

Implementing a RESTful Service Objects Web Service into a Java Project

Introduction

This guide provides a step by step on how to implement a Service Objects DOTS web service into a simple web form using Java. This project will be using the REST implementation of FastTax but will be similar for any REST implementation of a Service Objects web service. This guide will be very basic and should help serve as a foundation to integrating one of Service Object's web services into your Java application.

What to Expect:

We will demonstrate the implementation in two parts. The first part will demonstrate the process of calling and displaying the results from a Service Objects web service and the second part will include a more applicable example of how to integrate some useful business logic around the validated data from the Service Objects web service.

Audience

Any developer (beginner to expert), that is interested in integrating a DOTS web service into Java web application.

Requirements

- Java IDE – *Eclipse was used in this example, but the steps should be relatively similar for another Java IDE*
- Basic Working Knowledge of Java and Object Oriented Programming
- A working License Key(trial or production) from Service Objects. [Click here](#) to obtain a free trial key for the service you are most interested in.

Contents

Setting Up Your Project.....	2
Designing Your Web Form and Adding the REST Call Code	4
Failover Configuration.....	6
Calling a Service Objects Web Service and Displaying the Results	7
Implementing Business Logic into Your Web Application.	9
Conclusion.....	12

Implementing a RESTful Service Objects Web Service into a Java Project

Setting Up Your Project

Launch Eclipse and select an appropriate workspace to keep your project. Select **File**, **New** and then **Dynamic Web Project** to create the web application. If the **Dynamic Web Project** option does not appear, then select **Other...** and search for it manually or ensure that the proper Java plug-ins are installed for Eclipse.

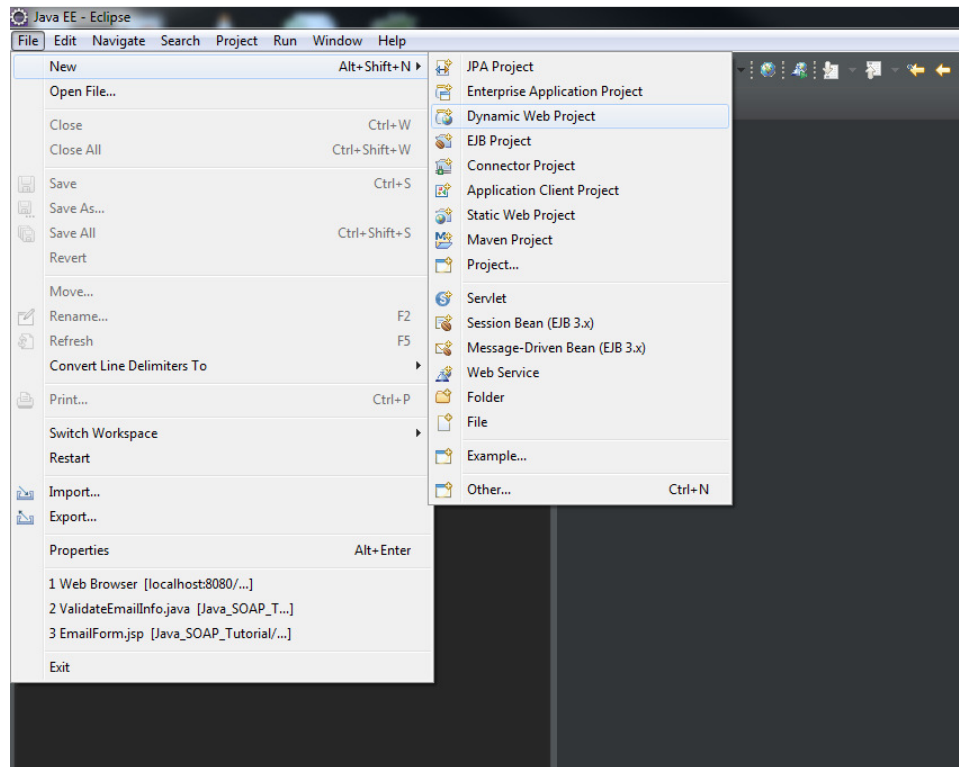


Figure 1

On the next screen, give the project a name and location. Select the appropriate runtime and configuration for your project. We'll be using Tomcat v7.0 to test our project in this tutorial. This screen can be seen in figure 2.

