# Servlet JSP Workshop
# Lab 1

## 1 Lab setup

Make sure you have the following items installed

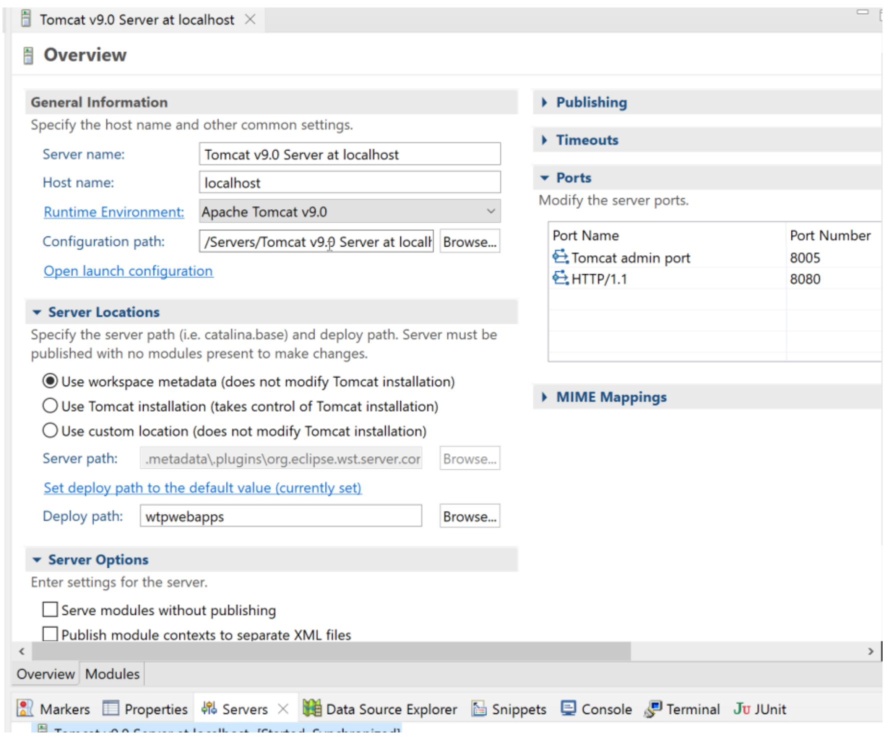
- Latest version of JDK 11 (note: labs are tested with JDK 11 but should work with higher versions with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Tomcat 9.0 installed as a Windows service
- Latest version of Maven
- A suitable text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

The source code for this lab is found in `Servlet-Basic-Project/changes` folder.

Switch to Java EE perspective.

Go to File -> New -> Dynamic Web Project. Type in the project name: `ServletBasicProject`
Make sure that the Target runtime is set to the correct version of Apache Tomcat
Click Next to go to the Source Folders build path dialog box.
Click Next again to go to the WebModule dialog box.
Check the Generate web.xml deployment descriptor box and click Finish.

Eclipse will take a while to complete generating the project.
If the project Java version (given in the JRE System Library) is not compliant with the target JVM that it will be eventually executed on, you can change it by going to Properties -> Project Facets and selecting the correct version of Java there and then selecting Apply and Close.

In the newly generated project, there should be a `web.xml` (the deployment descriptor) in the `src/main/webapp/WEB-INF` directory. Note that if you did not initially generate a deployment descriptor `web.xml` for your project during the creation phase, you can do so later through Eclipse by right clicking on the project, Java EE Tools -> Generate Deployment Descriptor. Create it this way (rather than manually creating a `web.xml` in the `WEB-INF` folder) to ensure that Eclipse takes into account its settings when deploying the app.

The basic folder organization for a Dynamic Web Project in Eclipse:

- The main web resources (HTML and JSP files) should be placed directly in `src/main/webapp` or a subfolder contained within.

- All Java source code (for servlets, POJOs, etc) should be placed in suitable package hierarchy in `src/main/java`
- Any additional external library JARs that your source code references or that your JSP files require should be placed in `src/main/webapp` /`WEB-INF/lib`
- Create appropriate subfolders (`css, images, js`) in in `src/main/webapp` to hold the relevant content.

The `src/main/webapp` folder represents the contents of the WAR file that will be deployed to the server. Any files that are not placed in this folder are considered development-time resources (for example, Java source code files, SQL scripts, etc), and are not deployed when the project is unit tested or published. Also note that any files placed in `WEB-INF` are NOT deployable, thus be very careful not to accidentally place any of your HTML / JSP files in here.

