Implementing a 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 SOAP implementation of Email Validation 3 but will be similar for any SOAP 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

Implementing a 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.

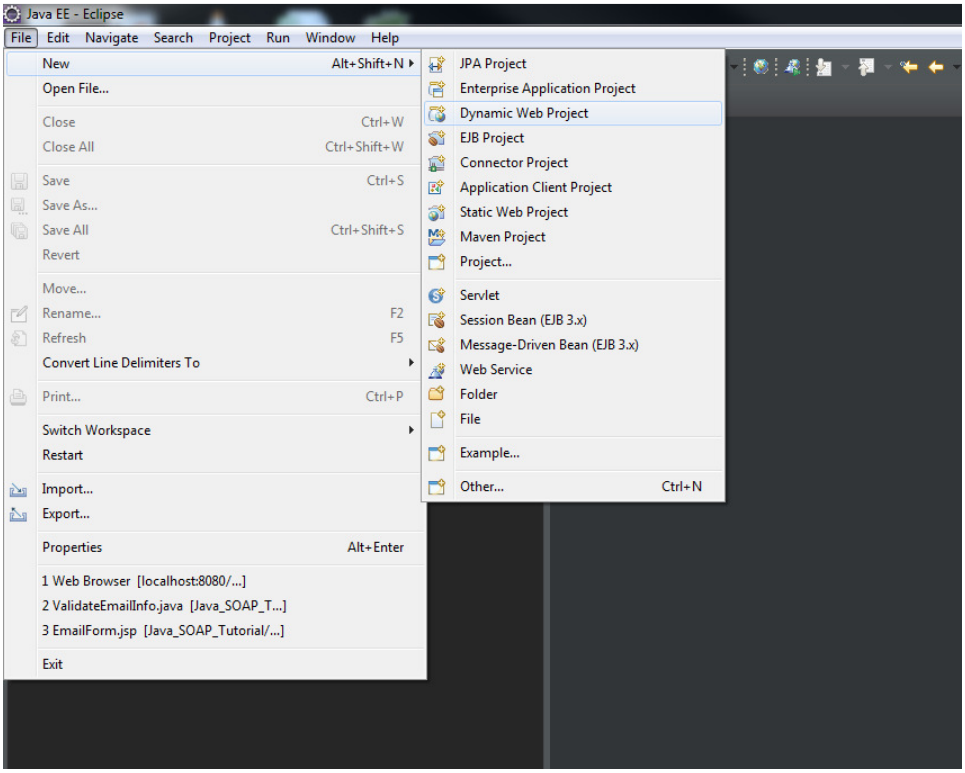

Figure 1

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 Service Objects Web Service into a Java Project

