Create a translation application by using Watson services, Eclipse, and Bluemix, Part 1: **Set up the environment**

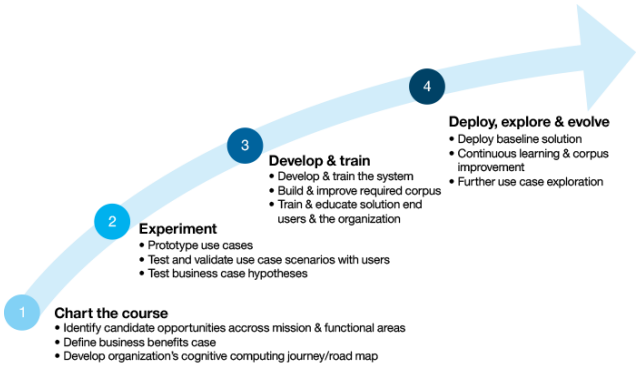
John J. Andersen

December 05, 2016

In this series, learn how to create a translation application with a speech to text front end. The tutorials in the series cover setting up your environment and then using Java programming to develop the cognitive services.

View more content in this series

Cognitive computing offers great promise and benefits, but it can also introduce disruption. It relies on a robust cloud environment and a strong API economy much more than traditional computing, and the methods that are used to develop cognitive services and capabilities demand an increasing reliance on agile methodology and experimentation. However, no matter how disruptive change can be, it is possible to see a path forward. The disruption from cognitive computing, with its shift from traditional programming methods to those relying on natural language, machine learning, massive amounts of unstructured data, and born-in-the-cloud heritage, can be minimized by focusing on four steps: charting the course; experimenting; developing and training; and deploying, exploring, and evolving.



Experimentation, in particular, is a useful and practical way of developing agents and services that support cognitive computing. In this tutorial, I explain an experiment I did on how to create a simple translation application that uses two IBM Watson services: Language Translator and Speech to Text. I use the Eclipse IDE and test the application by using the `localhost` before deploying it

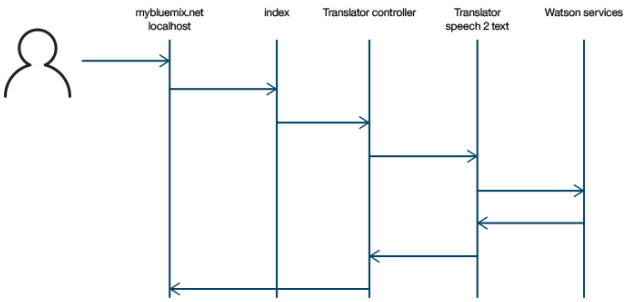
to Bluemix. The series shows you how to set up the Eclipse environment and then develop the translator as a Java servlet controlling the two cognitive services.

Setting up the environment

This tutorial, the first in the series, has the simple objective of setting up the Eclipse environment. To accomplish this, I develop a user experience (UX) using HTML and a Java servlet, test them using the `localhost` and push the artifacts to Bluemix. It's successful if I can see the Translation UX in Bluemix. I create or use the following artifacts in this experiment:

1. Bluemix server using Liberty for Java
2. Local WebSphere Application Server for testing
3. Java servlet controller
4. HTML page

The sequence of events flows from someone using a web browser, through each artifact, and back to the web browser as follows.



1. Get a Bluemix account

If you don't have a Bluemix account, you'll need to get one. Signing up for Bluemix is quick and easy, and you can get a 30-day trial to try it out, which should give you sufficient time to explore the services that are covered in this tutorial. If you already have an account, you can skip to the next step.

Bluemix is a Platform as a Service (PaaS) offering that lets you create, deploy, and run applications. So instead of focusing your time on the plumbing of an IT infrastructure, you can focus on the important job at hand: providing new capabilities to your users. As you explore Bluemix, check out the cost estimator. You will be pleasantly surprised how inexpensive a PaaS application can be. Note that if you decide to enroll in Bluemix beyond 30 days, running this experiment might incur a small cost.

2. Install Eclipse

I developed this app by using Eclipse Neon 1 Release 4.6.1, as this was the most current version of the Eclipse Java **IDE**. Neon comes packaged with IBM Eclipse Tools for Bluemix and provides support for the IBM Node.js Tools for Eclipse. It requires a clean installation, meaning that it will not install over a prior version such as Mars.

