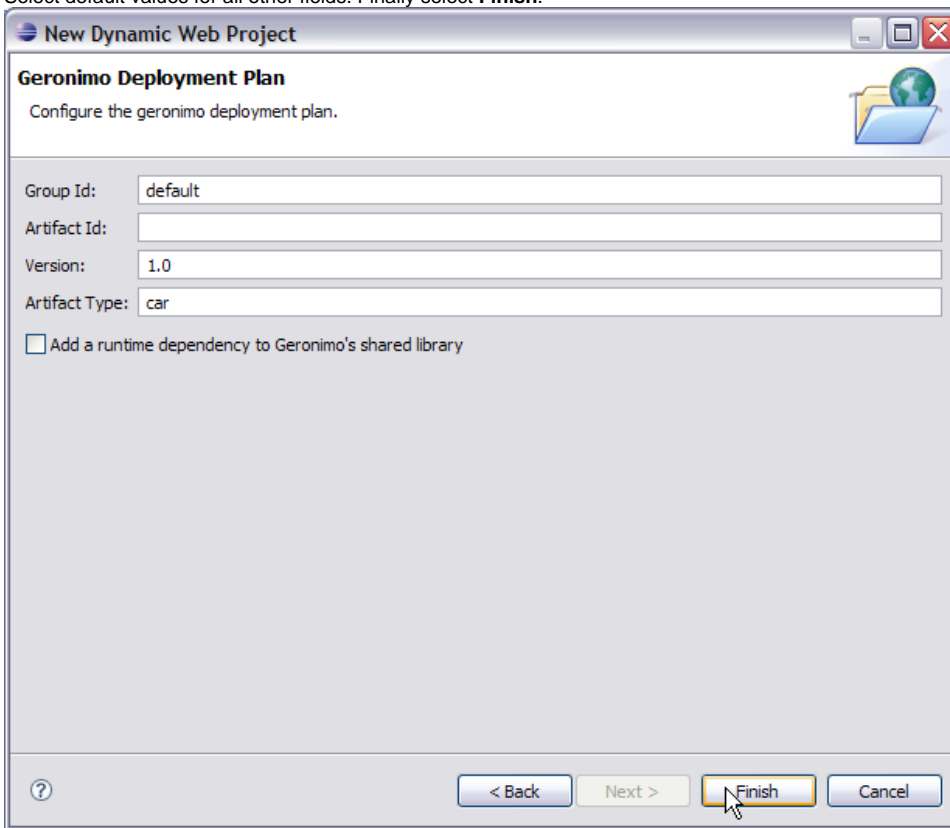
2. Select **Web** -> **Dynamic Web Project**. Select **Next**.



3. On the next screen give the name of the project as *WebJDBC*.



4. Select default values for all other fields. Finally select **Finish**.



New Dynamic Web Project

Geronimo Deployment Plan
Configure the geronimo deployment plan.

Group Id:

Artifact Id:

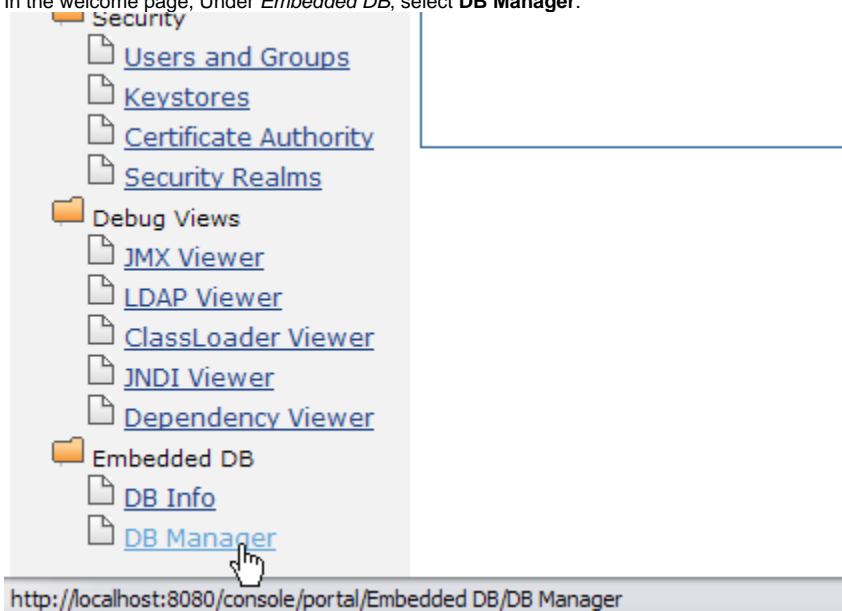
Version:

Artifact Type:

☐ Add a runtime dependency to Geronimo's shared library

Creating a database using the administrative console

1. Start the server and Launch the Administrative Console using the URL <http://localhost:8080/console>.
2. Enter default username and password.
3. In the welcome page, Under *Embedded DB*, select **DB Manager**.



