

Create a translation application by using Watson services, Eclipse, and Bluemix, Part 2: Add cognitive services to the app

John J. Andersen

January 31, 2017

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

[View more content in this series](#)

Experimentation is a useful and practical way of developing agents and services that support cognitive computing. This tutorial series explains 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 that controls the two cognitive services.

The first [tutorial](#) in this series explained how to set up the Eclipse environment. This tutorial explains how to attach two cognitive services to the app: Language Translator and Speech to Text. I built this app to show how you can invoke cognitive services in a Java application. Feel free to expand upon this approach by improving the functionality and incorporating extra services.

The target run time remains [Bluemix Liberty for Java](#) for Language Translator. This experiment also continues the best practice of testing the web application by using the `localhost` WebSphere Application Server before pushing the app to [Bluemix](#).

There are two ways to run this experiment:

- Locally using the URL `localhost:9080/TranslationApp`. You can invoke this URL either within the Eclipse IDE or external to the IDE by using a local browser. For help installing an Eclipse local Application Server instance, see Part 1 of the series. This method leverages both the Language Translator service as well as the Speech to Text service.
- Using a Bluemix Liberty server with the URL `https://<TranslationAppName>.myBluemix.net`. This option illustrates the Language Translator service. The Speech to Text service will be enhanced in a future experiment on using Web Socket to establish a connection between a

local microphone agent and the cloud-based server. The <TranslationAppName> will be a unique name that you assign in [Step 8](#).

Getting started

The first tutorial in this series focused setting up the Eclipse and Bluemix environments to develop cognitive applications. That experiment used the Java language as the development language but the steps could apply to other languages such as Node.js.

I used Eclipse Neon Release 4.6.1 to develop the experiment in this tutorial. Neon comes packaged with IBM Eclipse Tools for Bluemix and provides support for the IBM Node.js Tools for Eclipse. If you need to install the Bluemix toolkit into an earlier Eclipse instance or you are looking for the site to download Eclipse go to https://developer.ibm.com/wasdev/downloads/#asset/tools-IBM_Eclipse_Tools_for_Bluemix. For help downloading Eclipse, see [Part 1](#).

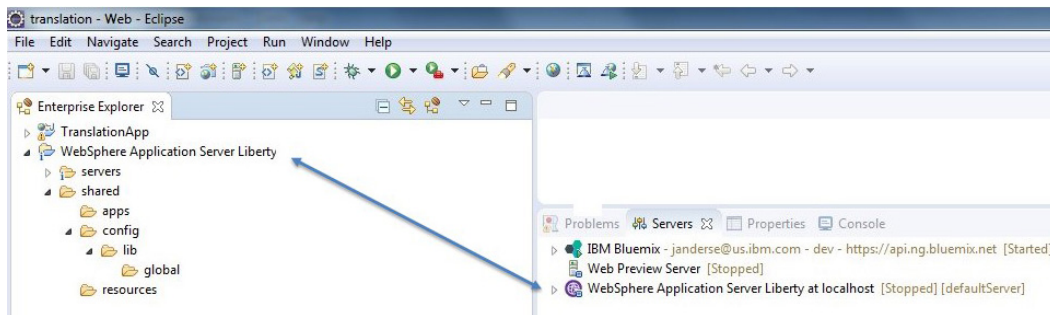
There is the compressed file, [TranslationAppV2.zip](#), that contains the source files. These files will be imported in [Step 3](#).

Start Eclipse and select the workspace that you used for Part 1 of this series. You can select the workspace immediately after starting Eclipse or by using the File menu in the menu bar and selecting **Switch Workspace**. Keeping the artifacts produced by this experiment and other Bluemix experiments in a common workspace simplifies the reuse of artifacts.

1. Update local WebSphere Application Server

[Part 1](#) described how to install Bluemix tools for Eclipse and the local WebSphere Application Server for Liberty. Refer to that tutorial if you need help. I use a new toolset in this tutorial.

1. Review where you installed Application Server and its directory structure. For illustration purposes, assume the installation directory (<WAS Install Directory>) is WASLiberty with the server name defaultServer. To find this server, the directory structure is WASLiberty/usr/servers/defaultServer.
2. WebSphere Developer Tools for Eclipse is a lightweight set of tools for developing and deploying applications to the WebSphere instance. Refer to the following URL for instructions on downloading and installing this toolset: https://www.ibm.com/support/knowledgecenter/SSRTLW_8.5.5/com.ibm.rad.install.doc/topics/t_install_wdt_eclipse.html.
3. In [Step 4](#), you will download a JAR file that contains the classes for the Language Translator and Speech to Text services. You must modify the directory structure of Application Server so that the Watson service classes are visible. Add subdirectories /lib/global to the Application Server <WAS Install Directory>/usr/shared/config directory. This directory is where the cognitive service JAR files will be downloaded.
You can also add this subdirectory in Eclipse by using the File > New > Folder wizard. For either method the Project Explorer view should look like the following image:

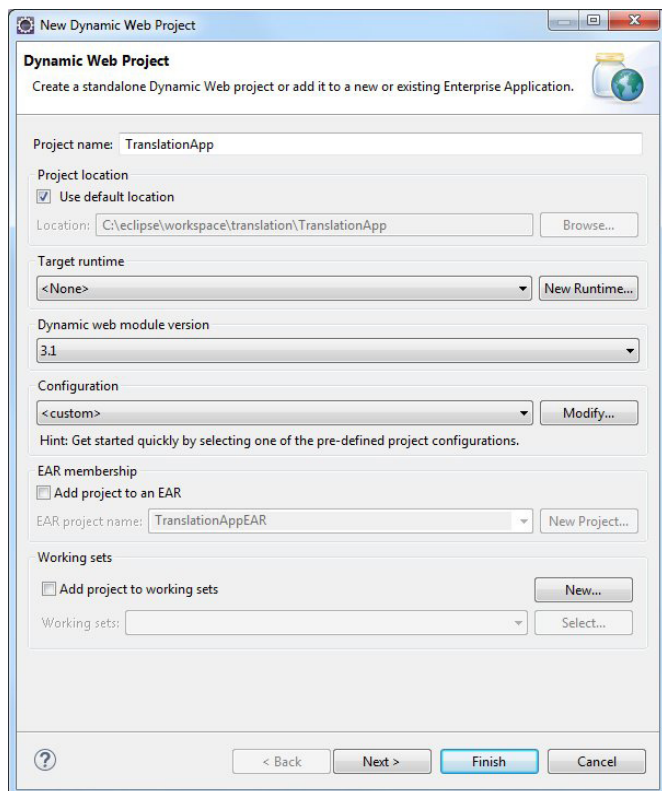


4. In [Step 3](#), you add an environment variable that is structured the same as the Bluemix Watson services environmental variable.