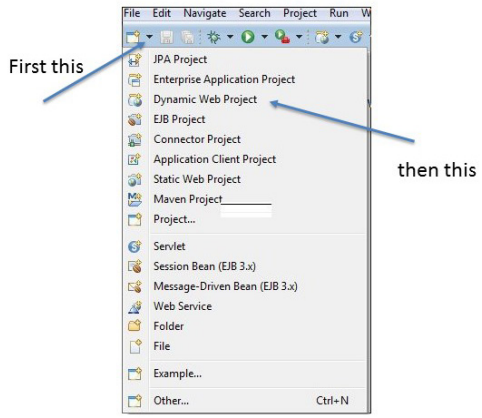
To install Eclipse:

1. Go to Eclipse.
2. Look for a link to download the package. Keep in mind that web pages often change, so if this link doesn't work look for the most current one.
3. Select the package that meets your needs. I selected Eclipse IDE for Java EE Developers for this app. Follow the download instructions. The download copies a compressed file to your system.
4. Extract the file to a location on your system. The eclipse.exe file is used to load the Eclipse Interactive Development Environment (IDE).
5. After loading, Eclipse displays a panel requesting the workspace that will be used to organize the artifacts you will create. I suggest you call this workspace `workspaceBluemix`. By default, Eclipse requests this workspace each time it loads. You also have an option to change the workspace later by selecting **File > Switch Workspace**. Keeping this and other Bluemix labs in a common workspace will simplify reusing artifacts in subsequent apps.
6. When starting Eclipse for the first time, a Welcome screen appears. To proceed to the Workbench, click the **Workbench** in the upper right corner.

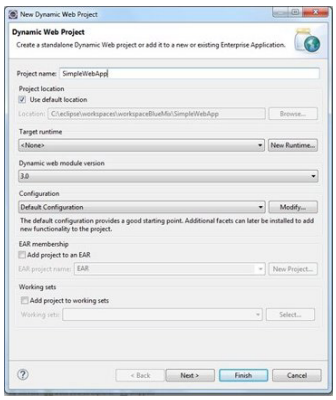
3. Create the project

Eclipse organizes work artifacts in a project. Because the objective of this tutorial is to set up the development environment, I start by creating a project called SimpleWebApp and use it for testing the servers. To do this:

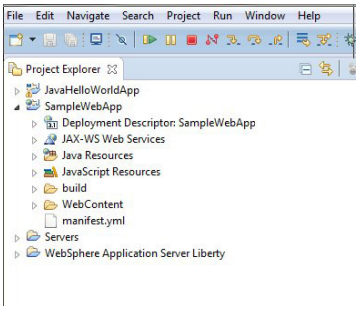
1. Select **New Drop Down** in the icon bar. It's the arrow pointing down below and mid-way between the File and Edit menu options. Then select **Dynamic Web Project**.



   The Dynamic Web Project window appears.

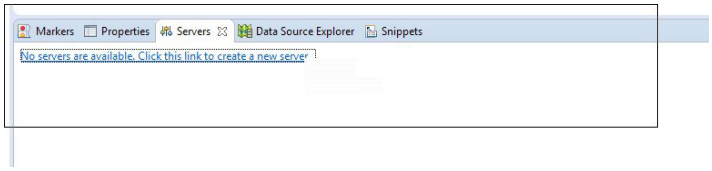


2. Enter `SampleWebApp` as the Project name and select **Finish**. (Note that if a server, for example, IBM Bluemix, is already created, it might appear in the Target runtime option.) The Project Explorer panel now looks similar to the following figure, depending upon any other projects you might have created:

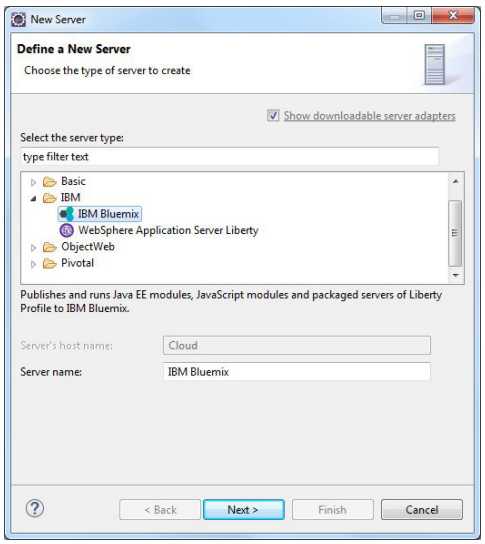


4. Create the Bluemix server for Liberty

If this is a clean installation, there are no servers installed and the Servers tab at the bottom of the workbench will look like this:

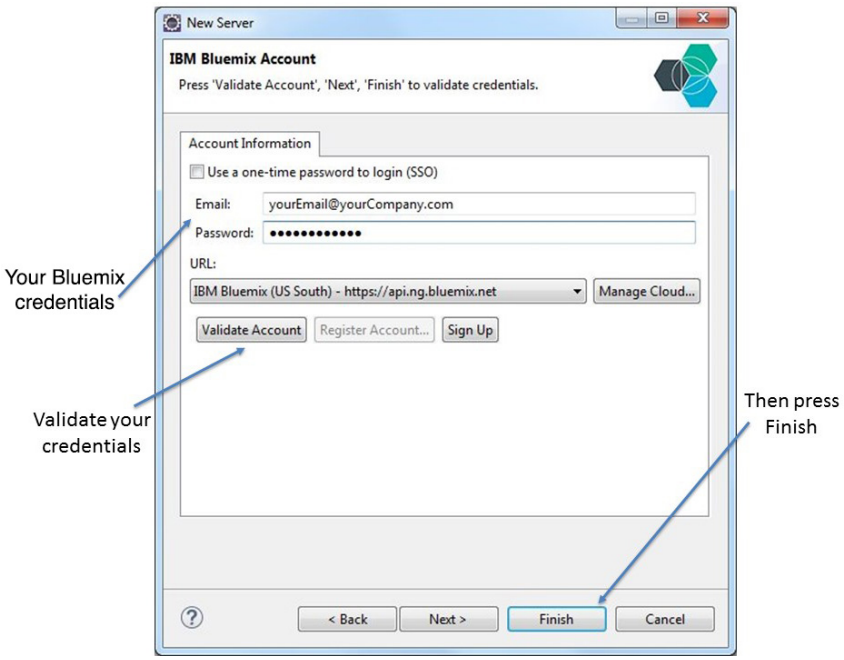


If the panel contains servers (and they're not Bluemix) you create a new Bluemix server by using the process for creating a new project that is described above, except that you select Server vice Dynamic Web Project by choosing **Other > Server > Server**. Both of these approaches get you to the Define a New Server window.

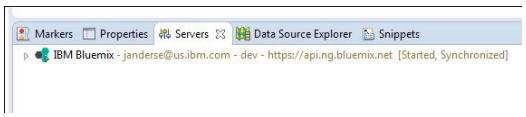


Because this is a Bluemix project, I select the IBM Bluemix option. Leave the server name as IBM Bluemix and click **Next**. Based on the state of your Eclipse instance, you might need to restart Eclipse. Do so and then continue this step.

The IBM Bluemix Account information window appears. You need the credentials that you used to log in to Bluemix. It's a good idea to check your entry by clicking **Validate Account**. After your account is validated, click **Finish**.



This server is used to run the Web UX and ultimately contain information that is produced by the Watson services.