4. On the next page create a database *userdbs* and select **Create**.

DB Viewer

Database List	
Databases	
ActiveMRCDB	Application
ArchiveMRCDB	Application
SystemDatabase	Application
test	Application
UddiDatabase	Application

Run SQL

Create DB:

Delete DB:

Use DB:

SQL Command/s:

5. Once done you can see the *userdbs* database listed in DB Viewer portlet under *Databases*. This confirms that the database has been successfully created.

DB Viewer

Database List	
Databases	View
ActiveMRCDB	Application
ArchiveMRCDB	Application
SystemDatabase	Application
test	Application
UddiDatabase	Application
userdbs	Application

Run SQL

6. As shown in the figure below, select **userdbs** from the dropdown box.

Run SQL

Create DB:

Delete DB:

Use DB:

ActiveMRCDB
ArchiveMRCDB
SystemDatabase
test
UddiDatabase
userdbs

7. Run the query as shown in the figure. This query will create table *PICTURES* with the columns *name* and *pic*.

Run SQL

Create DB:

Delete DB: ActiveMRCDB

Use DB: userdb

SQL Command/s:

```
create table pictures(name varchar(32) not null primary key, pic
blob(16M));
```

CreateTable.sql

```
create table pictures(name varchar(32) not null primary key, pic blob(16M));
```

Creating a datasource using the administrative console

1. Start the server and Launch the Administrative Console using the URL <http://localhost:8080/console>.
2. Enter default username and password.
3. Once in the welcome page. In console navigation, Under **Services**, select **Database Pools**.

4. On the next screen, Create a new database pool **Using the Geronimo database pool wizard**.

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 it.

Name	Deployed As
MonitoringClientDS	Server-wide
NoTxDatasource	Server-wide
SystemDatasource	Server-wide
jdbc/ActiveDS	Server-wide
jdbc/ArchiveDS	Server-wide
jdbc/testds	Server-wide
jdbc/juddiDB	org.apache.geronimo.configs/uddi-tomcat/2.1/car

Create a new database pool:

- ♦ [Using the Geronimo database pool wizard](#)
- ♦ [Import from JBoss 4](#)
- ♦ [Import from WebLogic 8.1](#)

5. On the next screen give the name as suggested in the figure *jdbc/userds*. This will initiate the process to create a Derby Embedded XA datasource.

Database Pools

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

Name of Database Pool:

A name that is different than the name for any other database (the name please).

Database Type:

The type of database the pool will connect to.

[Cancel](#)

6. Select the **Driver Jar** and give the database name as *userds* (remember this is the database we created in the previous step). All other fields can be set to default.

Database Pools

+

-

?

This page edits a new or existing database pool.

Pool Name: jdbc/userds

A name that is different than the name for any other database pools in the server (no spaces in the name please).

Pool Type: TranQL Embedded XA Resource Adapter for Apache Derby

A resource adaptor that provides access to an embedded Apache Derby database with XA support.

Basic Connection Properties

Driver JAR:

org.apache.geronimo.configs/system-database/2.1/car

The JAR(s) required to make a connection to the database. Use CTRL-click or SHIFT-click to select multiple jars.

The JAR(s) should already be installed under GERONIMO/repository/ (or [Download a Driver](#))

Database Name: userdbs

Name of the database to connect to.

Password:

7. Select **Deploy** to deploy the connector plan.

+

-

?

This config-property is currently ignored by Derby.

Create Database: true

Flag indicating that the database should be created if it does not exist. This is a

Connection Pool Parameters

Pool Min Size: 0

The minimum number of connections in the pool. The default is 0.

Pool Max Size: 10

The maximum number of connections in the pool. The default is 10.

Blocking Timeout: (in milliseconds)

The length of time a caller will wait for a connection. The default is 5000.

Idle Timeout: (in minutes)

How long a connection can be idle before being closed. The default is 15.