2. Create translation app project

Creating the project is similar to those steps in [Part 1](#).

1. Use the **Create New Dynamic Web Project** wizard to create the project.
2. Name the project **TranslationApp**.
3. Make sure that the Target runtime is None.
4. Use 3.1 for the Dynamic web module version.
5. Make sure the Add project to EAR check box is unchecked.
6. Click **Finish**.



3. Import source files

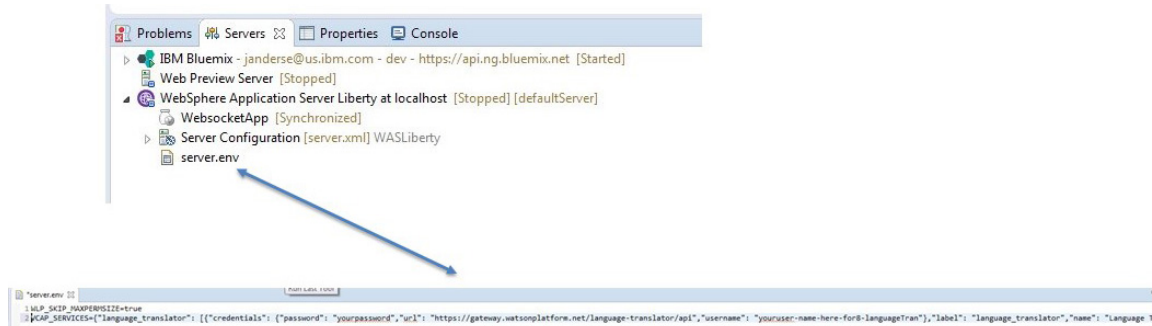
Because the purpose of this experiment is to show you one way of invoking cognitive services in a Java application, I've included a compressed file that contains the source files necessary to complete this experiment— and save you from typing or cutting and pasting. The [source files](#) are:

- TranslateServlet.java
- TranslationProxy.java
- S2TProxy.java
- Index.html

The compressed file also contains the myStyle.css, an icon that you can use if necessary and a template for the JSON object that contains the structure of the `VCAP-SERVICES` environmental variable that can be modified and copied into the server.env file.

Importing files into a project is very straight forward:

1. If you created the project by using [Step 2](#), a folder called Java Resources contains a **src** subdirectory. Right-click the src directory and use the **Import** menu option.
2. In the Select window under General, select **Archive File**, and click **Next**.
3. In the Archive File window, use **Browse** to navigate to the compressed file and then pick the three Java source files to import into the src directory. The errors will be fixed after you download the supporting classes.
4. Follow the same steps to import the index.html file into the WebContent directory.
5. The icon is imported into the WebContent/images directory. Right-click on the WebContent folder and select **New > Folder**. Name the folder images. Then, use the previous instructions to import the icon into this folder.
6. Import the myStyle.css file into the WebContent/theme directory by using the same steps. These steps will ensure the css and icon match the index.html entries.
7. Finally, in the Server window, open the Application Server server.env file. Copy the first line of the jsonTemplate.txt file and paste the line into the server.env file. The server.env file should now have a complete `VCAP_SERVICES` line. You need to determine your user name and password and update that line with that information in [Step 9](#).