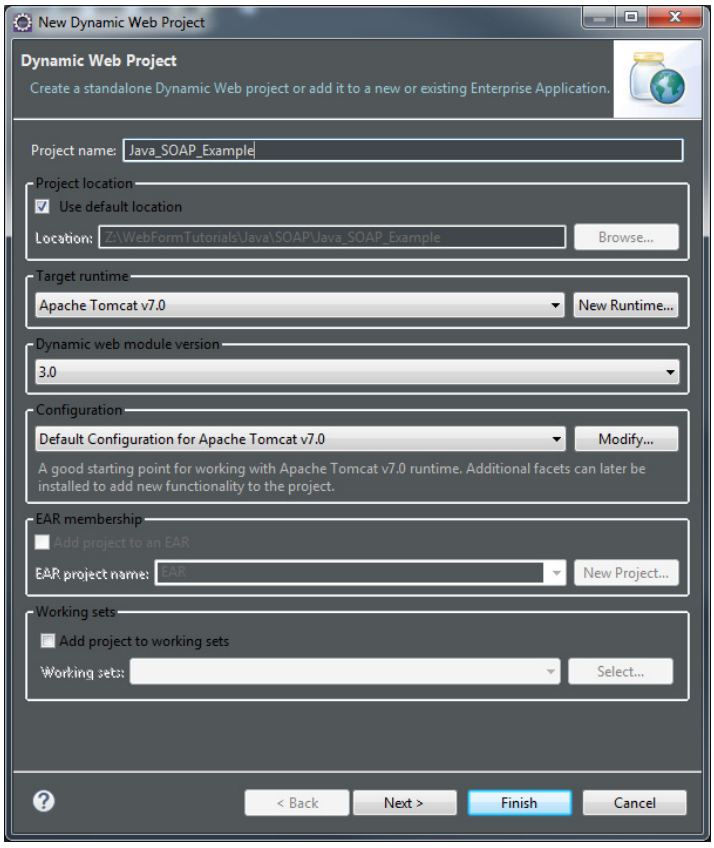

Figure 2

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 ***EmailForm*** which will serve as our input form, and ***ev3soap*** which will serve as the code that will call the Email Validation 3 web service and process the results.

Implementing a Service Objects Web Service into a Java Project

Designing Your Web Form and Adding a Web Service Client

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 an email field 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 code in the ***EmailForm_Contents.txt*** file that is included with this tutorial.

```
<body>
        <h2>Example Sign Up Form</h2>
        <form action ="ev3soap.jsp" method="post">
            <table width="25%" border="0" cellspacing="1" cellpadding="3" >
                <tr >
                    <td>Email Address</td>
                    <td><input type="text" name="email" value="" size="25"></td>
                </tr>

                <tr >
                    <td colspan="2" width="100%">
                        <input type="submit" value="Sign Up" style="margin-right:10px;margin-left:20px;">
                        <input type ="reset" value="Start Over">
                    </td>
                </tr>
            </table>
        </form>
</body>
```

Figure 3: Basic Sign-Up Form

We have the a small table with our input field for an email address and two buttons that will either submit the email to be validated or reset the web form. An import 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 a web service client so we can properly access and utilize the Service Objects web service. Now, we need to add a Web Service Client for this project. To do this, right click the project folder, select and select ***Other***. In the text box search for "Web Service Client" select that option and click next. On the next screen, paste in the WSDL for the web service you are using and move the slider down to "Develop Client" as shown in figure 3.

