Servlet JSP Workshop
Lab 2

1 Lab setup

Make sure you have the following items installed

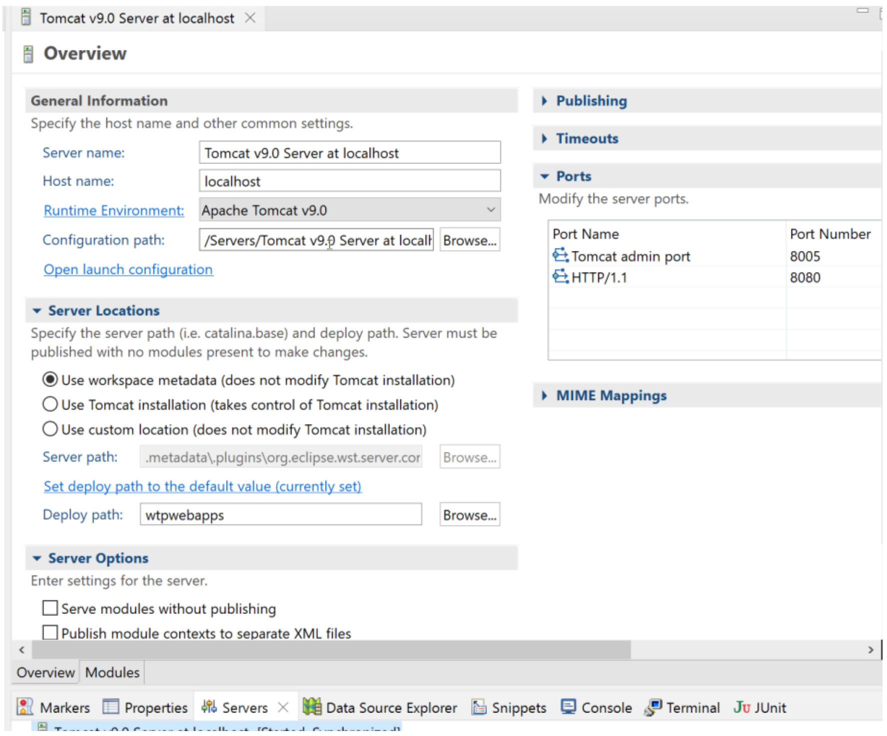
- Latest version of JDK 11 (note: labs are tested with JDK 11 but should work with higher versions with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Tomcat 9.0 installed as a Windows service
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder holds the source code and other related files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Creating a Dynamic Web Project

Make sure you have completed the installation of Tomcat and integration into Eclipse.

To check that the correct port numbers have been assigned to the integrated Tomcat server, double click on the Tomcat server entry in the Servers view tab below the main editor view.
Make sure that the Tomcat admin port and HTTP port are set to their typical default values of 8005 and 8080.



Tomcat v9.0 Server at localhost

Overview

General Information
Specify the host name and other common settings.

Server name: Tomcat v9.0 Server at localhost
Host name: localhost
Runtime Environment: Apache Tomcat v9.0
Configuration path: /Servers/Tomcat v9.0 Server at localh
Open launch configuration

Server Locations
Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.

[✓ Use workspace metadata (does not modify Tomcat installation)]
Use Tomcat installation (takes control of Tomcat installation)
Use custom location (does not modify Tomcat installation)
Server path: .metadata\.plugins\org.eclipse.wst.server.cor
Set deploy path to the default value (currently set)
Deploy path: wtpwebapps

Server Options
Enter settings for the server.

☐ Serve modules without publishing
☐ Publish module contexts to separate XML files

Publishing

Timeouts

Ports
Modify the server ports.

Port Name	Port Number
Tomcat admin port	8005
HTTP/1.1	8080

MIME Mappings

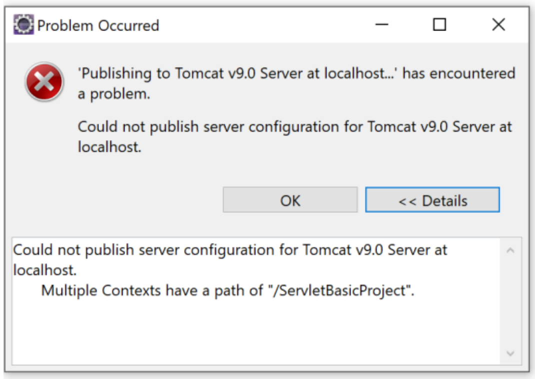
In the Project Explorer, right click on `ServletBasicProject`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `JSPBasicProject`

The main issue with creating a copy of an existing project is that the new copy will retain the context root of the previous project. This will result in a naming conflict on the Tomcat server if you attempt to deploy this project there.