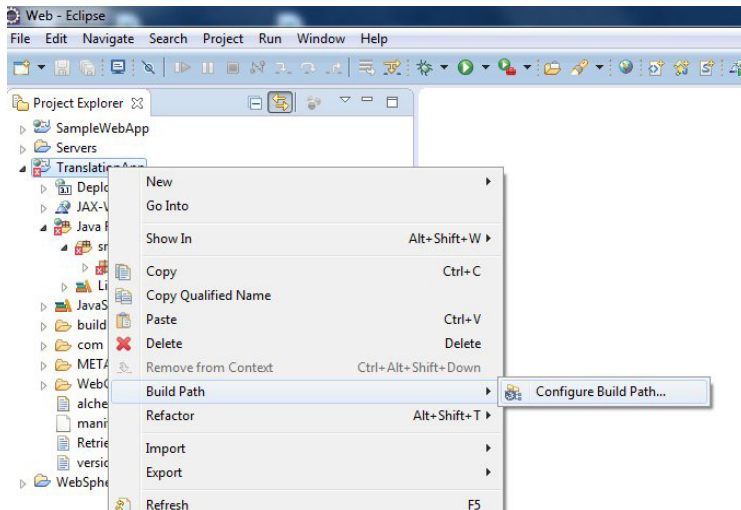


4. Add the Watson Developer Cloud SDK

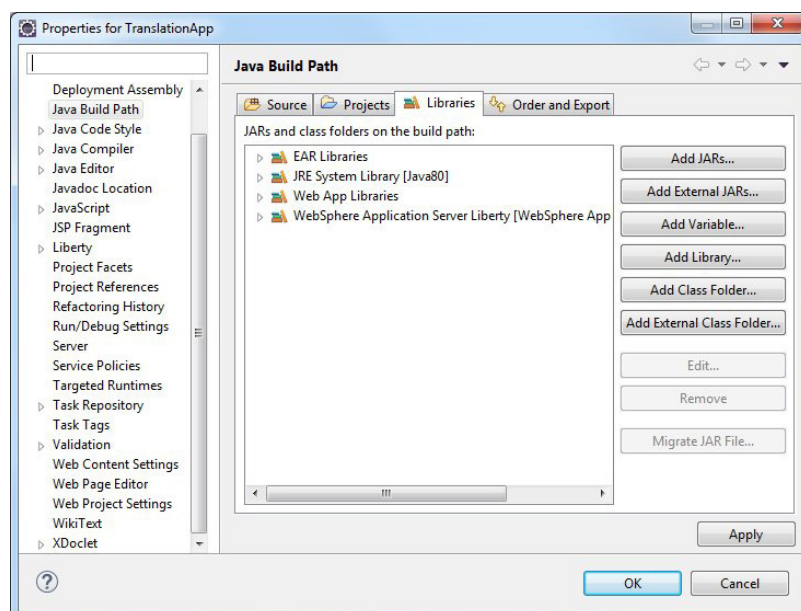
This experiment used the Watson Developer Cloud Java SDK V3.5.2. The SDK can be downloaded from <https://github.com/watson-developer-cloud/java-sdk/releases>. If a more current version is available, use it. Look for Version 3.5.2 Downloads and select the **java-sdk-3.5.2-**

with-dependencies.jar file to download. This SDK supports all Watson and Alchemy service development including Language Translator services.

1. In [Step 1](#), you created a directory that is called `/lib/global`. Save this JAR file to that directory, and the Application Server class loader is able to access it.
2. Select **Project > TranslationApp**, then select **Build Path > Configure Build Path**.

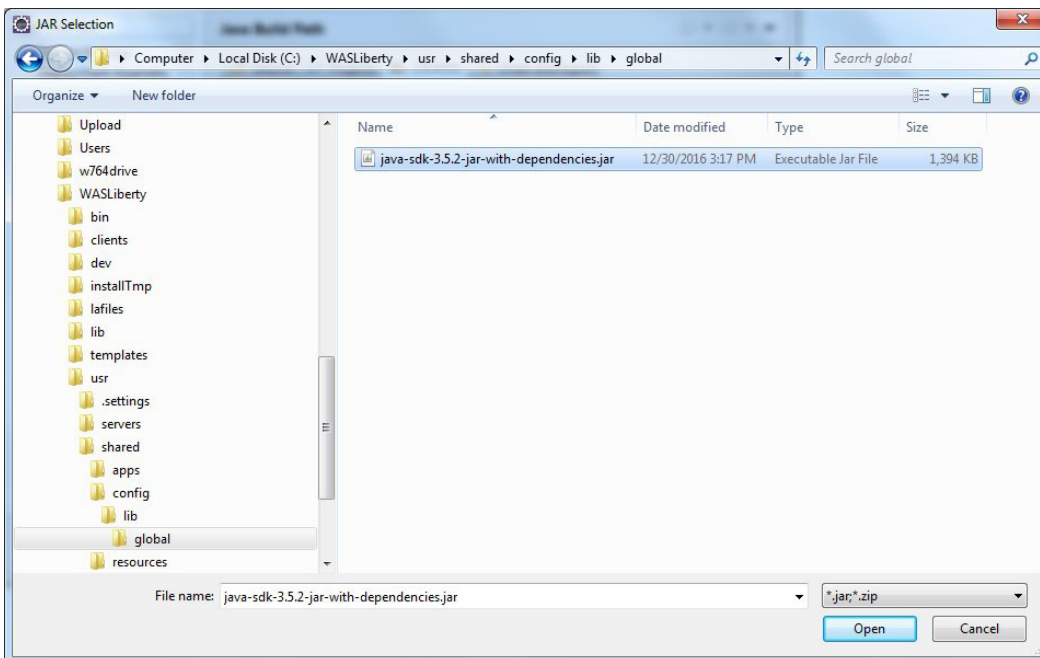


3.

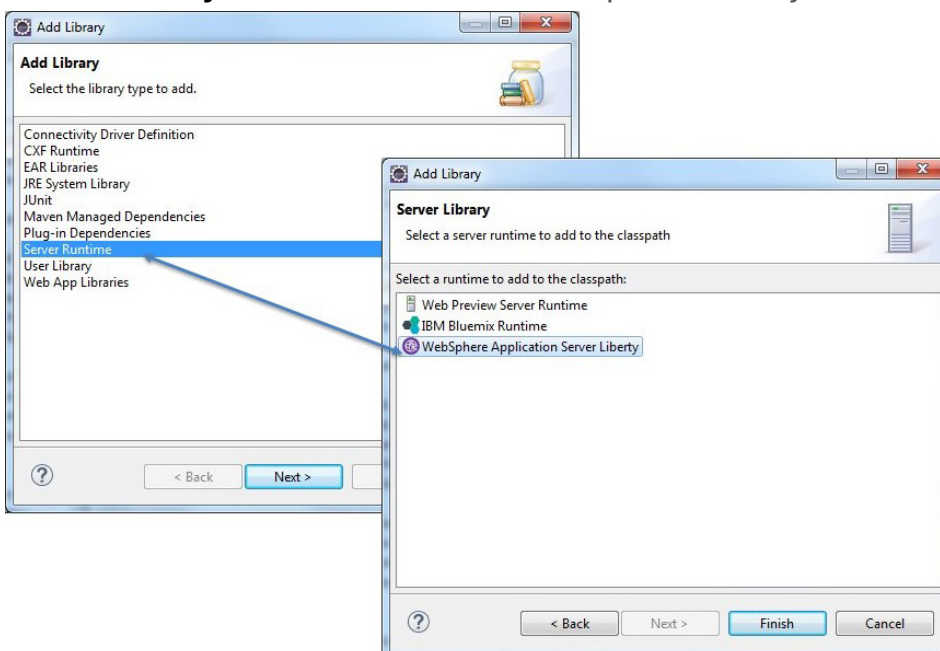


Click **Add External JARs**.