[Deploy](#) [Show Plan](#)

[Cancel](#)

8. Once done you can see the Database Pool *jdbc/userds* listed in the available database pools.

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

Name	Deployed As
MonitoringClientDS	Server-wide
NoTxDatasource	Server-wide
SystemDatasource	Server-wide
jdbc/ActiveDS	Server-wide
jdbc/ArchiveDS	Server-wide
jdbc/testds	Server-wide
jdbc/userds	Server-wide
jdbc/juddiDB	org.apache.geronimo.configs/uddi-tomcat/2.1/car

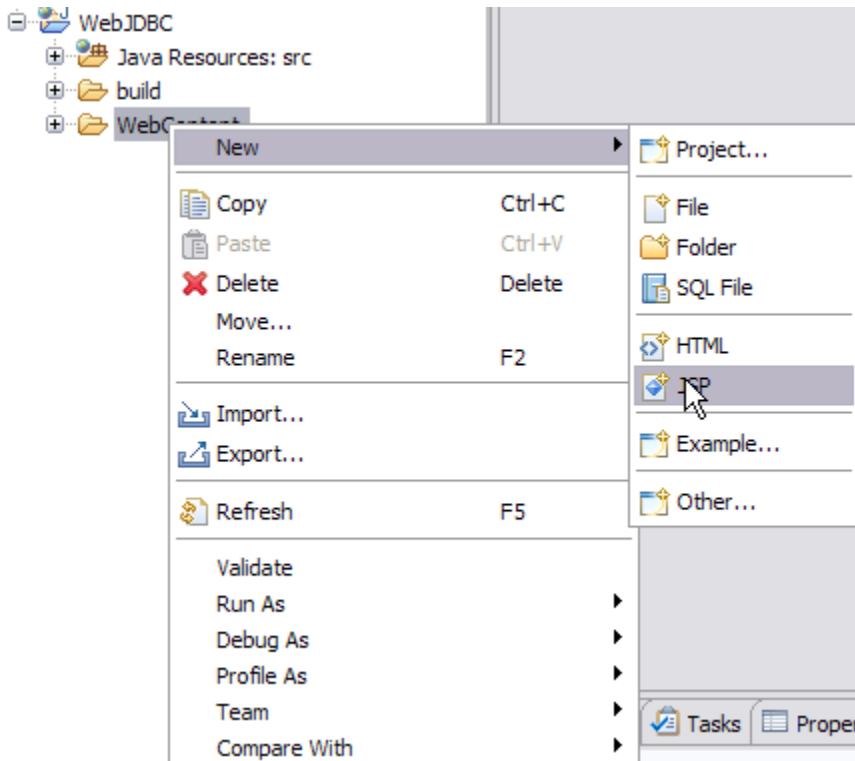
Create a new database pool:

- ◆ [Using the Geronimo database pool wizard](#)
- ◆ [Import from JBoss 4](#)
- ◆ [Import from WebLogic 8.1](#)