## 3 Working with <welcome-file-list> in web.xml

https://beginnersbook.com/2014/04/welcome-file-list-in-web-xml/

Create the following 5 new files in WebContent (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).

```
first-welcome.jsp
index.html
index.jsp
second-welcome.html
third-welcome.jsp
```

Replace the contents of the existing `WEB-INF/web.xml` with `web.xml` from `changes`

To run the web application, 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.

Eclipse will open up a browser tab in the Editor view at the deployment URL of this app in Tomcat, which in this case is:

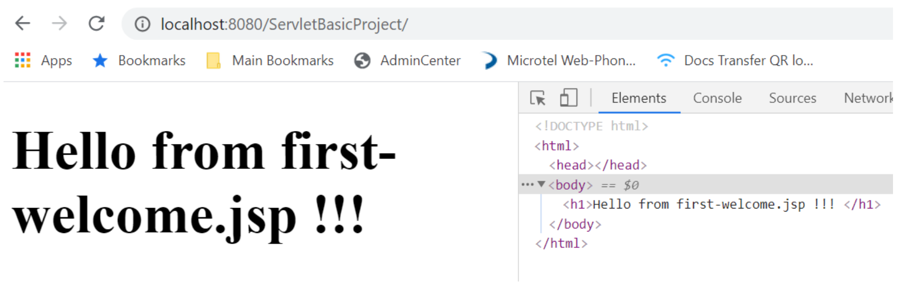
http://localhost:8080/ServletBasicProject/

and you should see the contents from `first-welcome.jsp` being displayed.

Try typing this URL into a browser tab in a standalone browser such as Chrome or Firefox. You should also be able to view the contents from `first-welcome.jsp`

All major browsers provide facility to inspect and interact with the HTML content rendered in the browser. In Chrome, select: More tools -> Developer tools.

Select the Elements view, and you should be able to see the HTML for the content currently rendered in the browser view.

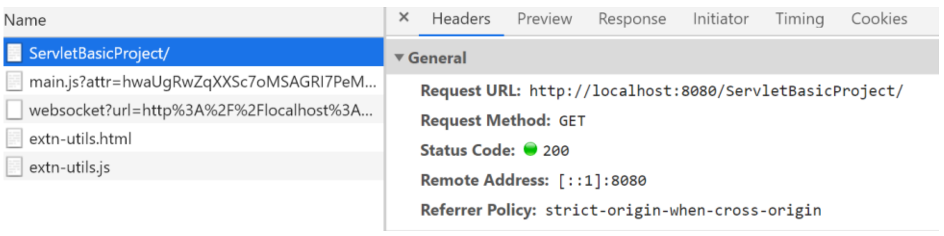


Hello from first-welcome.jsp !!!

```
<!DOCTYPE html>
<html>
  <head></head>
  <body> == $0
    <h1>Hello from first-welcome.jsp !!! </h1>
  </body>
</html>
```

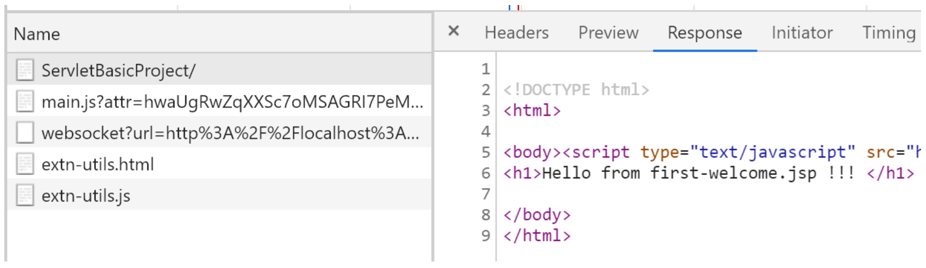
Select the Network view and refresh the page (F5). You should be able to see list of outgoing HTTP requests from the browser. Some of these requests may be related to plugins that you already have in your browser (for e.g. for your antivirus software), however you should see at least one request for the URL: `/ServletBasicProject`.

Click on this and in the Headers view you should be able to see information about the outgoing HTTP request (URL, Request Method, remote address, request headers) as well as the HTTP response (status code, response headers).



**Request URL:** http://localhost:8080/ServletBasicProject/
**Request Method:** GET
**Status Code:** ● 200
**Remote Address:** [::1]:8080
**Referrer Policy:** strict-origin-when-cross-origin

In the Response view, you can see the actual content of the HTTP response body (in the event there is content).



```

<!DOCTYPE html>
<html>

<body><script type="text/javascript" src="
<h1>Hello from first-welcome.jsp !!! </h1>

</body>
</html>
```

A web app is deployed into Tomcat at a context root, which is usually the same as the project name in Eclipse (in this case, ServletBasicProject).
You can view the current context root by right clicking on the project -> Properties -> Web Project Settings.