4. Select the JAR file in its directory



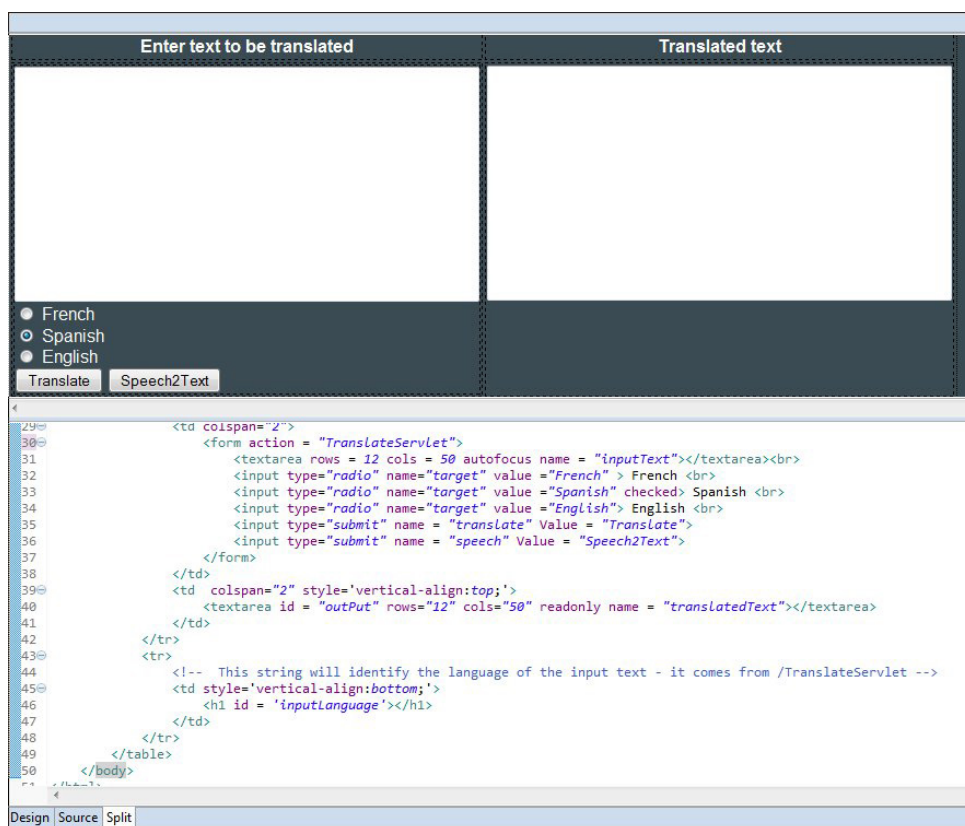
5. Click **Open** in the Java Build Path window. The SDK now appears in the project in Project Explorer as a Reference Library.
6. At this point, most of the errors that are preventing the Java classes from compiling will be resolved. The remaining errors are from missing `Servlet` classes. At run time, both Bluemix and Application Server have these classes available to the class loader. But you must get a clean compile first. To provide these classes to the compiler, I use Task 2 through Task 5 in this step. However, instead of using Add External JAR in Task 3, click **Add Library**. Select **Server Runtime** in the Add Library window and **WebSphere Application Server Liberty** in the next window. At this point, the only errors that remain are warnings.



7. The previous steps were required to get the Watson services and servlet classes visible to the IDE to compile the application classes and to the local Application Server for the application to run in the local host instance. One last step is required to get the Watson service classes to be deployed to Bluemix. You must import the JAR file to the project's WebContent/WEB-INF/lib folder. Right-click the **lib** folder and select **Import**. In the Import Select window, select **General > File System**, and navigate to the directory that contains the JAR file. This maintains the file as a JAR file. Using the Archive option expands the JAR file into individual artifacts.

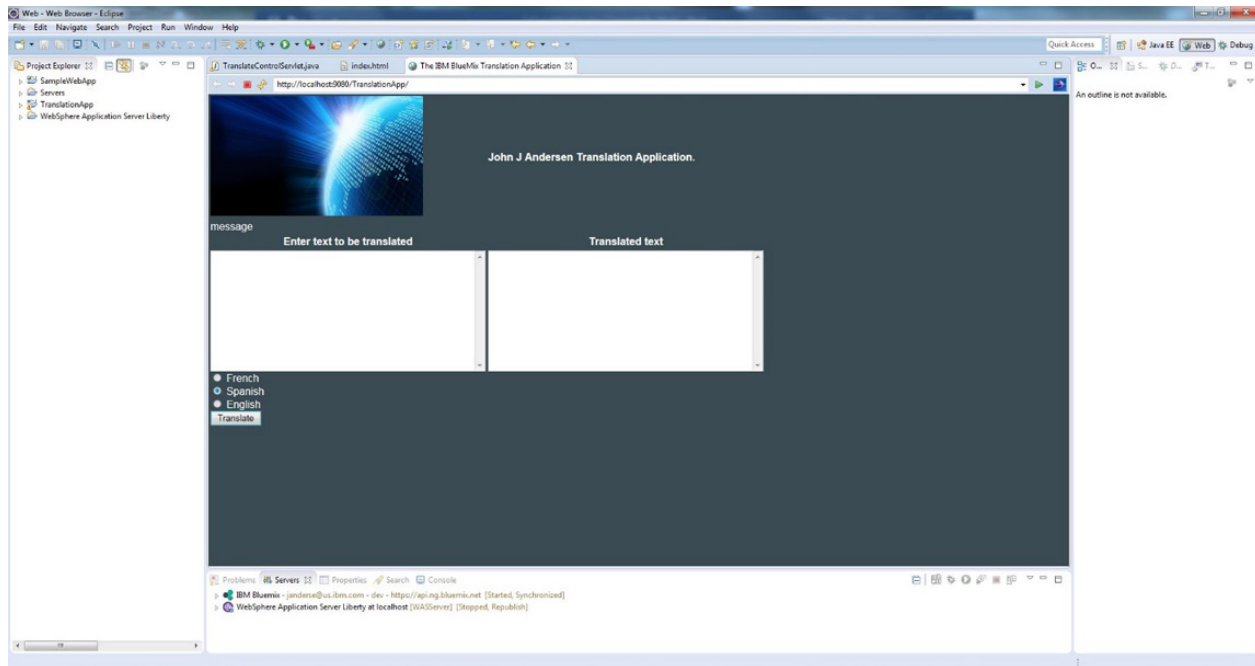
5. Review HTML

The Application Server Developer tool that is installed in [Step 1](#) provides an enhanced view of HTML, as shown below.