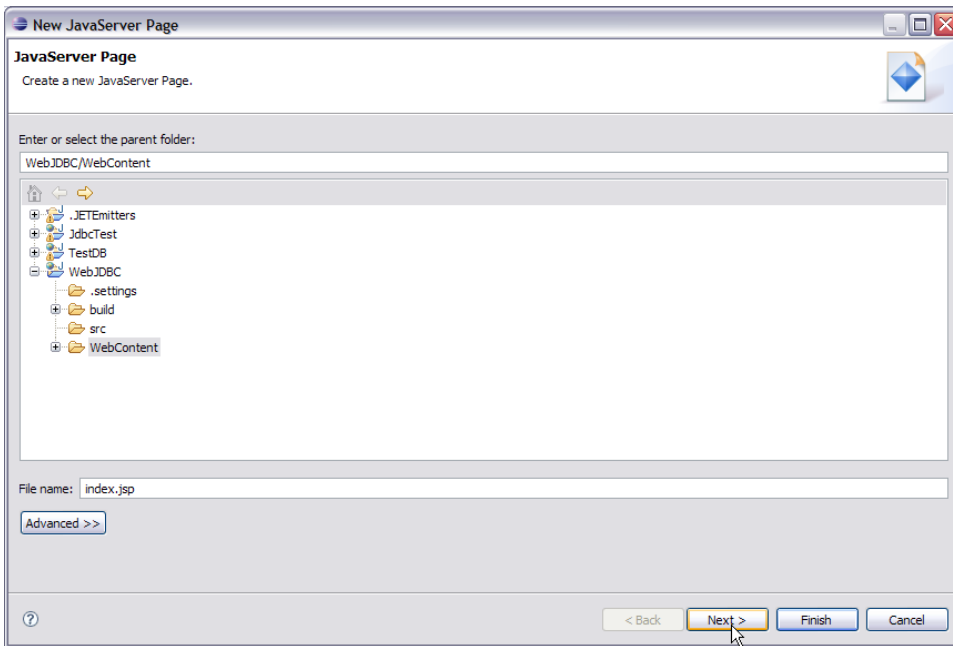


Adding code for image upload to derby database

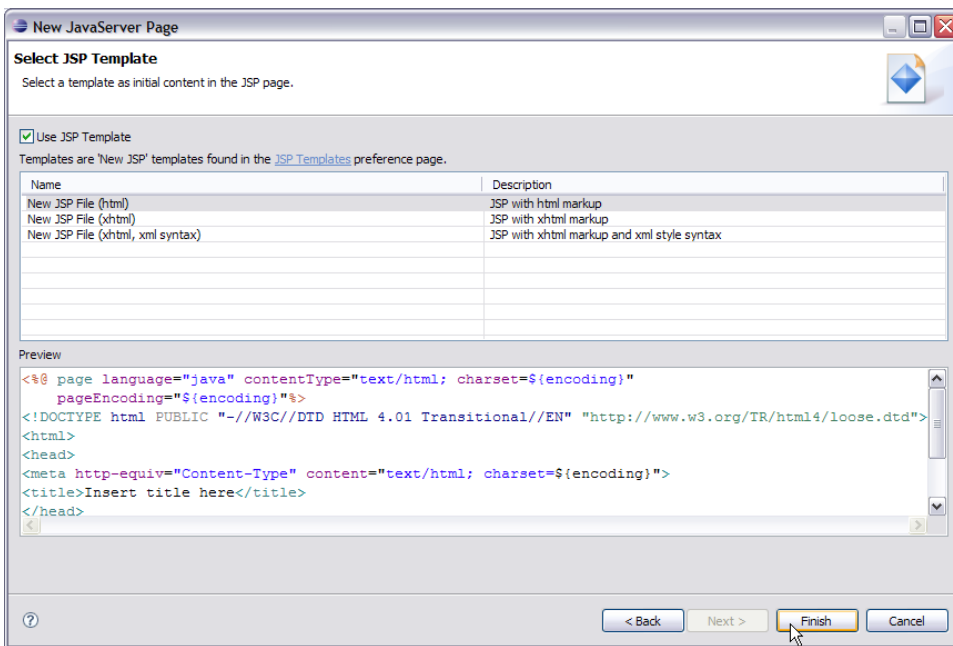
1. Right click on **WebContent** and select **New -> JSP**.



2. Name the JSP as *index.jsp* and select **Next**.



3. Select **Finish**. This will create a template for `index.jsp`.



4. Add the following code to `index.jsp`.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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">
<title>Image Upload</title>
</head>
<body>
<h2>Select a Image and Upload it</h2>
```

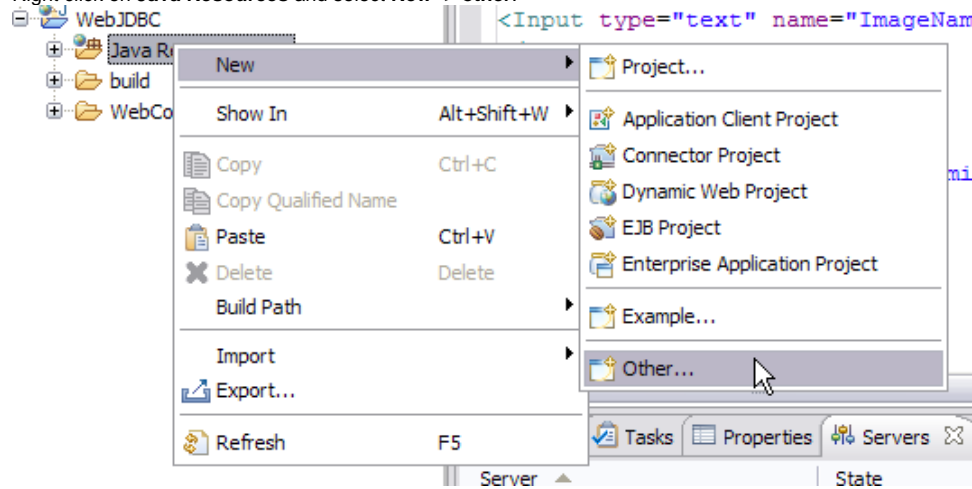
```

<form action="/WebJDBC/ImageUpload">
<table>
<tr>
<td>
Location of the Image(full path)
</td>
<td>
<input type="text" name="ImageLoc">
</td>
</tr>
<tr>
<td>
Name of the Image(Unique Name)
</td>
<td>
<input type="text" name="ImageName">
</td>
</tr>
<tr>
<td>
<input type="submit" value="submit">
</td>
</tr>
</table>
</form>
</body>
</html>