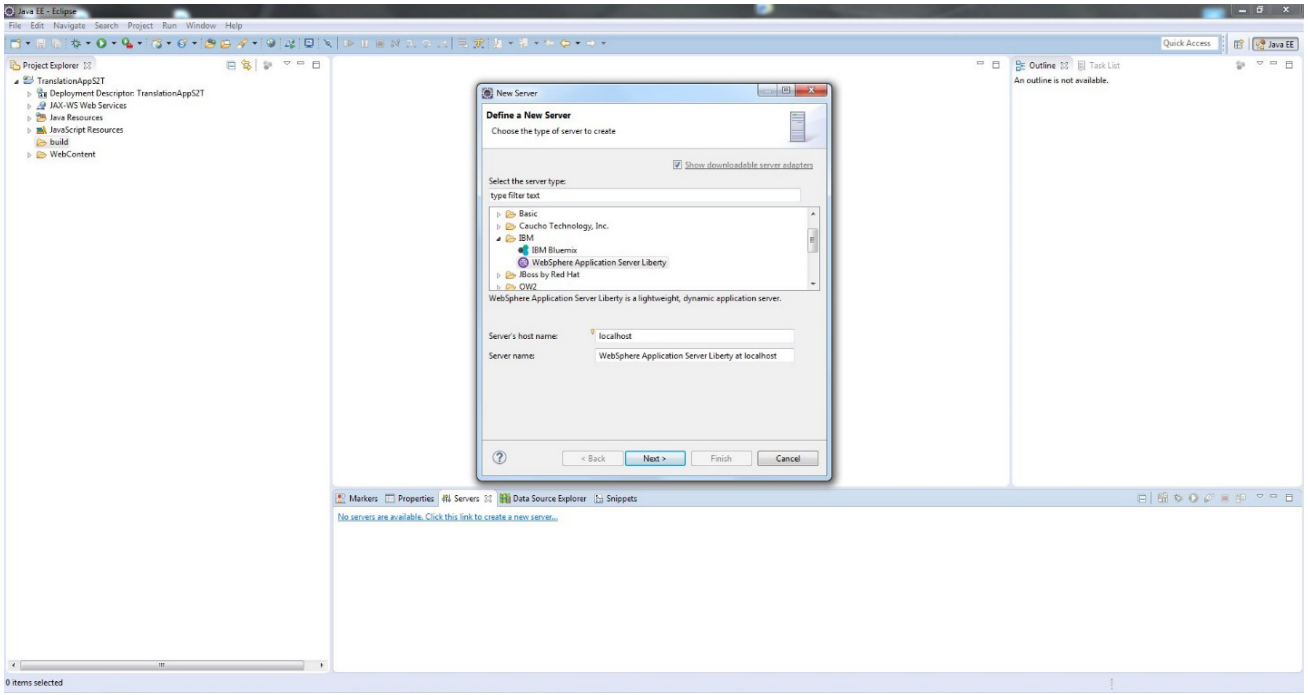


5. Create the local host server

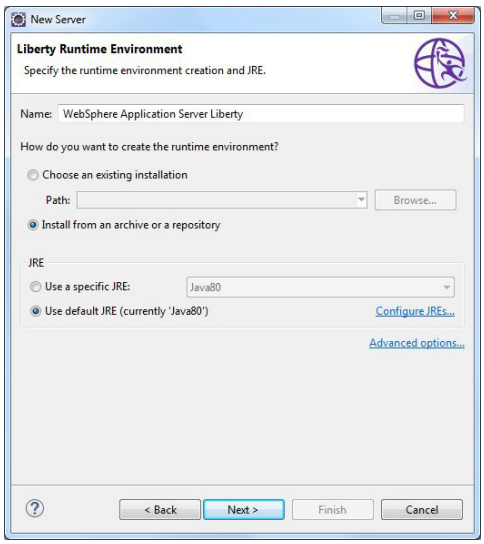
Testing your application locally is a smart strategy, and Eclipse provides for several variations of servers to help you do so. For this app, I use the IBM WebSphere Application Server for Liberty.

You will use the same approach to display the Define a New Server panel as you used for the Bluemix server.

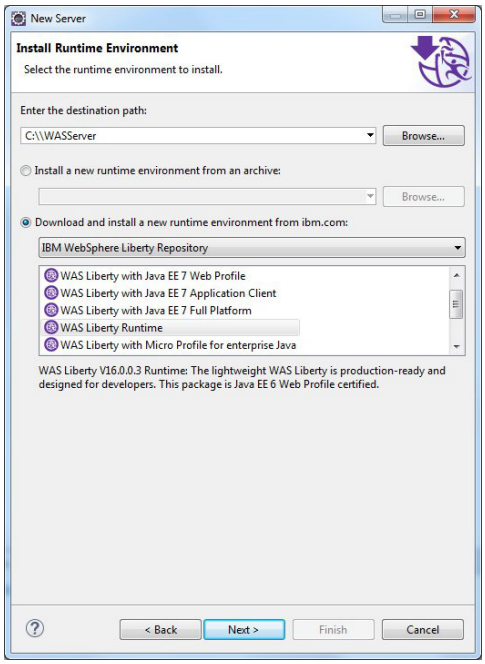
1. Select **IBM > WebSphere Application Server Liberty** and leave the default of localhost for the Server's host name and the default for the Server name. Click **Next**.



2. Select **Install from an archive or a repository**, and leave the default for the JRE. Click **Next**.



3. Enter a destination for the server, for example, WASServer. This app used WebSphere Application Server Liberty Runtime with good success. Click **Next**.