The index.html file provides an entry point to the application. It provides two text boxes (`textarea`), one for entering the text to be translated (`inputText`) and the other for the translated text (`translatedText`). The radio buttons define the targeted languages while the command buttons either invoke the translation or the Speech to Text sequences.

The index.html file provides the starting point for this experiment. The source is located in the WebContent folder. The following image shows the initial application window.



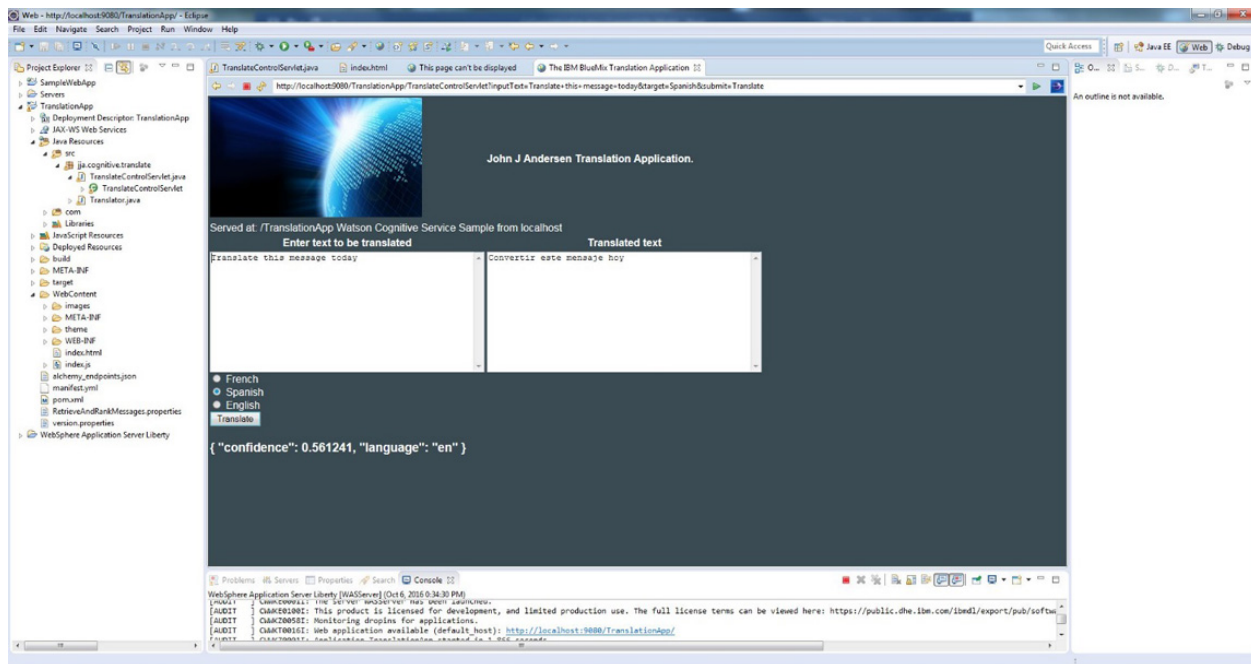
Examine the index.html file to understand how the user experience is managed.

6. Review translate servlet

Servlets have some special characteristics. The `init` method is guaranteed to run once, when the servlet is loaded. That is fine for an experiment that is parsing the JSON object for a user name and password. In production, a `session` would be a better option for saving user information. Each user should probably supply their own user name and password. There is a stub supplied that any aspiring cognitive programmer can leverage.

The bulk of the work in this servlet is performed by `doGet`. `doGet` determines which command button was clicked, constructs the HTML stream for the browser, and manages the proxies.

When the user enters text to be translated and clicks the Translate button the following is displayed:



The text is translated to the language selected by the radio button. Note the message directly under the icon. It points to the server that ran the servlet. In this case, the server was the `localhost` WebSphere Application Server. Had the application been executed on the Bluemix server, its name would be displayed.

In addition to the translated text, the Language Translator can identify the language of the original text, English in this case, along with the confidence derived from the language model. The longer the text to be translated, the greater the confidence in identifying the correct original language. The Language Translator service also has the capability, not explored in this experiment, to produce models of the original language so that the idiosyncrasies of the author can be captured and used to produce a more exact translation.

7. Review cognitive service proxies

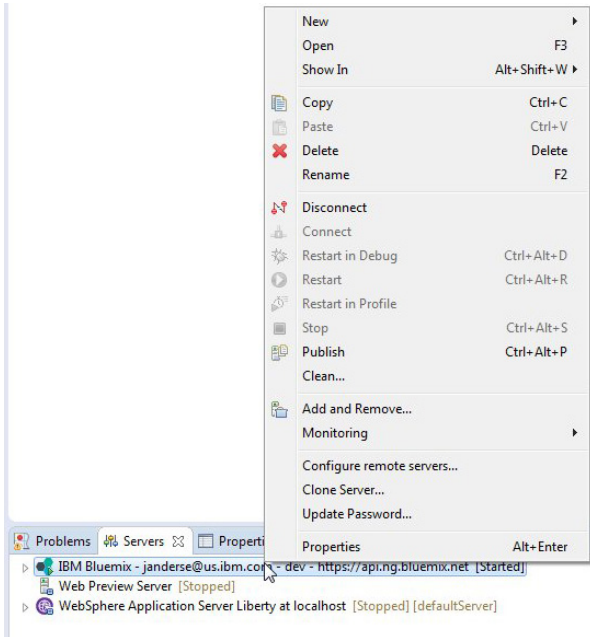
Both proxies are supplied as starter code by IBM through GitHub. For example, the Translator proxy is derived from a sample found at GitHub: https://github.com/watson-developer-cloud/java-sdk/tree/master/language-translator/src/main/java/com/ibm/watson/developer_cloud/language_translator/v2. IBM provides this sample to help you incorporate language translations into your application quickly. Examine it carefully. The `TranslationProxy` in this experiment modified the GitHub sample to offload the `TranslationServlet` from the responsibility of interacting with the Watson service endpoint. By abstracting that function from the servlet, the UX can be isolated from changes in the behavior of the API set.