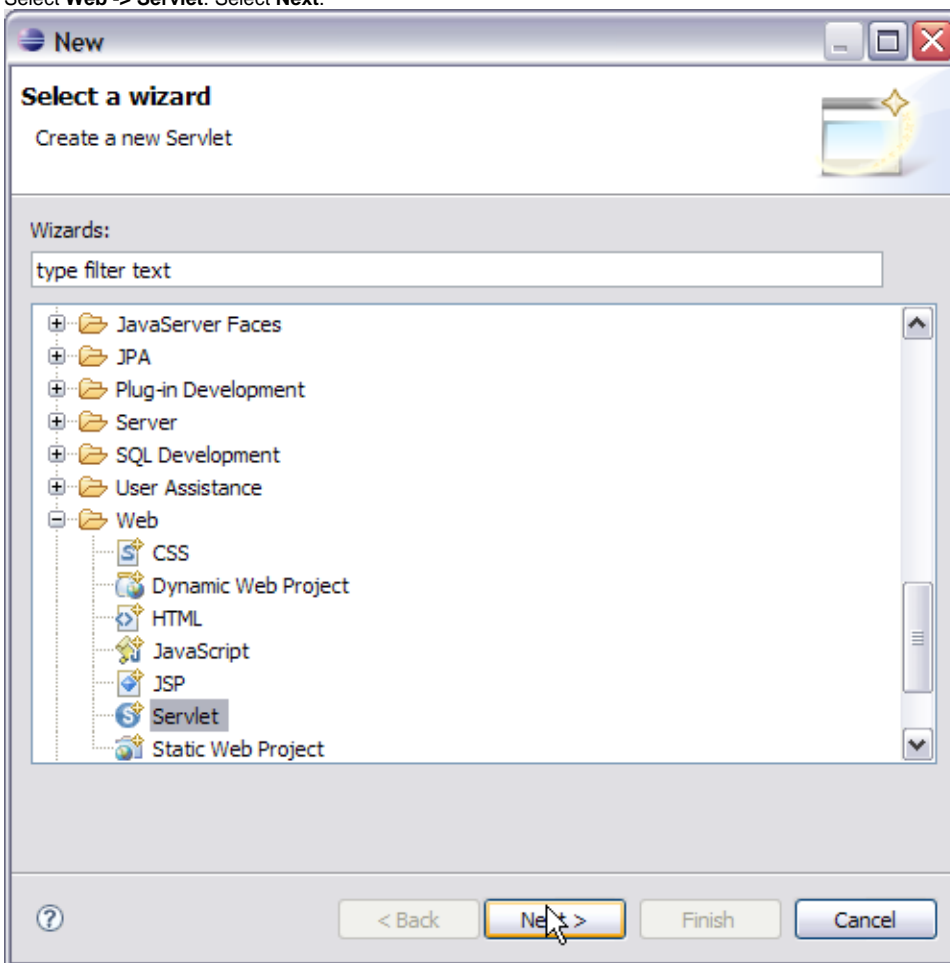
```

The `<form action="/WebJDBC/ImageUpload">` suggests that once the form is submitted the request will be passed to **ImageUpload**. The next step is to add the servlet **ImageUpload** to process the request sent by the JSP client.

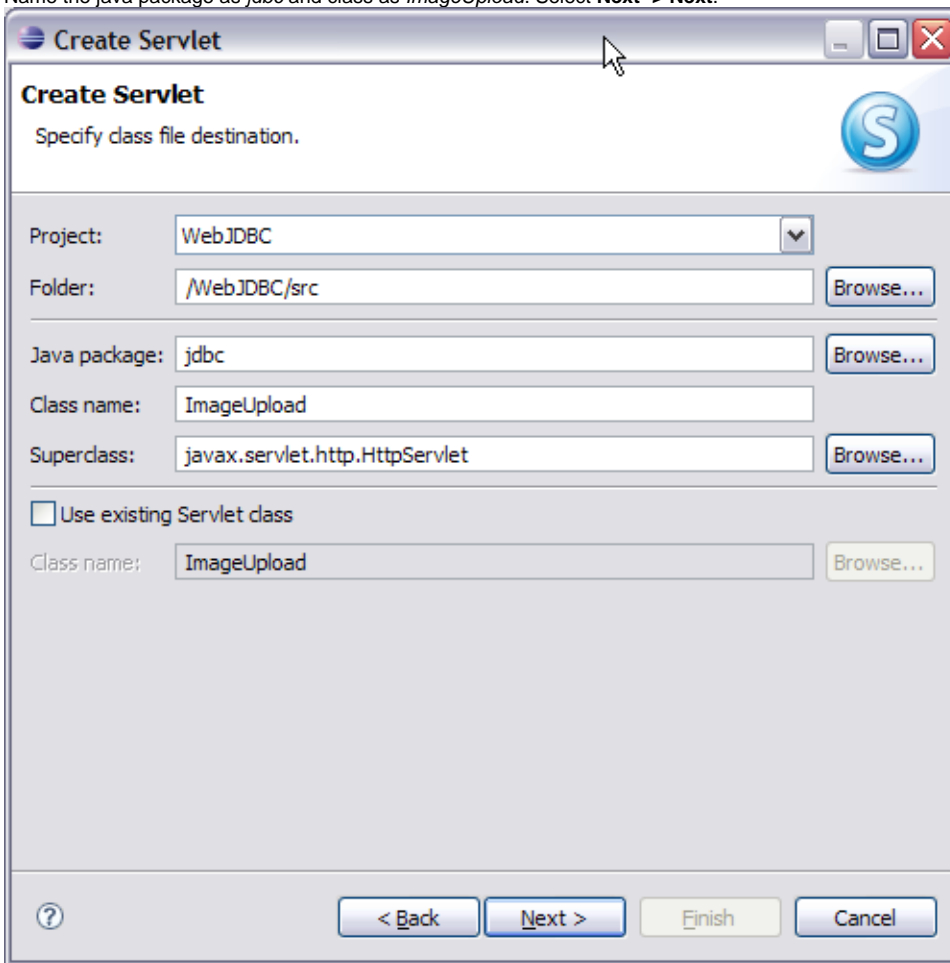
5. Right click on **Java Resources** and select **New -> other**.



6. Select **Web** -> **Servlet**. Select **Next**.



7. Name the java package as *jdbc* and class as *ImageUpload*. Select **Next** -> **Next**.



The image shows a 'Create Servlet' dialog box from an IDE. The title bar says 'Create Servlet'. Below the title bar, there's a sub-header 'Create Servlet' and a description 'Specify class file destination.' with a blue 'S' icon. The dialog contains several input fields and buttons. The 'Project' field is a dropdown menu showing 'WebJDBC'. The 'Folder' field is a text box showing '/WebJDBC/src' with a 'Browse...' button to its right. The 'Java package' field is a text box showing 'jdbc' with a 'Browse...' button to its right. The 'Class name' field is a text box showing 'ImageUpload'. The 'Superclass' field is a text box showing 'javax.servlet.http.HttpServlet' with a 'Browse...' button to its right. Below these fields, there is a checkbox labeled 'Use existing Servlet class' which is currently unchecked. Below the checkbox, there is a 'Class name' field showing 'ImageUpload' and a 'Browse...' button. At the bottom of the dialog, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. A mouse cursor is pointing at the top center of the dialog box.