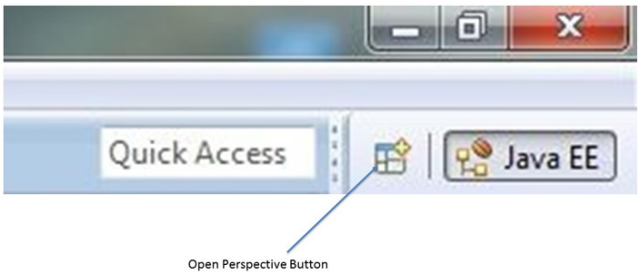


4. Continue to follow the instructions of the installation wizard to complete the installation.

6. Create sample web app HTML page

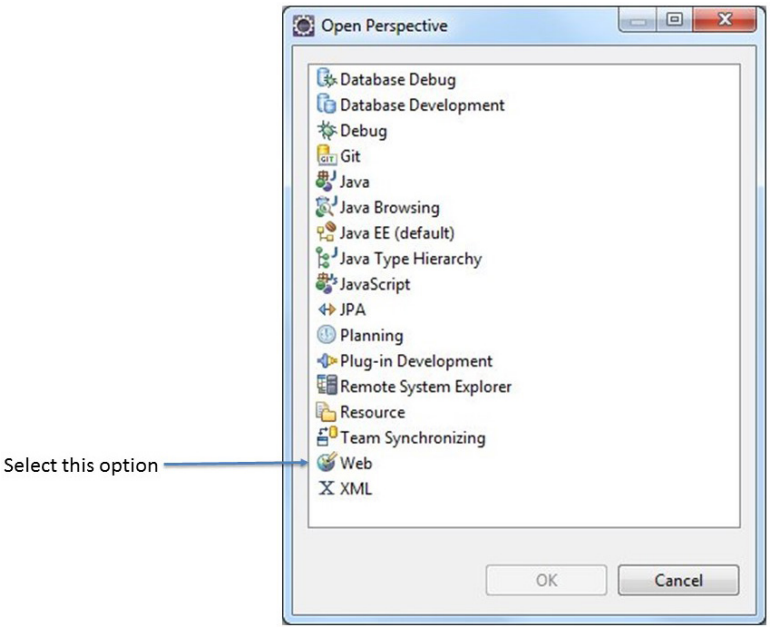
To create a sample web app HTML page:

1. Change the perspective to a Web Perspective by clicking the Open Perspective icon on the



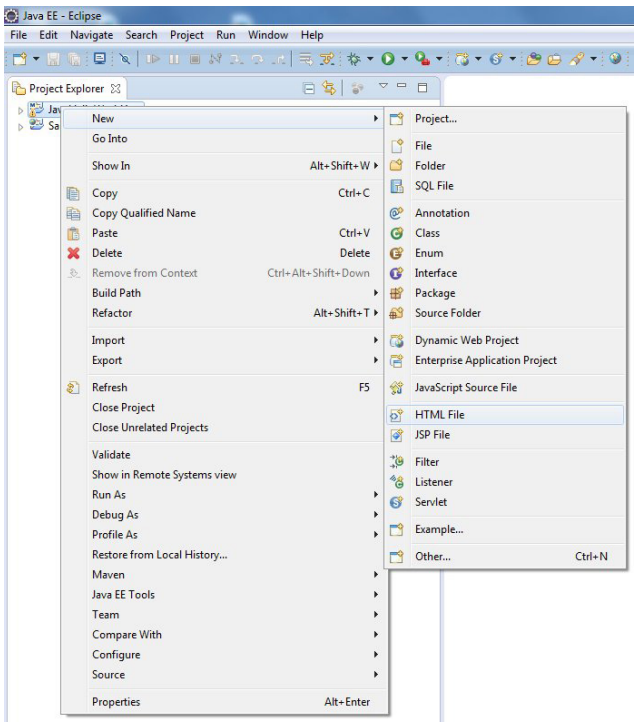
Icon Bar.
Eclipse provides several perspectives, each designed to organize the workbench for the project at hand. For example, Debug provides a debug panel that is essential for debugging an application but unnecessary when you're writing the code for it.

2.

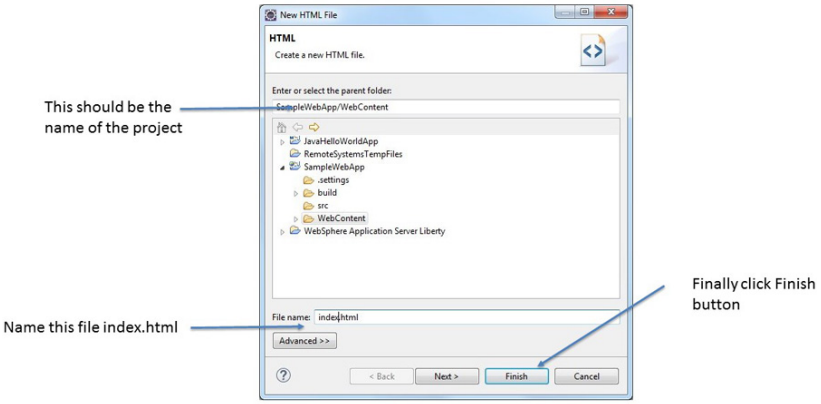


Select the **Web** option.
The perspective for the Eclipse Workbench is now optimized for Web development.