8. Add project to Bluemix

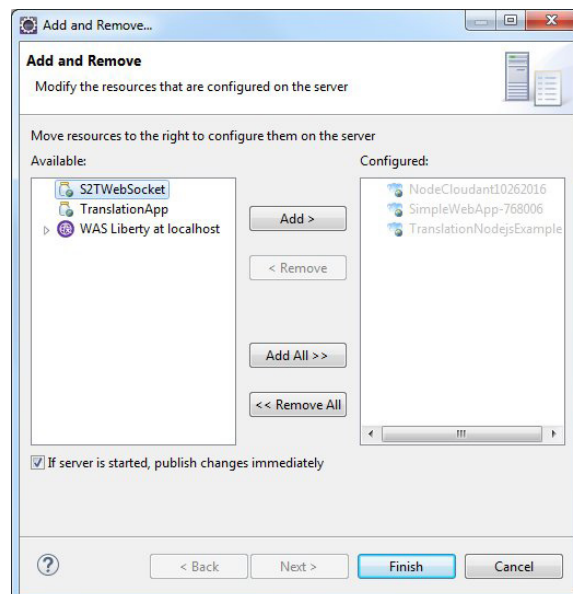
It's a good and productive practice to develop an application and test it using the local Application Server instance. However, Bluemix provides the credentials necessary to access the cognitive services. So, now is the time to push the app to Bluemix. After the credentials are available, you

can use the local Application Server instance. The following steps explain how to deploy the app to Bluemix.

1. Select the Bluemix server in the Server tab and click **Add and Remove**.

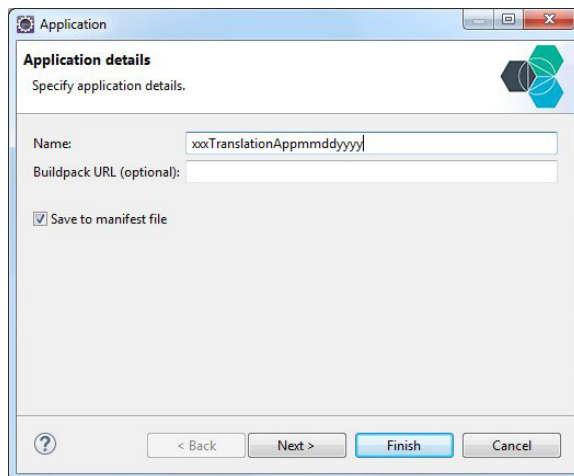


- 2.



Select **TranslationApp** and click **Add**.

3. Click **Finish**, and then Bluemix must collect some additional information. The Application

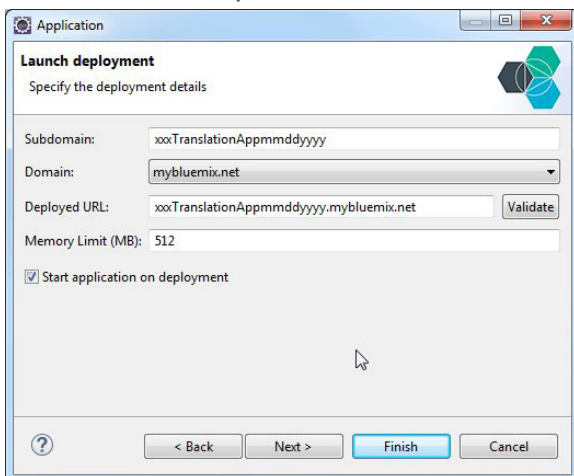


The screenshot shows a dialog box titled "Application" with a sub-header "Application details". Below the sub-header is the instruction "Specify application details." There are two text input fields: "Name:" with the value "xxxTranslationAppmmdyyyyy" and "Buildpack URL (optional):" which is empty. Below these fields is a checked checkbox labeled "Save to manifest file". At the bottom of the dialog are four buttons: a help button with a question mark, "< Back", "Next >", and "Finish".

details window starts that collection process.

4. Enter a unique name, such as <your initials>TranslationApp<today's date>, and check the **Save to manifest file** check box. We'll look at the contents of the manifest file later. Click **Next**.

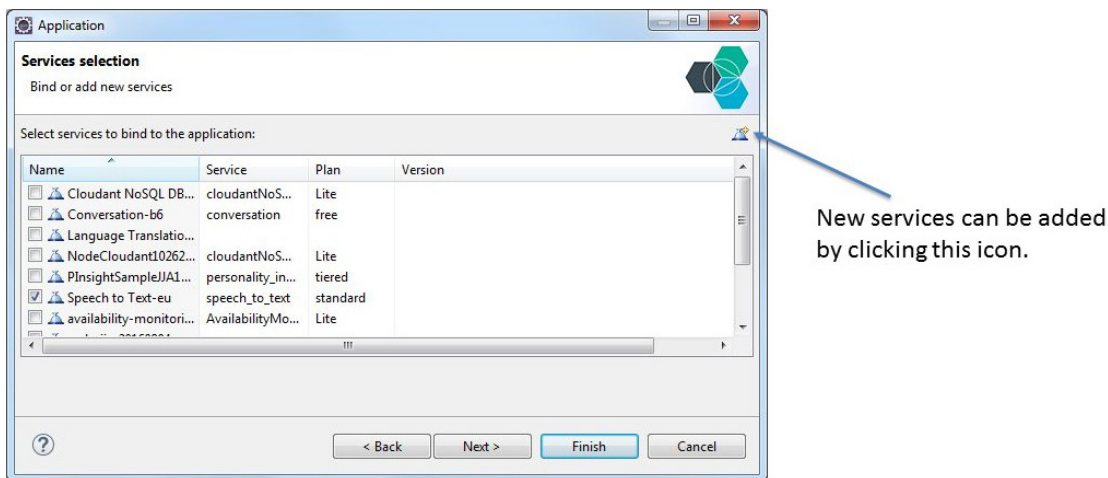
5. Bluemix also requires details about starting the application.