Implementing a Service Objects Web Service into a Java Project

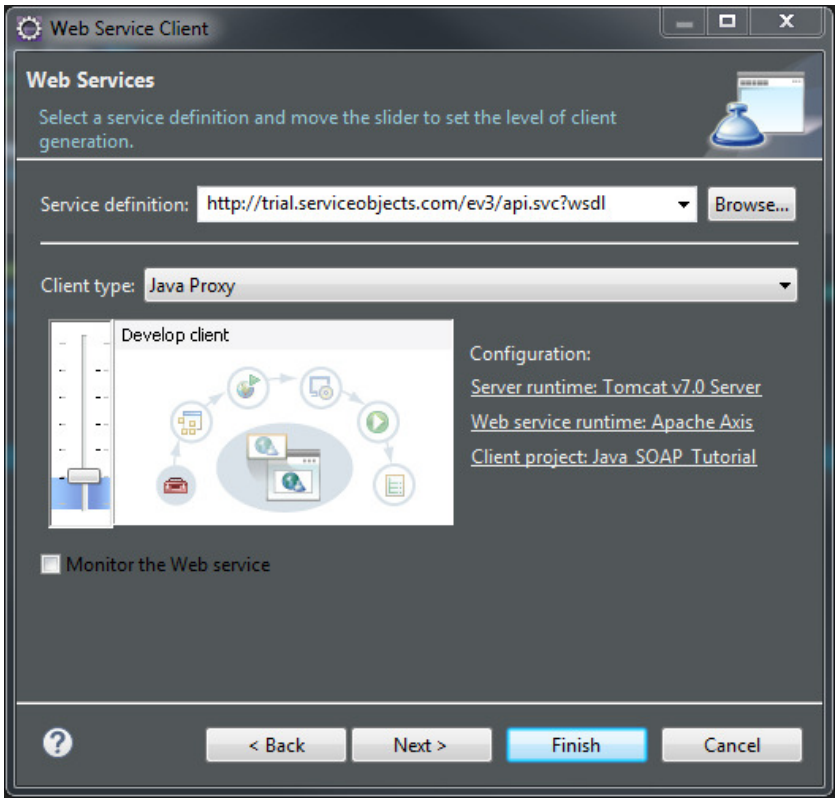

Figure 4: Adding a Web Service

The new class definitions that were created will be located in the Java Resources/src/com.serviceobjects.www folder.

In this code both the primary and backup are created from the same service reference 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/ev3/api.svc?wsdl

http://wsbackup.serviceobjects.com/ev3/api.svc?wsdl

Implementing a Service Objects Web Service into a Java Project

Calling a Service Objects Web Service

Now that we have all the appropriate project files created, we can add the code to our ***ev3soap.jsp*** page that will make the actual call to our web services. The code for this page can be found in the ***ev3soap_Contents.txt*** file that has been included with this tutorial.

```
//Initializes the output response and error response objects
ValidateEmailResponse outputs = null;
com.serviceobjects.www.Error ev3error = null;

//This code pulls the user entered email address from the input form and initializes our LicenseKey string
String email = request.getParameter("email");
String LicenseKey = "";

EV3LibraryLocator locator = new EV3LibraryLocator();
locator.setDOTSEmailValidation3EndpointAddress("https://trial.serviceobjects.com/ev3/api.svc/soap");

IEV3Library ev3 = locator.getDOTSEmailValidation3();
DOTSEmailValidation3Stub soapcall = (DOTSEmailValidation3Stub)ev3;
soapcall.setTimeout(5000);

outputs = soapcall.validateEmailFull(email, LicenseKey);
ev3error = outputs.getError();
```

Figure 5: Passing parameters to the EV3 Soap Service

After this code has been pasted into your project it should operate correctly and provide the basic functionality of displaying the results to the user. We'll go over the general flow of the application and highlight some important items of the project.

The first part of the code initializes the objects that will hold our valid response and error response from the Email Validation 3 web service. Next, the code pulls the user inputted email address and sets it to a local string. The code then goes through all the necessary steps on instantiating a call to the service objects web service. Another thing to note about this is that user must manually set the endpoint address as shown in the screen shot in figure 6.

```
EV3LibraryLocator locator = new EV3LibraryLocator();
locator.setDOTSEmailValidation3EndpointAddress("https://trial.serviceobjects.com/ev3/api.svc/soap");
```

Figure 6

This endpoint address will have to set so that it matches the endpoint address defined in the WSDL that we initially used in creating the Web Service Client for this project. This can typically be found at the bottom of the WSDL document within the soap:address tag.