The first part of the URL path after the domain name / port number (localhost:8080) is always the context-root, i.e.:

http://localhost:8080/ServletBasicProject/

Change the order of listing of the three `<welcome-file>` elements in `web.xml`. To check which file is now loaded by the server, you can reload the web app to the integrated Eclipse server by:

- Right click on the Tomcat server entry in the Servers tab, and selecting Restart
- Right click on the project, Run As -> Run on Server

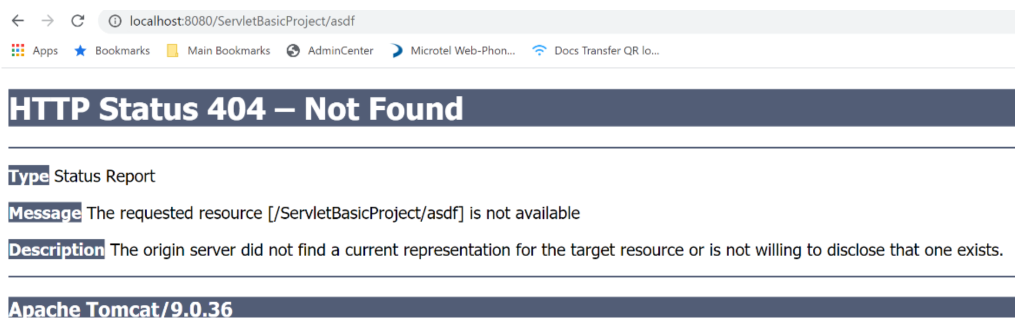
By default, Eclipse will periodically reload the web app to the integrated Eclipse server after you make changes to any of the relevant files in the web project. However, if these changes do not appear, then do an explicit reload by using any of the two steps above.

Refresh the browser tab to reload the web app. Verify this change in your standalone browser.

Notice that you can always retrieve a specific file by appending it to the end of the basic deployment URL, for e.g.:

http://localhost:8080/ServletBasicProject/second-welcome.html
http://localhost:8080/ServletBasicProject/third-welcome.jsp

Notice also that if you type a random second part to the initial deployment URL path( for e.g. http://localhost:8080/ServletBasicProject/asdf), the server returns a 404 Status error message. This is because the server attempts to find a mapping for a resource (such as a servlet) for the `asdf` portion of the path and fails to do so. You can check the Network view in the Chrome Developer tools to verify this in more detail



HTTP Status 404 – Not Found

Type Status Report

Message The requested resource [/ServletBasicProject/asdf] is not available

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/9.0.36

Comment out the entire `<welcome-file-list>` block of elements (highlight the statements, right click -> Source -> Toggle Comment). Save and reload.

Notice now that `index.hml` is loaded.

If you are having issues obtaining the correct results in the embedded browser in Eclipse, do this in a standalone browser (Chrome, Firefox, etc) and make sure you rerun the project or restart the server.

Rename `index.hml` to `temp.html`. Reload the app.

Notice now that `index.jsp` is loaded.

Rename `index.jsp` to `temp.jsp`. Reload the app.

Notice now that the server returns a 404 Status error message as it is no longer able to find any of the default files that it is set up to look for.

Uncomment the entire `<welcome-file-list>` block of elements. Save and reload. Confirm that the relevant files are displayed accordingly as usual.

## 3.1 Schema for web.xml

It is important to double check that you are using the right schema version and layout for `web.xml`, especially when you are working with examples from the Internet that use older versions.

The official schema page:
http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html

All Java EE 7 and newer Deployment Descriptor Schemas share the namespace http://xmlns.jcp.org/xml/ns/javaee/. This corresponds to Servlet 3.1 (Java EE 7) and 4.0 (Java EE 8) specifications.

Java EE 6 and older Deployment Descriptor Schemas share the namespace: http://java.sun.com/xml/ns/javaee. This corresponds to Servlet 3.0 (Java EE6) and Servlet 2.5 (Java EE5).

Schemas with a DTD with a DOCTYPE DTD are even older.