Implementing a RESTful Service Objects Web Service into a Java Project

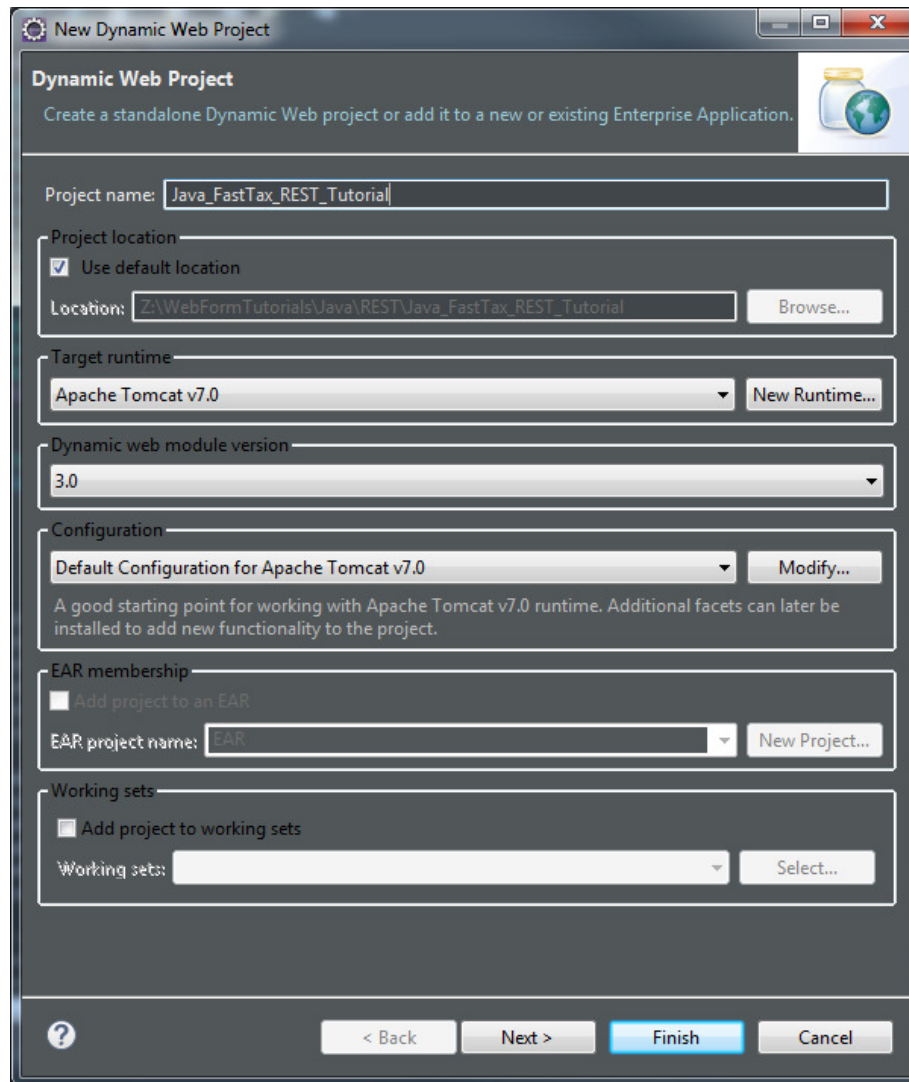


Figure 2

Now that the project has been created, we will need to add two JSP files to function as our input form and to handle the call to the Service Objects web service. To do this, right click the project folder, select **New** and then select **JSP File** and you will be prompted to name the files you are adding. For this example, we will name the files **OrderForm** which will serve as our input form, and **ProcessOrder** which will serve as the code that will call the FastTax web service and process the results.