Create Servlet

Specify class file destination.

Project: WebJDBC

Folder: /WebJDBC/src [Browse...](#)

Java package: jdbc [Browse...](#)

Class name: ImageUpload

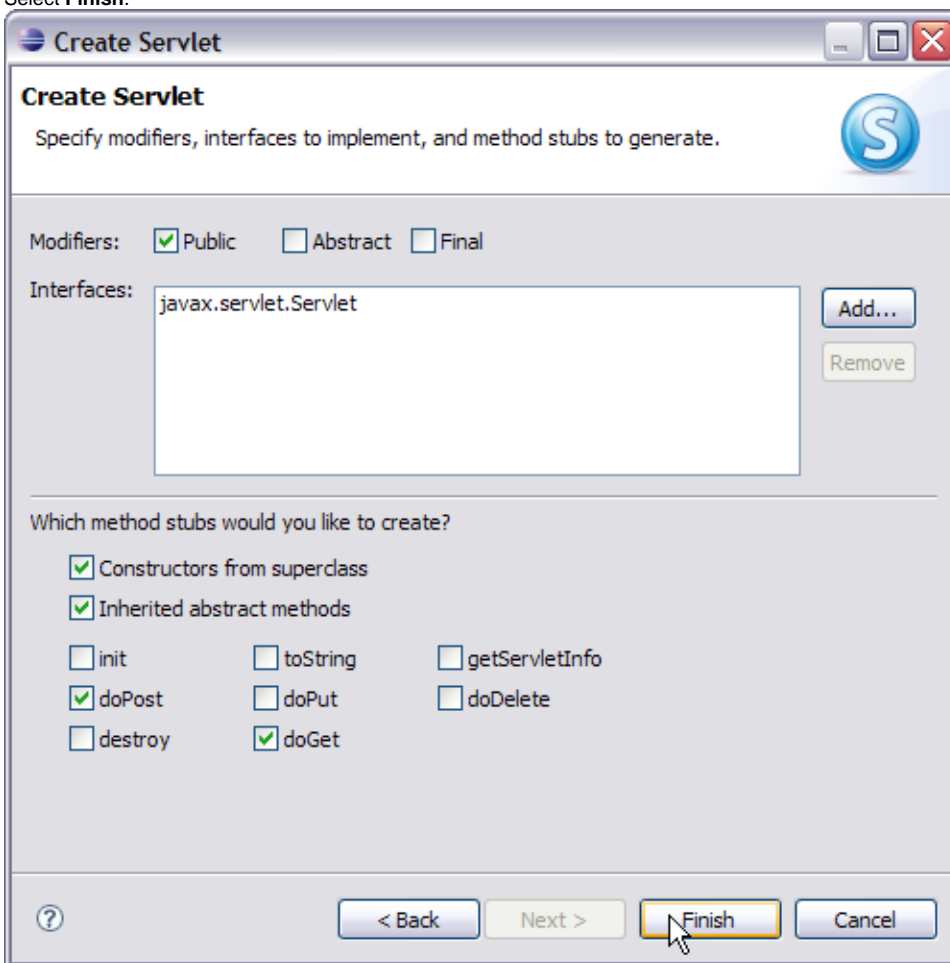
Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use existing Servlet class

Class name: ImageUpload [Browse...](#)

[?< Back](#) [Next >](#) [Finish](#) [Cancel](#)

8. Select **Finish**.



The image shows the 'Create Servlet' dialog box in an IDE. The title bar says 'Create Servlet'. Below the title bar, there's a subtitle 'Specify modifiers, interfaces to implement, and method stubs to generate.' and a blue 'S' icon. The 'Modifiers' section has three checkboxes: 'Public' (checked), 'Abstract' (unchecked), and 'Final' (unchecked). The 'Interfaces' section has a text box containing 'javax.servlet.Servlet' and two buttons: 'Add...' and 'Remove'. Below this, the question 'Which method stubs would you like to create?' is followed by a grid of checkboxes. The checked options are: 'Constructors from superclass', 'Inherited abstract methods', 'doPost', 'doGet', and 'destroy'. The unchecked options are: 'init', 'toString', 'getServletInfo', 'doPut', and 'doDelete'. At the bottom, there are four buttons: '?', '< Back', 'Next >', and 'Finish' (which is highlighted with a mouse cursor). There is also a 'Cancel' button.

9. Add the following code to `ImageUpload.java`.

ImageUpload.java