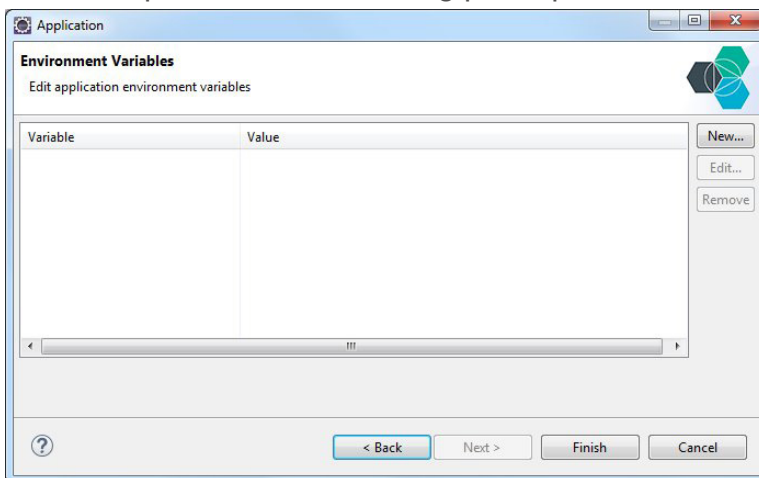
The screenshot shows a dialog box titled "Application" with a sub-header "Launch deployment". Below the sub-header is the instruction "Specify the deployment details." There are four input fields: "Subdomain:" with the value "xxxTranslationAppmmdyyyyy", "Domain:" with a dropdown menu showing "mybluemix.net", "Deployed URL:" with the value "xxxTranslationAppmmdyyyyy.mybluemix.net" and a "Validate" button, and "Memory Limit (MB):" with the value "512". Below these fields is a checked checkbox labeled "Start application on deployment". At the bottom of the dialog are four buttons: a help button with a question mark, "< Back", "Next >", and "Finish".

6. The next window binds the new cognitive services to the application. In this case, the Language Translator and Speech to Text services. Note that in addition to the

displayed services, the icon by the right border allows for new services to be added.

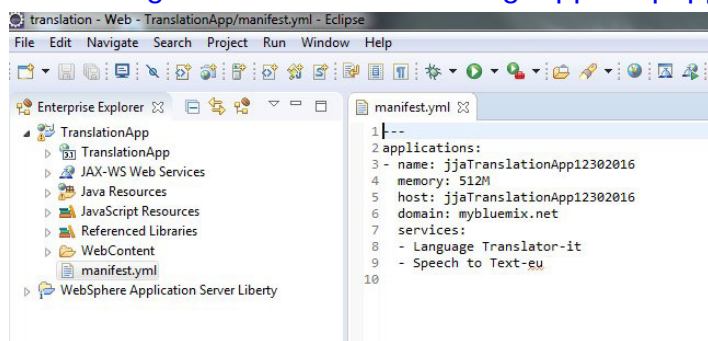


7. Environment variables are a useful way of passing information to an application. Bluemix, for example, stores the user name and password as environment variables, which makes for a more secure way of using these credentials. Although not used for this experiment the following panel provides for an easy way of identifying them.



8. Click **Finish** and the IBM Eclipse Tools for Bluemix pushes the application to Bluemix. This lets you stay within the Eclipse IDE and avoid having to transition to a Cloud Foundry session. Note the long time that it takes to deploy to Bluemix, which is a good reason to do most of your application testing using the local Application Server server.
9. The manifest.yml file describes the application. It contains options that the `cf push` command and the IBM Eclipse Tools for Bluemix use when pushing the application and supporting artifacts to Bluemix. See <https://>

console.ng.bluemix.net/docs/manageapps/depapps.html#appmanifest for more details.



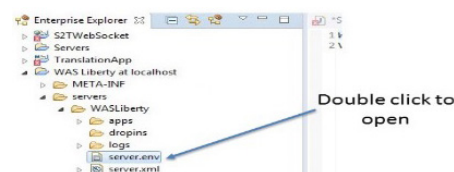
9. Review cognitive services credentials

Before running the application, you must ensure that the Watson Developer Cloud credentials match the credentials that are used by this experiment. Because the credentials are long, it would be too burdensome to ask the user to enter them in a window. For this application, the password and user name are retrieved as a JSON object in the `init` method of the `control` servlet. Take a look at the `init` method in the `TranslateServlet` source code.

Each service has its own unique credentials. You can identify what your credentials are by logging in to Bluemix after pushing your application to Bluemix and viewing it. Alternatively, you can use the Cloud Foundry method: `$ cf env <xxxTranslationAppmddyyyy>`.

To make the local Application Server instance as similar as possible to the Bluemix Liberty instance, you'll need to add an environmental variable to the local Application Server server.

1. Environmental variables are stored in the Application Server `server.env` file. Select the



instance of Application Server in the Project Explorer window.