3. In the Project Explorer panel, right-click **SampleWebApp**. In the drop-down menu, select **New > HTML File**.



The create HTML file panel appears. Leave the default for the name of the project, enter `index.html` as the File name, and click **Finish**.



4. Copy and paste the following HTML code into the HTML file.

```
  <!DOCTYPE html>
<html>
 <head>
  <title>The IBM Bluemix Translation Application</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <link rel="stylesheet" href="theme/myStyle.css" />
 </head>
 <body>
  <table>
   <tr>
    <td colspan="2">
     <img class = "eclipseIcon" src='images/WorldIcon.png'>
    </td>
    <td>
     <h4> John J Andersen <span class="blue">Translation Application</span>. </h4>
    </td>
```

```
   </tr>
   <tr>
    <!-- The message comes from /TranslationServlet -->
    <td colspan="4">
     message
    </td>
   </tr>
   <tr>
    <th colspan="2">Enter text to be translated</th>
    <th colspan="2">Translated text</th>
   </tr>
   <tr>
    <td colspan="2">
     <form action = "SimpleServlet">
      <textarea rows = 12 cols = 50 autofocus name = "inputText"></textarea><br>
      <input type="radio" name="target" value ="French" > French <br>
      <input type="radio" name="target" value ="Spanish" checked> Spanish <br>
      <input type="radio" name="target" value ="English"> English <br>
      <input type="submit" name = "translate" Value = "Translate">
      <input type="submit" name = "speech" Value = "Speech2Text">
     </form>
    </td>
    <td  colspan="2" style='vertical-align:top;'>
     <textarea id = "outPut" rows="12" cols="50" readonly name = "translatedText"></textarea>
    </td>
   </tr>
   <tr>
    <!--  This string will identify the language of the input text - it comes from /TranslateServlet --
>
    <td style='vertical-align:bottom;'>
     <h1 id = 'inputLanguage'></h1>
    </td>
   </tr>
  </table>
 </body>
</html>
```

This tutorial assumes that you have some basic HTML skills. Therefore, the structure should be familiar. Note that you can personalize it to your standards. For example, change the following:

- Icon: Note the `<img class` line in the code. You can use any icon available. To copy an icon into the project requires two steps:

1. Create a folder called images under the WebContent directory by using the **New > Folder** menu options. Name the folder **images**.
2. Select this folder and use **Import** to find and import the new icon.

- Author: As much as I can use the publicity, change John J Andersen to your own name.
- Style sheet: Change to a style sheet that you feel is most appropriate. If you do not change this, the screen will look very boring. If you would like to see `myStyle.css`, look at the following code.

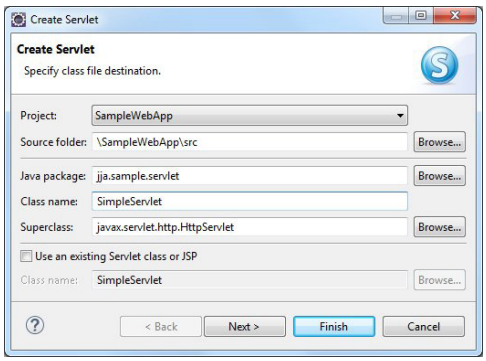
```
@CHARSET "ISO-8859-1";

body,html {
 background-color: #3b4b54; width : 100%;
 height: 100%;
 margin: 0 auto;
 font-family: "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lucida
 Grande", sans-serif;
 color: #ffffff;
}
```

You should add the theme folder that contains the css to the WebContent directory, which should look like:

Note that the images/newapp-icon.png file is also located in the WebContent directory. The file name must match the `img class` name in the HTML file.

7. Create application servlet

You create a new Servlet class by using the same method as creating a new HTML file. The following window appears:



You might consider using your own Java package to contain this `servlet` class. Note that the `class` name `SimpleServlet` must be the same as the servlet name in the HTML file form action line. You can use the following code:

```
package jja.sample.servlet;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

/**
 * Servlet implementation class SimpleServlet
 */
@WebServlet("/SimpleServlet")
public class SimpleServlet
 extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
    {
        response.setContentType("text/html");
```