Implementing a Service Objects Web Service into a Java Project

Failover Configuration

Now that we have the code to call the Email Validation 3 service, we need some code that will check and process the response from the web service. 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 TypeCode 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(outputs == null || (ev3error != null && ev3error.getTypeCode()=="3"))
{
    //In the event that a fatal error is found in the code above, the code will call the backup endpoint.
    //Be sure to instantiate the backup endpoint to the wsbackup.serviceobjects.com point.
    outputs = soapcall.validateEmailFull(email, LicenseKey);
}
```

Figure 7: Failover Configuration

As noted in the screen shot above, if the code detects a null response or TypeCode 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 ValidateEmailFull method once again. If no error response is detected then the code will move onto the rest of the project.

Implementing a Service Objects Web Service into a Java Project

Display Results to the User

This section will illustrate how to simply display the results from the service to user. If you are interested in seeing possible applications or business logic that Email Validation 3 may be used for, skip to the next section as it will present some possible ways that the EV3 service can be used in your application.

Now, our code will check what kind of response the service has returned and resulting information that the service has returned. In each of these instances, the code will create a table to display the resulting information. Below is an example of a valid response.

Email Validation 3 Results

Score:	0
IsDeliverable:	true
EmailAddressIn:	support@serviceobjects.com
EmailAddressOut:	support@serviceobjects.com
EmailCorrected:	false
Box:	support
Domain:	serviceobjects.com
TopLevelDomain:	.com
TopLevelDomainDescription:	commercial
IsSMTPServerGood:	true
IsCatchAllDomain:	false
IsSMTPMailBoxGood:	true
Warning Codes:	
Warning Descriptions:	
Note Codes:	3,4
NotesDescriptions:	Role,BusinessAddress

Start Over

Figure 8

In the example above we pass the service a good email address (Service Objects' technical support email) and the service returns some valuable information about this email address. Most notably, the Score that the email has received is a score of 0. In the next section of this tutorial, we will implement some logic into the application that will allow us to decide if an email is valid or not.

The code will also display an error message if one is returned from the service. An example of this can be seen in figure 7 below.

Implementing a Service Objects Web Service into a Java Project

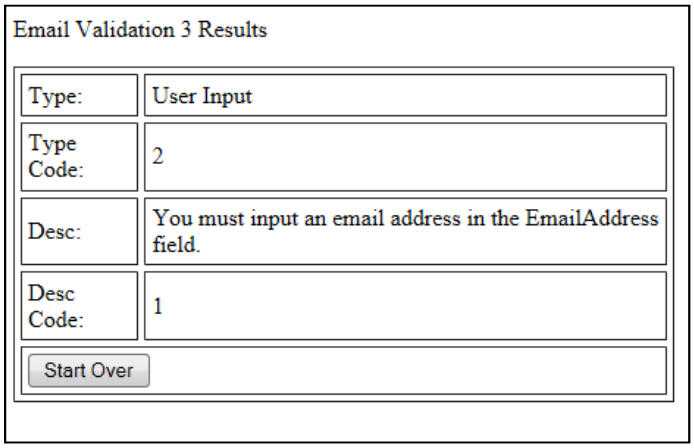

Figure 9

This code is now ready to run. 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 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 example, let's presume that a user will be entering an email address to sign up for an online account and access further services. Upon the user signing up, we can run a quick test against the email validation service and use the results to determine if the user entered email is valid.

In this section we will change the code that is implemented after our call to the service objects web service. The updated code is available in the ***Updated_ev3soap_Contents.txt*** file that is included with this tutorial. Copy and paste this code into the ***ev3soap.jsp*** and we'll highlight some key aspects about this new logic we have implemented.

Our updated code has logic surrounding the ***Score*** value that is returned from the Email Validation 3 service. The service will perform real time tests on the email that the user passes to it and give it a corresponding score based on the results from the tests are performed. Using the ***Score*** that is returned, we can make an informed decision about the validity of the email address.

In Figure 8 below, we see that if an email receives a ***Score*** of 0, 1 or 2, then it we will determine that the email is valid and let the user continue with their sign-up process.

```
if(ev3error == null)
{
    ValidateEmailInfo emailinfo = outputs.getValidateEmailInfo();