Implementing a RESTful Service Objects Web Service into a Java Project

Designing Your Web Form and Adding the REST Call Code

Now that we have successfully created the web application and added the necessary pages, we need to fill in the input form. Our input form will be a very simple form with the elements for an input address and two buttons that we will use to call the web service and reset the form if need be. The necessary code will be in **OrderForm_Contents.txt** file that is included with this tutorial.

We have the a small table with our input fields for address, city, state, zip code, tax type and two buttons that will either submit the address to receive tax information for or reset the web form. An important thing to note is that the form action will dictate what page the form gets posted to, so be sure to have the appropriate page listed in the form action section.

Now we'll need to add the package and class that will make the actually call to our restful service. While in your project explorer navigate to the **Java Resources** folder and right click on **src** navigate down to **new** and then select **Package**. This can be seen in figure 3.

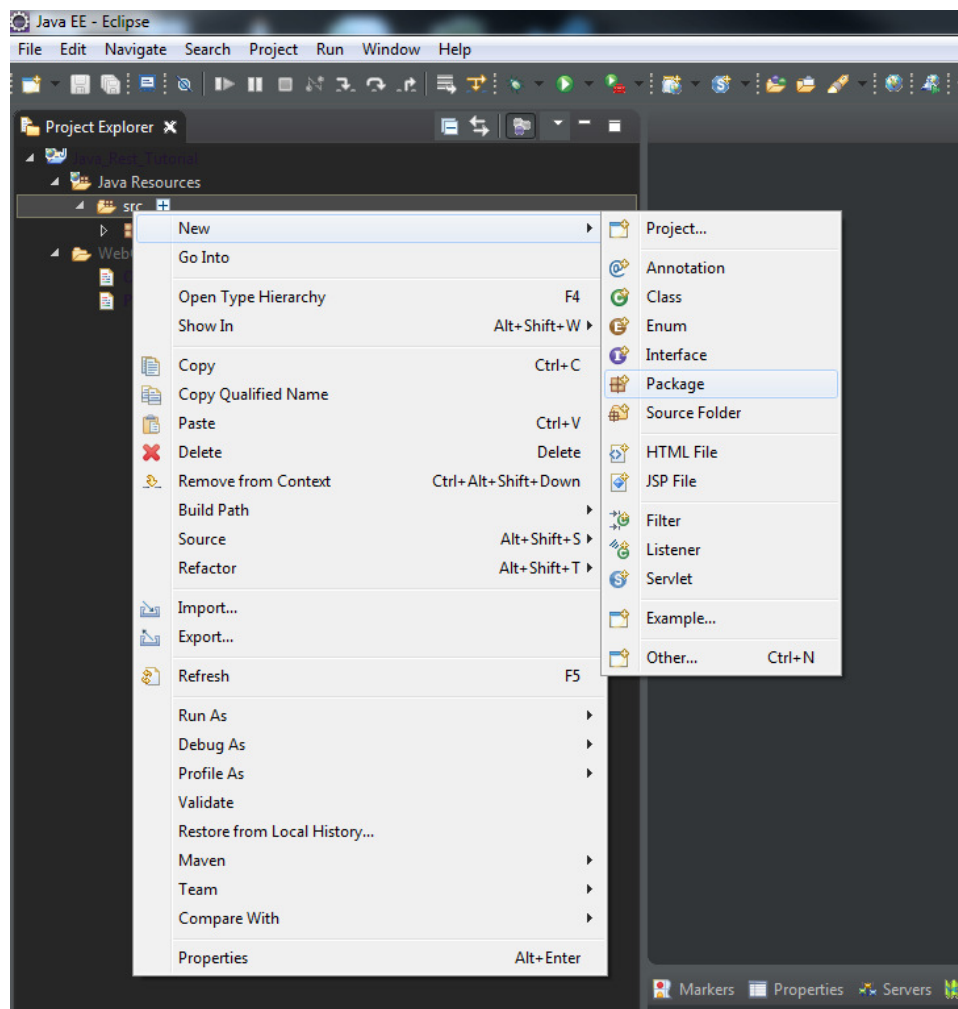


Figure 3

Implementing a RESTful Service Objects Web Service into a Java Project

A new dialog box will appear, give the package an appropriate name. For this example we will call the package **soRestClient**. After the package has been created right-click it, select **New** and then select **Class**. On the dialogue box that appears, give the class a proper name. For this example we'll name the class **RestClient**. After the class has been added, copy and paste the code from the **RestClient_Contents.txt** file.

```
URL url = null;
URLConnection conn = null;
JSONObject outputs = new JSONObject();

try {
    url = new URL(mainURL);

    conn = (URLConnection) url.openConnection();
    conn.setConnectTimeout(Timeout);
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept", "xml");

    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : "
            + conn.getResponseCode());
    }

    BufferedReader br = new BufferedReader(new InputStreamReader(
        (conn.getInputStream())));
    StringBuilder sb = new StringBuilder();
    String output;
    // System.out.println("info from the server \n");
    while ((output = br.readLine()) != null) {
        sb.append(output);
        //System.out.println(output);
    }

    outputs = XML.toJSONObject(sb.toString());

    if(outputs == null)
    { throw new Exception("ERROR: trying to get Response");}
```

Figure 4: Basic Rest call configuration