Example of schemas for `web.xml` for Servlet 3.1 and 4.0. Replace `x.y` in the snippet below with 3.1 or 4.0

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_x_y.xsd"
   version="x.y">

……………
……………
……………
……………

</web-app>
```

Example of schemas for `web.xml` for Servlet 3.0 and 2.5. Replace `x.y` in the snippet below with 3.0 or 2.5

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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-app_x_y.xsd"
    version="x.y">
……………
……………
……………
……………

</web-app>
```

## 4 Defining and mapping a servlet in web.xml

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`:

```
FirstServlet
SecondServlet
```

Make changes to the following files from `changes`:

```
web-v2.xml
```

Deploy the web app as usual. Go to:

http://localhost:8080/ServletBasicProject/awesome

and verify that the HTML content from `FirstServlet` is returned to the browser. You should also be able to see the console output from the `System.out.println` in the `doGet` method

Check that the following URLs map to `SecondServlet` by typing them manually into the address bar and verifying that the HTML content from `SecondServlet` is returned to the browser

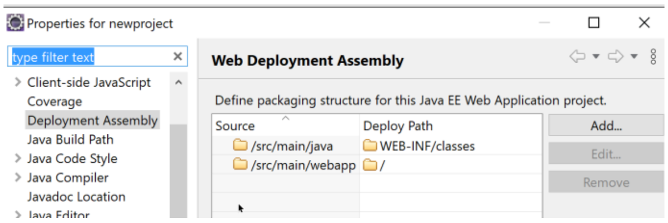
http://localhost:8080/ServletBasicProject/cool
http://localhost:8080/ServletBasicProject/first/second
http://localhost:8080/ServletBasicProject/third
http://localhost:8080/ServletBasicProject/third/fourth
http://localhost:8080/ServletBasicProject/random.html
http://localhost:8080/ServletBasicProject/second-welcome.html

Notice that the last URL still maps to `SecondServlet`, even though there is actually a `second-welcome.html` file in `WebContent` that could be returned. The mapping in `web.xml` thus overrides the default mapping of the web server to resources in the web project.

## 5 Deploying as a WAR to the standalone Tomcat instance

When you create a web app project that uses Tomcat as its run time environment, Eclipse will deploy this project to Tomcat server using the Properties -> Deployment Assembly Settings. This matches the particular folder structure of the project to its deployment folder structure in Tomcat.

For Dynamic Web projects in newer versions of Eclipse, the Deployment Assembly Settings should look like:



Define packaging structure for this Java EE Web Application project.

| Source | Deploy Path |
|---|---|
| /src/main/java | WEB-INF/classes |
| /src/main/webapp | / |

The / in Deploy Path is the "document root" of the web application mentioned in https://tomcat.apache.org/tomcat-9.0-doc/appdev/deployment.html#Standard_Directory_Layout

For Dynamic Web projects, the application source code in the `src/main/java` folder of the project is compiled and stored temporarily in `build/classes` (you can't see them in the Project Explorer in Eclipse, you will have to browse the workspace directory using Windows explorer for this purpose). This is then finally transferred to `WEB-INF/classes` at deploy time.

Everything from `src/main/webapp` is copied to the "document root" directory. This folder typically contains the META-INF, as well as any initial HTML or JSP files that are loaded when the web app is started.

The `WEB-INF/lib` deployment folder contains all the external library JARs required by the project to execute properly. For a Dynamic Web project, all external JARs that was placed in `src/main/webapp` /`WEB-INF/lib` will be deployed here.

If you generate a WAR file for this project (i.e. right click on the project and select Export As -> WAR), it will also use the Web Deployment Assembly Settings to generate the specific folders and their content in the WAR file.
In addition, there will be also be a META-INF folder in the WAR root directory, which contains the JAR manifest

Right click on the project, Export -> WAR file. In the WAR Export dialog box, click the Browse button and select a suitable folder (for e.g. Desktop) to save the `ServletBasicProject.war` file in. Then click Finish.

Open the WAR file with 7-Zip and browse its contents and verify the structure.

Before starting up the standalone Tomcat instance that you had previously installed as a Windows service, you need to ensure that the current integrated Tomcat instance has stopped running or is setup to listen on a different port than the standalone Tomcat instance to prevent a port conflict.

To do that, double click on the Tomcat entry in the Servers view, and in the Overview service select an alternative port number for HTTP/1.1. in the Ports Entry (for e.g. 8181) then save (Ctrl-S) and exit.

Alternatively, to set up the standalone Tomcat instance that is running as a Windows service to run on a different port from 8080:
https://mkyong.com/tomcat/how-to-change-tomcat-default-port/

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 `ServletBasicProject.war` and click Deploy. You should be able to see `/ServletBasicProject` among the list of deployed applications; click on this link to be redirected to the app.

http://localhost:8080/ServletBasicProject/

Verify that the functionality is exactly the same as when it was 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 `ServletBasicProject` folder is in here with its contents matching that of WebContent in the same project in Eclipse

Click on the Undeploy button for the `/ServletBasicProject` entry. The application disappears from the list and the project folder is also removed from `webapps`

Another way to deploy is to simply copy and paste `ServletBasicProject.war` into the `webapps` folder. After a short while, Tomcat will automatically unzip this WAR into a project folder and the app is now deployed (you should be able to see it listed in the Tomcat Web Application Manager).