2. Add the following line: `VCAP_SERVICES=`.
3. Use whatever method that you like (for example, copy and paste) to add to this line the text of the JSON object as a single line. For example, here is the `speech_to_text` JSON object from Bluemix:

```
{
  "VCAP_SERVICES": {
    "speech_to_text": [
      {
        "credentials": {
          "password": "password1234",
          "url": "https://stream.watsonplatform.net/speech-to-text/api",
          "username": "xxxxxxxx-7b7d-45c1-a44c-xxxxxxxxxxxxx"
        },
        "label": "speech_to_text",
        "name": "Speech to Text-eu",
        "plan": "standard",
        "provider": null,
        "syslog_drain_url": null,

```

```
    "tags": [
      "watson",
      "ibm_created",
      "ibm_dedicated_public"
    ]
  }
]
}
```

The `server.env` file does not span multiple lines, so you must change the previous code to a single line:

```
VCAP_SERVICES={ "speech_to_text": [{"credentials": {"password": "password1234", "url": "https://stream.watsonplatform.net/speech-to-text/api", "username": "xxxxxxx-7b7d-45c1-a44c-xxxxxxxxxxxx"}, "label": "speech_to_text", "name": "Speech to Text-eu", "plan": "standard", "provider": null, "syslog_drain_url": null, "tags": ["watson", "ibm_created", "ibm_dedicated_public"]}]}
```

The compressed file contains a text file with a JSON template for your use. Use the user name and password that is provided by Bluemix and substitute them for the defaults in the template.

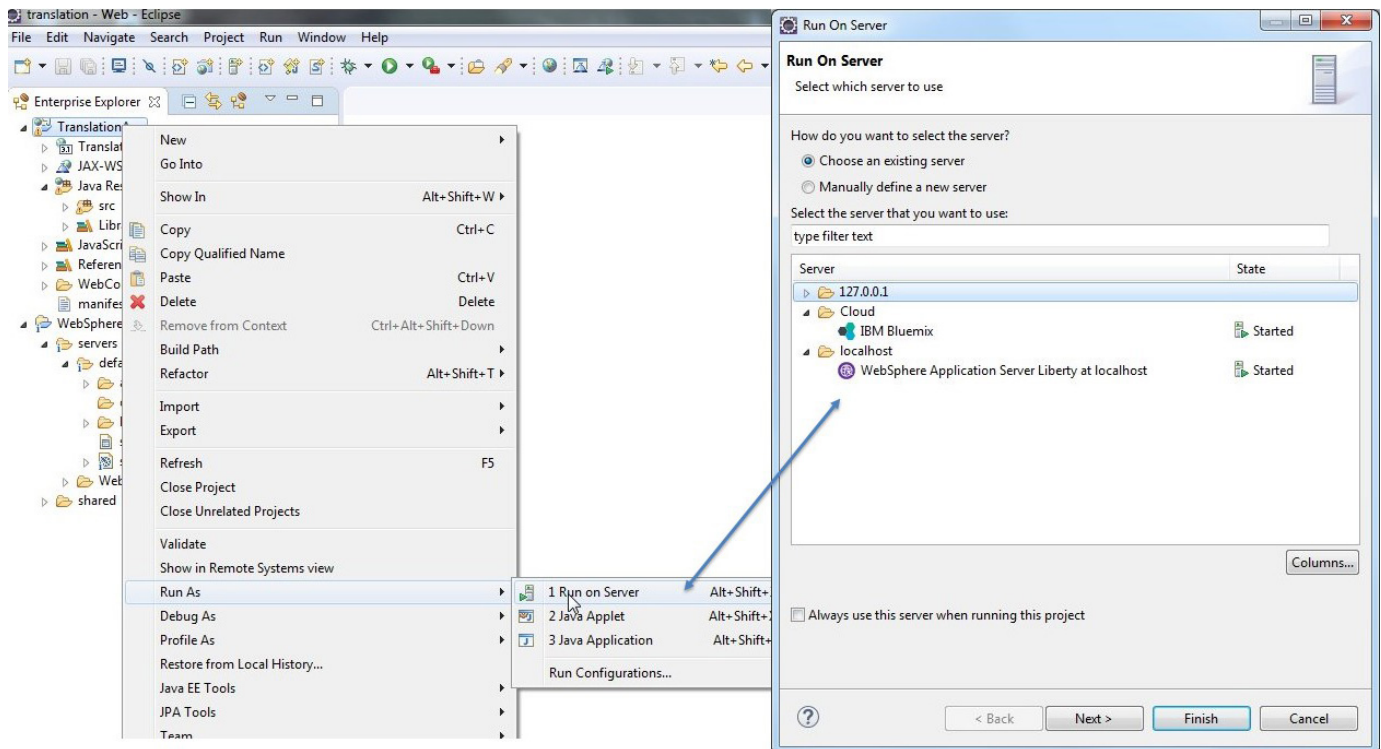
One important note about the `server.env` file. It is only loaded when the server starts. If you change the file, you must restart the server for the change to take effect.

10. Test the application

You must deploy the project to the local Application Server server to run it. The process essentially is the same for deploying the project to Bluemix.

1. Right-click the Application Server server in the Server window, and select **Add and Remove**.
2. Select **TranslationApp** in the Add and Remove window, and click **Add**, then **Finish**.

To run the application, select the TranslationApp project in the Project Explorer and select **Run As > Run on Server**. Use the `localhost` for testing before running the app in the Bluemix environment.



There are two things to note about the Speech to Text option:

1. The fidelity of the microphone is very important.
2. When attempting to use Speech to Text with the Bluemix server, you will get a Line not supported message. This message is expected and will be expanded upon in a future experiment using Web Sockets.

Conclusion

In this series and experiment, I have attempted to show how you can quickly incorporate cognitive services into your own projects. You can then expand upon this approach by improving the functions and incorporating additional services. You are embarking on a very interesting journey, and I hope this experiment helps. Enjoy the experience.

Related topics

- [Watson Language Translator](#)
- [Watson Speech to Text](#)
- [Bluemix](#)

© Copyright IBM Corporation 2017

(www.ibm.com/legal/copytrade.shtml)

[Trademarks](#)

(www.ibm.com/developerworks/ibm/trademarks/)