To do this, right click on the project entry and select Run As -> Run on Server. Select Choose an existing server and select the Tomcat entry in the server list. Check the: Always use this server when running the project box. Click Finish.

If ServletBasicProject is still running on Tomcat, then this error message will occur:



'Publishing to Tomcat v9.0 Server at localhost...' has encountered a problem.

Could not publish server configuration for Tomcat v9.0 Server at localhost.

Could not publish server configuration for Tomcat v9.0 Server at localhost.
Multiple Contexts have a path of "/ServletBasicProject".

You need to give the new project a new, distinct context root of its own, which will typically be the same as the project name.

Right click on the JSPBasicProject(ServletBasicProject) from the Server entries first and Remove it it.

Right click on the project entry, select Properties -> Web Project Settings and enter: `JSPBasicProject`. Click Apply, Ok and then click Apply and Close

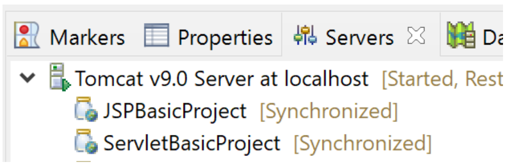
Right click on the project entry, select Show in - > System Explorer. Navigate into `JSPBasicProject/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
    <wb-module deploy-name="JSPBasicProject">
        <wb-resource        deploy-path="/"        source-path="/WebContent"
tag="defaultRootSource"/>
        <wb-resource deploy-path="/WEB-INF/classes" source-path="/src"/>
        <property                                       name="java-output-path"
value="/JSPBasicProject/build/classes"/>
        <property name="context-root" value="JSPBasicProject"/>
    </wb-module>
```

Save the file and refresh the project.

Go to the Servers view and remove the existing conflicting JSPBasicProject instance.

Run the project again in the server. It should now be running with its own name (no conflicts) reported in the list of modules below the Tomcat entry in the Servers view:



Tomcat v9.0 Server at localhost [Started, Rest
JSPBasicProject [Synchronized]
ServletBasicProject [Synchronized]

This time, the project deploys at a new URL with exactly the same view as the previous project

http://localhost:8080/JSPBasicProject/

Delete all the existing JSP files as well as `web.xml`

Delete all the Java source code in `com.workshop.servlets`

We will populate the project from scratch with new content.

3 Declaration, expression and scriptlet tags

Copy the following files from `changes` into `src/main/webapp`:
(NOTE: Be very careful to place these files in `src/main/webapp` and not any of its subfolders such as META-INF or WEB-INF).

```
FirstExample.jsp
```

Run the web app and access it at this URL (either in the built-in browser in Eclipse or in a separate Chrome browser)

http://localhost:8080/JSPBasicProject/FirstExample.jsp

Note: As a shortcut to immediately open the Eclipse browser tab at this URL, right click on the specific JSP file you wish to execute (instead of on the project entry itself) and select Run As -> Run on Server

4 Using JSP implicit objects

Copy the following files from `changes` into `src/main/webapp`:

```
first-form.html
complete-first-form.jsp
```

Copy the following files from `changes` into `src/main/webapp/WEB-INF`

```
web.xml
```

Deploy the web app and access this URL:

http://localhost:8080/JSPBasicProject/first-form.html

Open DevTools and observe the Network tab. Notice that the form parameters are sent as query parameters in the URL itself using a GET method. This is the default method used to submit form data if no HTTP method is explicitly specified.

Provide the form data and notice that the response returned differs on the language selected

5 Servlet forwarding to a JSP

Copy the following files from `changes` into `src/main/webapp`:

```
basic-first-form.jsp
```

Create a new package `com.workshop.servlets` in `src/main/java`:

Copy the following files from `changes` into `com.workshop.servlets` in `src/main/java`:

```
FirstFormServlet.java
```

Make changes to the following files in `src/main/webapp` from `changes`:

```
first-form-v2.html
```

Deploy the web app and access this URL:

http://localhost:8080/JSPBasicProject/first-form.html

Provide the form data and check that the response remains the same.

6 Using Expression Language (EL)

Copy the following files from `changes` into `src/main/webapp`:

```
el-first-form.jsp
```

Make changes to the following files in `com.workshop.servlets` in `src/main/java` from `changes`:

```
FirstFormServlet-v2.java
```

Deploy the web app and access this URL:

http://localhost:8080/JSPBasicProject/first-form.html

Provide the form data and check that the response remains the same.

7 JSP **page** and **include** directive

Copy the following files from `changes` into `src/main/webapp`:

```
page-directive-demo.jsp
demo-error.jsp
```

Copy the following files from `changes` into `com.workshop.servlets` in `src/main/java`:

```
Student.java
```

Deploy the web app and access this URL:

http://localhost:8080/JSPBasicProject/page-directive-demo.jsp

Then access this URL:

http://localhost:8080/JSPBasicProject/demo-error.jsp

Notice the status 500 error returned due to the arithmetic exception in the JSP.

Copy the following files from `changes` into `src/main/webapp`:

```
handle-error.jsp
```

Make changes to the following files in `src/main/webapp` from `changes`:

```
demo-error-v2.jsp
```

Access the URL again:

http://localhost:8080/JSPBasicProject/demo-error.jsp

Notice this time that a different JSP with a customized error message is loaded instead.

Copy the following files from `changes` into `src/main/webapp`:

```
header.html
footer.html
main-content.jsp
```

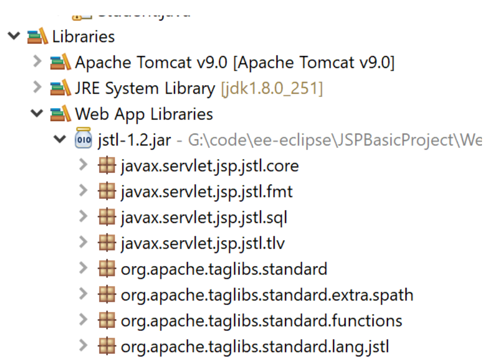
Access this URL:

http://localhost:8080/JSPBasicProject/main-content.jsp

Notice that the styling rule for <h1> in `header.html` is also applied to the content in `footer.html`

8 JSTL Core and Function Tags

JSTL is not part of the runtime libraries of the Tomcat servlet container, so you will need to provide a JAR containing its classes to your web project in order to use it. Copy `jstl-1.2.jar` in `changes` to `WEB-INF/lib` of the project. This places the JSTL classes on your build path. You can view this in the Project Explorer view by expanding on the Libraries -> Web App Libraries.