Now that this has been implemented, we'll touch of a few important elements of that the code utilizes to make the call to the Service Objects web service. The **RestClient** class has one method(**restCall**) that we will pass the values **address, city, state, zipcode** and **taxtype**. The license key value is hard coded right before we set the main and the backup URLs. Put your appropriate license key into this field. This code then forms the URLs based on the inputs received when the **restCall** method is called.

In this code both the primary and backup URLs are pointed to the same server, but once a production key is purchased there should be 2 service references added to the project: one for the primary Service Objects endpoint and one for the backup:

<https://ws.serviceobjects.com/ft/FastTax.asmx>

<https://wsbackup.serviceobjects.com/ft/FastTax.asmx>

Implementing a RESTful Service Objects Web Service into a Java Project

Failover Configuration

Within the code that makes the call to the web service, we've included failover configuration. In the event that you make your own REST call, we highly recommend implementing a failover call into your code. This is done by checking if the returned object is null or has an error with a **Number** of 3; which would indicate a fatal error. In the event that either of these instances occurs, the code will call the backup service. We strongly recommend implementing this fail over configuration into your project. With failover enabled you can ensure that your application will function as normal in the event that the production endpoint is unavailable.

```
if (results == null || (results.getJSONObject("TaxInfo").has("Error") && results.getJSONObject("TaxInfo").getJSONObject("Error").get("Number") == "3"))  
{  
    // BACKUP  
    results = callSOService(backupURL);  
}
```

Figure 5: Failover Configuration

As noted in the screen shot above, if the code detects a null response or **Number** of 3 it will call the wsbackup endpoint and attempt to get a valid response. After this, the code checks if there was an error response returned; if so, then it will call the **CallSOService** method. If no error response is detected then the code will return the **results** object back to the **ProcessOrder.jsp** page.

Implementing a RESTful Service Objects Web Service into a Java Project

Calling a Service Objects Web Service and Displaying the Results

This section will illustrate how to make a call to the web service using the class we created and then display the results from the service to user. If you are interested in seeing possible applications or business logic that Fast Tax service may be used for, skip to the next section as it will present some possible ways that the Fast Tax service can be used in your application.

The contents for the *ProcessOrder.jsp* page are contained in the *ProcessOrder_Contents.txt* file that was included with this tutorial. Copy and paste it into the page that will be processing the necessary information.