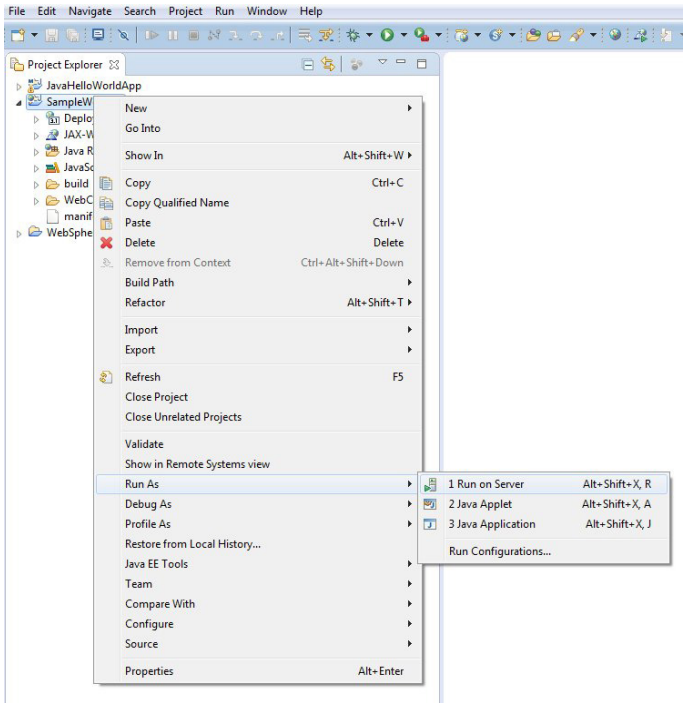
```
        response.getWriter().print("Hello World! from " + request.getServerName());
    }
}
```

This servlet is a placeholder for the more ambitious servlet that I'll develop in Part 2 of this series. If you click **Translate**, the message 'Hello World! from' is displayed.

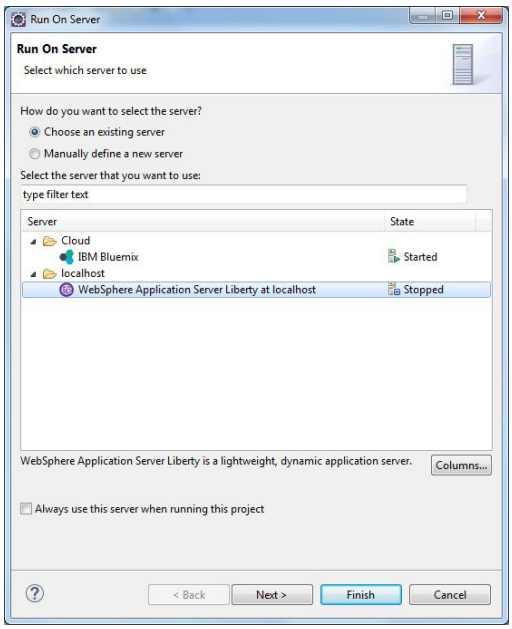
8. Test the application on localhost

It is a good practice to test the web application by using the local instance of WebSphere Application Server. To test it:

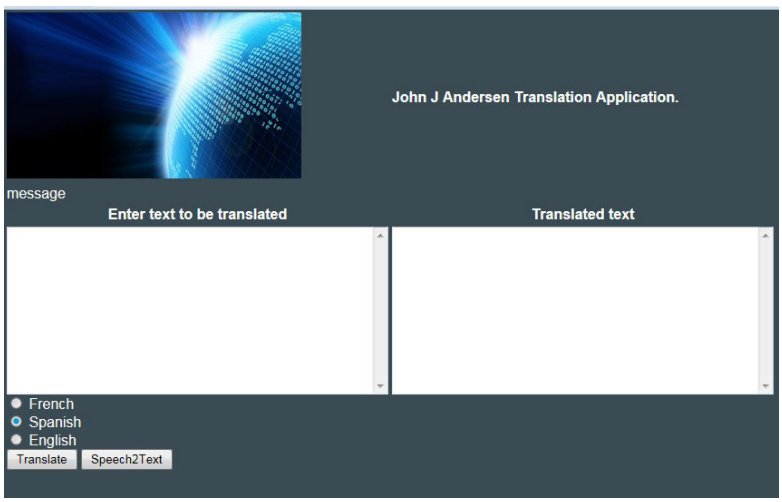
1. Select the application in the Project Explorer by right-clicking it. Then, select **Run As > 1 Run on Server**.



2. Select WebSphere Application Server in the Run On Server window, and click **Finish**.



After you click Finish, WebSphere Application Server displays the following image.