```
Libraries
  Apache Tomcat v9.0 [Apache Tomcat v9.0]
  JRE System Library [jdk1.8.0_251]
  Web App Libraries
    jstl-1.2.jar - G:\code\ee-eclipse\JSPBasicProject\We
      javax.servlet.jsp.jstl.core
      javax.servlet.jsp.jstl.fmt
      javax.servlet.jsp.jstl.sql
      javax.servlet.jsp.jstl.tlv
      org.apache.taglibs.standard
      org.apache.taglibs.standard.extra.spath
      org.apache.taglibs.standard.functions
      org.apache.taglibs.standard.lang.jstl
```

Copy the following files from `changes` into `src/main/webapp`:

`jstl-core-function-demo.jsp`

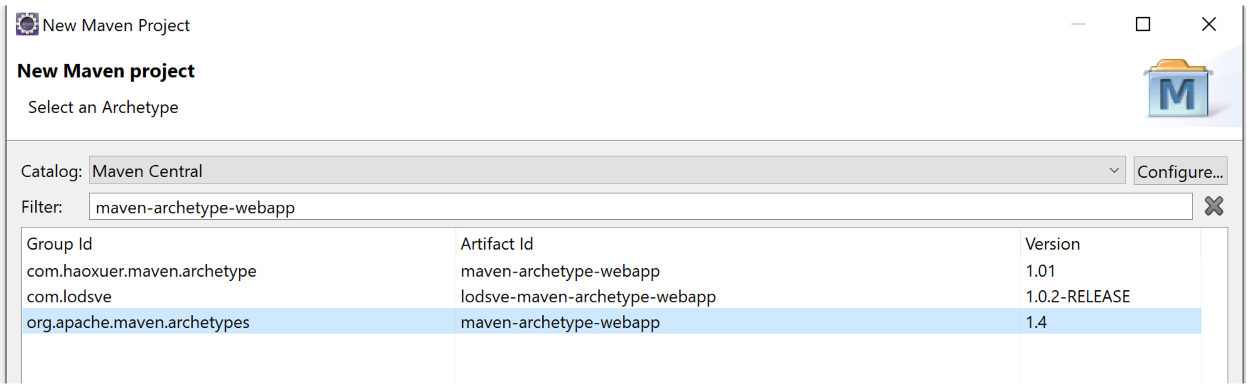
Access this URL:

http://localhost:8080/JSPBasicProject/jstl-core-function-demo.jsp

9 Creating a servlet/JSP webapp as a Maven project

Switch to Java EE perspective.

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-webapp` in the filter. Select the entry with group id: `org.apache.maven.archetypes` and click Next



New Maven project

Select an Archetype

Catalog: Maven Central

Filter: maven-archetype-webapp

Group Id	Artifact Id	Version
com.haoxuer.maven.archetype	maven-archetype-webapp	1.01
com.lodsve	lodsve-maven-archetype-webapp	1.0.2-RELEASE
org.apache.maven.archetypes	maven-archetype-webapp	1.4

Enter in the following details in the New Maven Project dialog box and click Finish

Group Id: `com.workshop.servlets`
Artifact Id: `JSPMavenProject`
Version: `0.0.1-SNAPSHOT`
Package: `com.workshop.servlets`

You will initially have an error flagged in project entry as there is an `index.jsp` generated automatically and the necessary Servlet classes to compile it are not yet present on the build path. To correct this, right click on the project entry -> Properties -> Targeted runtimes. In the dialog box, select the Apache Tomcat Server (or any other targeted application server of choice). Click Apply and Close.

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK. Notice that it now includes a dependency for JSTL as well as for the Servlet API. Although, we have included this project as a Tomcat Targeted Runtime, we still need to add the Servlet dependency as this is necessary for the build process to complete successfully. The scope is set to provided, so that this dependency is not included in the final generated WAR, since the Tomcat server itself will already have this dependency on its classpath.

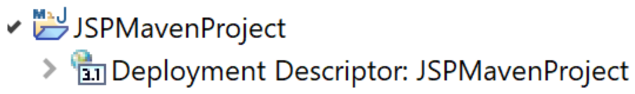
```
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
```

Currently, Web Module (or Servlet) version is set to 2.3. We need to update it to a slightly more recent version (3.1 or 4.0). We will use version 3.1.

Replace the contents of Deployed Resources -> `webapp/WEB-INF/web.xml` in the project with `web.xml` from `changes`. This is based on the schema for Servlet 3.1

Right click on the project entry Properties -> Project Facets. Select Version 3.1 for the Dynamic Web Module and click Apply and Close.

Right click on the project name -> Refresh. The deployment descriptor should now show version 3.1 (This might take some time to appear)


JSPMavenProject
Deployment Descriptor: JSPMavenProject

We now need to update the folder structure to match that expected of a typical Maven web app project:

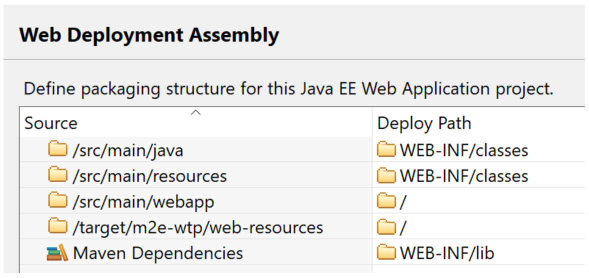
https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

In `src/main`, create two new folders:

- `src/main/java`
- `src/main/resources`

You will notice `src/main/java` is automatically added to the build path.
Right click on `src/main/resources`, select Build Path -> Use as Source Folder. This also adds it to the build path.

Right click on the project, Properties -> Deployment Assembly. This shows how the folders in your current Maven project maps to the document root of the project when it is deployed in the Tomcat server. It should look like something below:

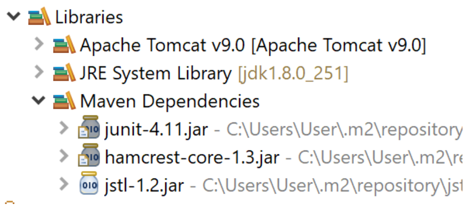


Web Deployment Assembly

Define packaging structure for this Java EE Web Application project.

Source	Deploy Path
/src/main/java	WEB-INF/classes
/src/main/resources	WEB-INF/classes
/src/main/webapp	/
/target/m2e-wtp/web-resources	/
Maven Dependencies	WEB-INF/lib

Notice that all the JARs included in the Maven dependencies will be placed in WEB-INF/lib, which is the default directory in Tomcat for external JAR classes. Recall that we placed `jstl-1.2.jar` into here when we were working on a JSTL feature. If you go to the Libraries entry, you should see the same JAR in the Maven dependencies entry:



```
Libraries
  Apache Tomcat v9.0 [Apache Tomcat v9.0]
  JRE System Library [jdk1.8.0_251]
  Maven Dependencies
    junit-4.11.jar - C:\Users\User\.m2\repository
    hamcrest-core-1.3.jar - C:\Users\User\.m2\re
    jstl-1.2.jar - C:\Users\User\.m2\repository\jst
```

We are now ready to populate the Maven web app project with JSP, HTML and Servlet resources:

In `src/main/java`, create the following package: `com.workshop.servlets`

Copy the following files from `changes` into `com.workshop.servlets` in `src/main/java`:

```
FirstFormServlet.java
Student.java
```

Copy `first-form-v2.html` into Deployed Resources -> webapp and rename it to `first-form.html`

In the same folder (Deployed Resources -> webapp), copy the following files from `changes`

```
header.html
basic-first-form.jsp
jstl-core-function-demo.jsp
```

Right click on the Project, select Run As -> Run on Server. Select the Tomcat server entry and tick the checkbox: Always use this server when running this project. Click finish.

Access this URL:

http://localhost:8080/JSPMavenProject/first-form.html

Enter the form details as usual, and check that the response is the same as was the case when we constructed it using a Dynamic Web Project.

Access this URL:

http://localhost:8080/JSPMavenProject/jstl-core-function-demo.jsp

Verify that the results for the JSTL functionality are the same as was the case when we constructed it using a Dynamic Web Project.

10 Deploying as a WAR to the standalone Tomcat instance

Right click on the project, Run as -> 4 Maven Build.

In the Goals text box, type: `clean package`
Click Run

You should see messages indicating a successful build in the Console:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------< com.workshop.servlets:JSPMavenProject >----------------
[INFO] Building JSPMavenProject Maven Webapp 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ JSPMavenProject ---
[INFO] Deleting G:\code\ee-eclipse\JSPMavenProject\target
….
…
…
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.385 s
[INFO] Finished at: 2020-12-21T10:59:41+08:00
[INFO] ------------------------------------------------------------------------
```

Right click on the project and select Refresh.

Expand the target directory, you should be able to see: `JSPMavenProject.war`

Copy this file to any folder (e.g. Desktop), open it with 7-Zip and browse its contents.
Notice that it has the same contents and directory layout as Deployed Resources - webapp in the Eclipse project which matches that which is expected by Tomcat for a proper web-app deployment. Notice as well that `WEB-INF\lib\` contains the JAR files for the JSTL dependency but not for Servlet 3.1; as this is specified with the scope of <provided>.

Stop the integrated Tomcat instance in Eclipse, or run it on a different port instead so that the Tomcat instance that is running as a Windows service can run on the default port of 8080.

Start up the service from the Services dialog box.
Navigate to: http://localhost:8080/

There are several ways to deploy a web app into a standalone Tomcat server instance:
https://tomcat.apache.org/tomcat-9.0-doc/appdev/deployment.html
Topic: Deployment With Tomcat

Click on Manager App and complete the username / password prompt. You will be navigated to the Tomcat Web Application Manager main page.

In the WAR file to deploy section, select Choose File button. Then select `JSPMavenProject.war` and click Deploy. You should be able to see `/JSPMavenProject` among the list of deployed applications; click on this link to be redirected to the app.

Navigate to:

http://localhost:8080/JSPMavenProject/first-form.html

Verify that the functionality is the same as it was when running in the integrated Tomcat server in Eclipse.

Go to the deployment folder for this Tomcat installation. If you installed it as a Windows Service, then it should likely be at:
`C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`

Just as in the case of the Eclipse instance, notice that the `JSPMavenProject` folder is in here with its contents matching that of Deployed Resources - webapp in the same project in Eclipse

Remember that you can also deploy this app by simply copying and pasting the WAR file into:
`C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`