The code will first pull the inputs from the *OrderForm.jsp* page, and then encode the strings so that they are acceptable for the HTTP GET call. Then the *RestClient* method and *results* objects are instantiated and the *restCall* method is called with the user inputted address as the parameters.

```
String address = request.getParameter("address");
String city = request.getParameter("city");
String state = request.getParameter("state");
String zipcode = request.getParameter("zipcode");
String taxtype = request.getParameter("taxtype");

address = URLEncoder.encode(address,"UTF-8").replaceAll("\\+", "%20");
city = URLEncoder.encode(city,"UTF-8").replaceAll("\\+", "%20");
state = URLEncoder.encode(state,"UTF-8").replaceAll("\\+", "%20");
zipcode = URLEncoder.encode(zipcode,"UTF-8").replaceAll("\\+", "%20");
taxtype = URLEncoder.encode(taxtype,"UTF-8").replaceAll("\\+", "%20");

//Creates the response objects and calls the service
RestClient rest = new RestClient();
JSONObject results = new JSONObject();

results = rest.restCall(address, city, state, zipcode, taxtype);
```

Figure 6: Parameter Cleanup and call to Rest package

After the call gets made to the service, the *if* statement will check for an error response; if one is found, then the form will output the error that was received as shown in the figure below.

Desc:	Please specify either 'sales' or 'use' tax.
Number:	2
<input type="button" value="Start Over"/>	

Figure 7: Error Response Example

If the user input is valid, the project will output all the results from the service. Below is a screen shot illustrating a valid response from the FastTax web service.

Implementing a RESTful Service Objects Web Service into a Java Project

Zip:	93101-7602
City:	SANTA BARBARA
County:	SANTA BARBARA
CountyFIPS:	83
StateName:	CALIFORNIA
StateAbbreviation:	CA
TotalTaxRate:	0.08
TotalTaxExempt:	LABOR/FREIGHT/SERVICES
StateRate:	0.065
CityRate:	0
CountyRate:	0.01
CountyDistrictRate:	0.005
CityDistrictRate:	0
<input type="button" value="Start Over"/>	

Figure 8: Valid Response Example

This code can be used for testing purposes to experiment with the different types of results that service will output depending on what the user enters.

Implementing a RESTful Service Objects Web Service into a Java Project

Implementing Business Logic into Your Web Application.

Simply displaying the results to the end user isn't a necessarily a good use of the Service Objects web service, so in this section we will go over an example of what it may look like to integrate some business logic around the results that web service returns. For this example we'll assume that a user is choosing some items to order and that the user is entering shipping information. The application will take the user entered shipping address, get the tax rate based on the address and then output total amount with tax included that the user will then pay for. Along with this logic, we'll integrate some logic that will display error responses based on potential error messages from the service.

To implement this functionality, we'll need to update the code in our **OrderForm.jsp** page. The new code contents for this page can be found below in the **Updated_OrderForm_Contents.txt** file with this tutorial.

A few changes have been made to this page to accommodate our new functionality. The first is that we have added some checkbox items to enable the user to select the items that they wish to purchase. If an item is chosen, then we will see it and process it in the updated **ProcessOrder.jsp** page. The other change we have made to this page is to remove the tax type input. We will hard code this value into our program so that the user won't have to encounter it.

```
<h2>Sample Purchase Form.</h2><br/>
<form action = "ProcessOrder.jsp" method="post">
  <br/><h4>Select Items to Purchase</h4>
  <input type="checkbox" name="purchaseItem" value="widget">Widget - $28.58<br/>
  <input type="checkbox" name="purchaseItem" value="gizmo">Gizmo - $53.99<br/>
  <input type="checkbox" name="purchaseItem" value="whatchaCallIt">WhatchaCallIt - $18.00 <br/>
```

Figure 9: New CheckBox addition to OrderForm.jsp

Now that our input form is set up we'll update the ProcessOrder.jsp code so that it will have the upgraded functionality. The updated code for the **ProcessOrder.jsp** page can be found in the **Updated_ProcessOrder_Contents.txt**

The project will have the same overall structure of cleaning up input variables, making the REST call, and then checking for an error response or for a valid response from the service. We have also instantiated a new array of strings that will hold all the order items that the user will choose. If no item is chosen by the user, it will remain null. Our code will later check for this value to ensure that it functions correctly. We have also hard coded the value "sales" into the tax type parameter that we pass the FastTax service. Depending on your application it may be useful to implement some logic into your program to determine whether or not "sales" or "use" tax is needed.