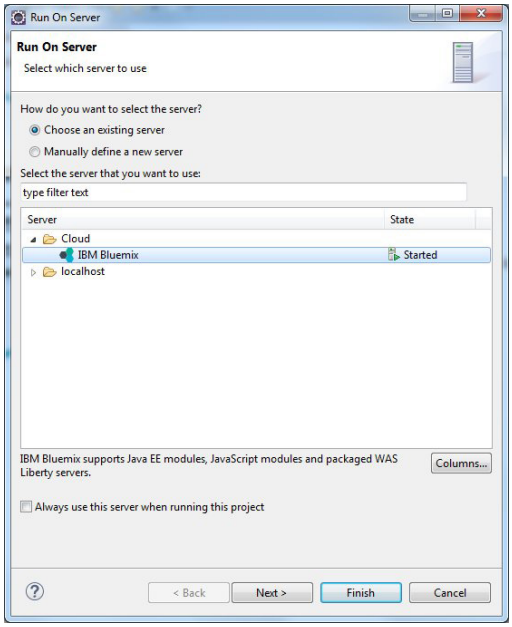


The browser that is used by Eclipse uses its internal cache to improve performance. If you make modifications to the HTML or CSS and those changes do not appear, it is most likely because the browser is using stale code in its cache. Try refreshing the browser to fix this by clicking the refresh icon.

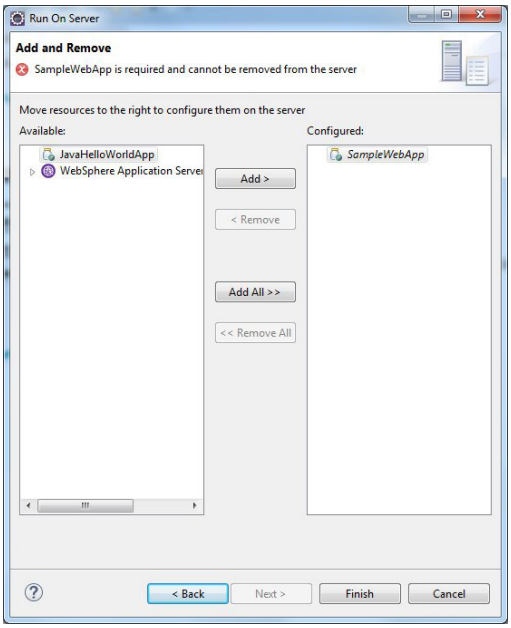
9. Test the application on the Bluemix server

To test the application on Bluemix:

1. Select the application in the Project Explorer like you did when you were testing the localhost server. However, this time select **IBM Bluemix**, then click **Next**.



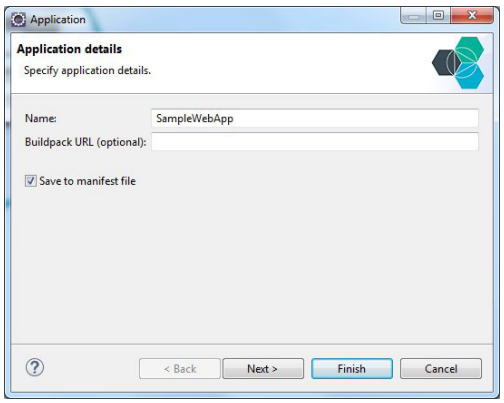
2. After you select **Next**, the Add and Remove window appears. In the Available section, select **SampleWebApp**, then click **Add**. The application moves from the available section to the Configured section.



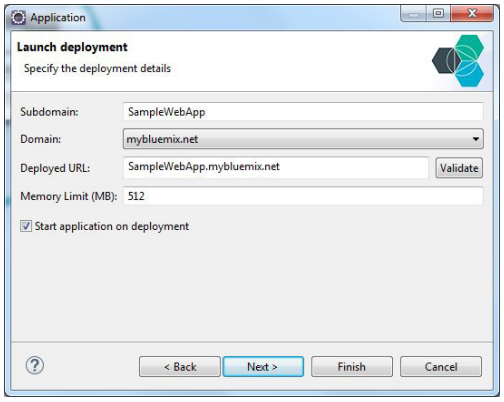
3. Click **Finish**.

Bluemix needs additional information to run the application. This information will be added in the remaining windows and provided to Bluemix in the form of the manifest file.

The Application Details window opens and collects the information for the manifest file.



The name is already provided. Remove any Buildpack information and select the **Save to manifest file** check box. Click **Next** and review the Launch deployment window.



If the Start application on deployment check box is selected, clicking **Finish** will deploy and launch the application. The application can also be started by selecting the application in the Project Explorer and running it on the Bluemix server. In either approach, when the application is selected to run on Bluemix the project is pushed to Bluemix.

Conclusion

Congratulations! You've now finished Part 1 of this tutorial series and should have your Eclipse environment set up. In the next tutorial in the series, I'll explain how to develop the translator as a Java servlet that controls the two cognitive services.