```
package jdbc;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class ImageUpload extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet {
    @Resource(name="jdbc/users")
    private DataSource ds;
    static final long serialVersionUID = 1L;

    public ImageUpload() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        doProcess(request, response);
    }
}
```

```

        protected void doProcess(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            Connection dbconnect = null;
            PreparedStatement stmtnt = null;
            String pic=request.getParameter("ImageLoc");
            String name=request.getParameter("ImageName");
            try{
                File f= new File(pic);
                FileInputStream fis=new FileInputStream(f);
                dbconnect= ds.getConnection();
                stmtnt = dbconnect.prepareStatement("INSERT INTO PICTURES ( " + "NAME," + "PIC )"
+ " VALUES(?,?)");
                stmtnt.setString(1, name);
                stmtnt.setBinaryStream(2, fis, (int)f.length());
                stmtnt.execute();
                PrintWriter out= response.getWriter();
                out.println("Congratulations your image has been successfully
uploaded");
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
            finally
            {
                try{
                    dbconnect.close();
                    stmtnt.close();
                }catch(SQLException e){
                    e.printStackTrace();
                }
            }
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            doProcess(request, response);
        }
    }
}

```

Once a request is submitted from `index.jsp` for an image upload with the name and exact location of image the request is sent to the **ImageUpload** servlet.

- `String pic=request.getParameter("ImageLoc");`:- Retrieves the location of the image
- `String name=request.getParameter("ImageName");`:- Retrieves the name of the image

Once we have the image available a new `File` object is created from the image. Thereafter the image file is read as a binary stream. **PreparedStatement** is used to insert the image onto database. This completes the code for **ImageUpload**.

Code to retrieve the image from derby database

1. Create a **JSP** page with the name `ImageDownload.jsp`.
2. Add the following code to `ImageDownload.jsp`:

ImageDownload.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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">
<title>Download Image</title>
</head>
<body>
<h2>Name a Image to download</h2>

```

```

<form action="/WebJDBC/ImageDownload">
<table>
<tr>
<td>
Name of the Image
</td>
<td>
<input type="text" name="ImageName">
</td>
</tr>
<tr>
<td>
<input type="submit" value="submit">
</td>
</tr>
</table>
</form>
</body>
</html>

```

As can be seen from **<form action="/WebJDBC/ImageDownload">** this JSP requests the **ImageDownload** servlet when the form is submitted.