Within the section of code that checks for an error response, there are 2 different error messages that could be displayed to the user. The code first checks for a specific **Desc** value from the error response. If the value is equal to "Address was not found", then code will display a message to the user indicating that the user entered address was not valid. If this is this case, the user should be prompted to enter a new address to verify the tax rate. A screen shot of this message can be seen below.

Implementing a RESTful Service Objects Web Service into a Java Project



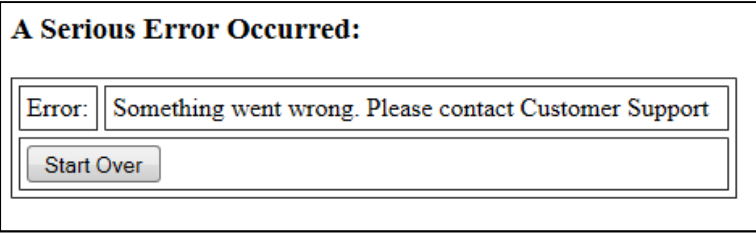
An Error Occurred:

Error: The entered address was not found. Please double check the input or enter a new address.

Start Over

Figure 10

If there was another error message, this may indicate that there was a problem with the call to the web service that may be out of the user's control. These possible scenarios could range from not entering the correct tax type, not having enough transactions on your license key, or a fatal error when connecting to the web service. If any of these events shows up, it will display a message urging the user to contact customer support. It may also be wise to log the error on your end in the event that you will need technical support from Service Objects. Below is an example of that error message



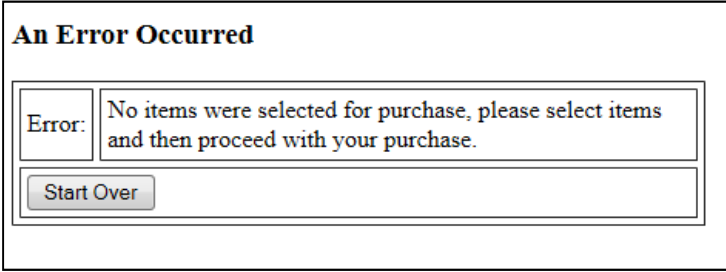
A Serious Error Occurred:

Error: Something went wrong. Please contact Customer Support

Start Over

Figure 11

The next check the code will make is to see if the user selected an item to purchase. If no item was selected, then the user will be given a message to select items to purchase. Below is an example of this message



An Error Occurred

Error: No items were selected for purchase, please select items and then proceed with your purchase.

Start Over

Figure 12

Finally, if a user selects items to purchase and enters correct address information, a form will appear and display information about the order that the customer is going to complete. This form will list the items that the user will purchase the subtotal of all the items, the tax rate, and tax to be collected, and the total amount due at check out for the order. A button will appear below this information for the user to continue with their purchase. This button will reset the page back to the **OrderForm.jsp** page. Below is a screen shot for this completed order form.

Implementing a RESTful Service Objects Web Service into a Java Project

Order Summary:

Items in Your Cart

Item:	widget
Item:	gizmo
Item:	whatchaCallIt

Purchase Sub Total	\$100.57
Tax To Be Collected: (0.08):	\$8.05
Total Due at Check Out	\$108.62

Continue With Purchase

Figure 13

Obviously this project is nowhere near a finished product but hopefully it provided a basic outline for how to use a Service Objects web service.

Implementing a RESTful Service Objects Web Service into a Java Project

Conclusion

That concludes our tutorial on how to create a web form that uses a RESTful call in Java. If you have any questions please email support@serviceobjects.com and we would gladly address any issues you may have.