3. Create a new servlet `ImageDownload.java` and add the following code:

ImageDownload.java

```

package jdbc;

import java.io.IOException;
import java.io.OutputStream;
import java.sql.Blob;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class ImageDownload extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet {
    @Resource(name = "jdbc/userds")
    private DataSource ds;
    static final long serialVersionUID = 1L;

    public ImageDownload() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doProcess(request, response);
    }

    protected void doProcess(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        Connection dbconnect = null;
        ResultSet rs = null;
        PreparedStatement stmt = null;
        try {
            dbconnect = ds.getConnection();
            String s=request.getParameter("ImageName");
            stmt = dbconnect.prepareStatement("SELECT PIC FROM PICTURES WHERE NAME=?");
            stmt.setString(1, s);
            rs = stmt.executeQuery();
            if (rs.next()) {
                // Get as a BLOB
                Blob aBlob = rs.getBlob(1);

```



```

        byte[] b = new byte[4096];
        java.io.InputStream ip = aBlob.getBinaryStream();
        OutputStream out = null;
        int c = 0;
        out = response.getOutputStream();
        response.setContentType("image/jpeg");
        while (c != -1) {
            c = ip.read(b);
            if (c > 0) {
                out.write(b, 0, c);
                out.flush();
            }
        }
        ip.close();
    }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try{
            dbconnect.close();
            stmt.close();
            rs.close();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    doProcess(request, response);
}
}

```

The **ImageDownload** servlet works in a similar way once the request is sent from `ImageDownload.jsp` to it. The servlet uses

- `String s=request.getParameter("ImageName");`- To retrieve the name of the image to be downloaded
- `rs = stmt.executeQuery();`- To execute the PreparedStatement
- `Blob aBlob = rs.getBlob(1);`- Data is stored as a Blob datatype
- `java.io.InputStream ip = aBlob.getBinaryStream();`- Blob data is retrieved as a binary stream
- `response.setContentType("image/jpeg");`- sets the output content type as a Image datatype
- `c = ip.read(b);, out.write(b);, out.flush();`- Data is read as a byte buffer and written on the web page. After each write the buffet is flushed.

This completes the code for Image download.

Modifying deployment plan

The next step is to modify the `geronimo-web.xml` deployment plan. We need to add the dependency element for the **JDBC** resource. Also we need to add a resource reference element for the *users* datasource.

geronimo-web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"
    xmlns:nam="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>default</sys:groupId>
            <sys:artifactId>WebJDBC</sys:artifactId>
            <sys:version>1.0</sys:version>
            <sys:type>car</sys:type>
        </sys:moduleId>
    </sys:environment>
    <sys:dependencies>
        <sys:dependency>
            <sys:groupId>console.dbpool</sys:groupId>

```

```

        <sys:artifactId>jdbc_usersds</sys:artifactId>
    </sys:dependency>
</sys:dependencies>
</sys:environment>
<context-root>/WebJDBC</context-root>
<nam:resource-ref>
    <nam:ref-name>jdbc/usersds</nam:ref-name>
    <nam:pattern>
        <nam:groupId>console.dbpool</nam:groupId>
        <nam:artifactId>jdbc_usersds</nam:artifactId>
        <nam:name>jdbc/usersds</nam:name>
    </nam:pattern>
</nam:resource-ref>
</web-app>

```

- <sys:dependency>- The dependency element is to suggest the dependency of the application on the database pool. In our case we have **usersds** as a dependency module.
- <nam:resource-ref>- This element is used to map the JDBC connection pool with a user defined name. In our case we have mapped **jdbc** /**usersds** with **jdbc_usersds**.

Deploy and run

1. Start the server within Eclipse.
2. Right click on **WebJDBC** project and select **Run as -> Run on server**.
3. Once done the application will be deployed on the server.
4. Launch the application with the following link <http://localhost:8080/WebJDBC/>

Select a Image and Upload it

Location of the Image(full path)

Name of the Image(Unique Name)

5. Fill up the form with the a image location and name. The same name will be used while populating the database with the image data. Select **Submit** it once done.

Select a Image and Upload it

Location of the Image(full path)

Name of the Image(Unique Name)

6. If your image is successfully inserted into the database you will get a Congratulation message.

Congratulations your image has been successfully uploaded

7. To retrieve an image from the database launch the following portlet <http://localhost:8080/WebJDBC/ImageDownload.jsp>. Select **submit** once done.

Name a Image to download

Name of the Image

8. This will display the image in the browser.