    //In this example the threshold for a good email address with be scores from 0 to 2.
    if(emailinfo.getScore() == 0 || emailinfo.getScore() == 1 || emailinfo.getScore() == 2)
    {
    %>
    <h2>Success!</h2>
    <table border='1' width="450" cellspacing="4" cellpadding="4" >
        <tr>
            <td>"<%=emailinfo.getEmailAddressIn()%>" Was Accepted as a Valid Email Address</td>
            <td>Please Continue With the Sign-Up Process</td>
        </tr>
        <tr >
            <td colspan="2" >
                <input type ="button" value="Continue" onClick="document.location='EmailForm.jsp';">
            </td>
        </tr>
    </table>
    <%
    }
```

Figure 10

In this code an email with a Score of 2 is considered valid. The score of 2 means that it is unknown whether or not that the email in question is valid. If you would like the logic to be more strict, then you could only pass an email with score of 0 or 1. This may result in good emails being flagged as bad but it really depends on the desired business logic.

If a user's email address is deemed to be valid, then the code will display a message saying that the email was determined to be valid and that they can proceed with the sign up process. Below is an example of that functionality.

Implementing a Service Objects Web Service into a Java Project

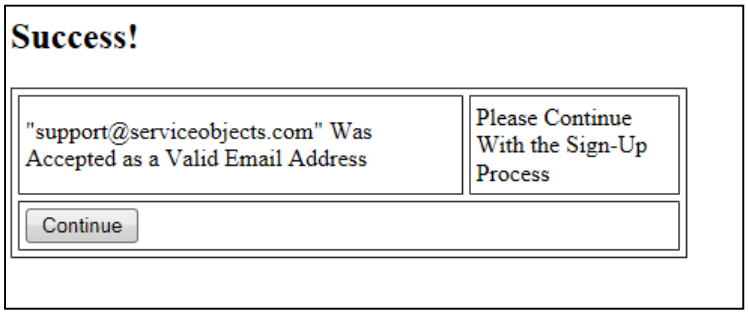

Figure 11

If an email is determined to not be valid, then user will see the below message indicating that there was a problem when trying to validate the email address.

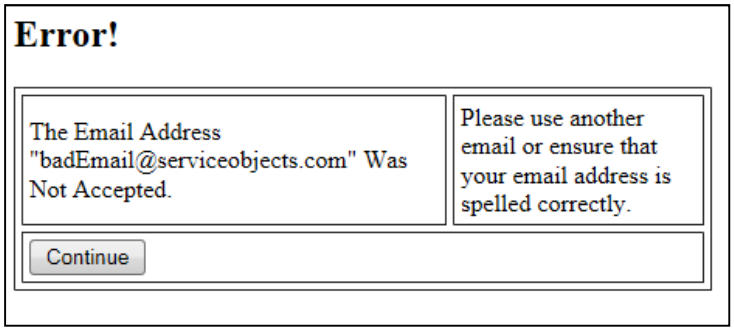

Figure 12

The updated code is also set up to display information to the user in the event that an error is returned from the service. Currently it is set up with a simple if else statement. A screen shot of the code can be seen below.

```
else if(ev3error.getTypeCode().equals("2") && ev3error.getDescCode().equals("1"))
//If the DescCode is equal to one, then this will indicate that an email address was not entered.
{
%>
    <h3>Error</h3>
    <table border='1' width="450" cellspacing="4" cellpadding="4" >
        <tr>
            <td>An Error has a occurred.</td>
            <td>Please Enter a Email Address</td>
        </tr>
        <tr >
            <td colspan="2" >
                <input type ="button" value="Start Over" onClick="document.location='EmailForm.jsp';">
            </td>
        </tr>
    </table>
<%
}
```

Figure 13

Implementing a Service Objects Web Service into a Java Project

If the service is not passed an email address, then it will give an error response. This response with have a ***TypeCode*** of 2 and a ***DescCode*** of 1. In this event, the user will be notified that the email address field cannot be blank and they will be asked to enter an email address again. Below is a screen shot of the message that is returned.

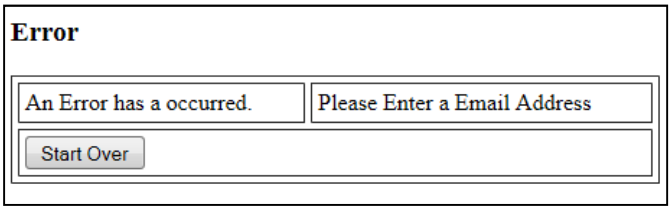

Figure 14

If another type of error is returned, this would likely indicate a larger problem is wrong with the call to the web service or with the Service Objects web service. In this event, the user will be given a message to contact customer support. This information can also be logged so that it can be referenced later to determine the issue. Below is an example of this.

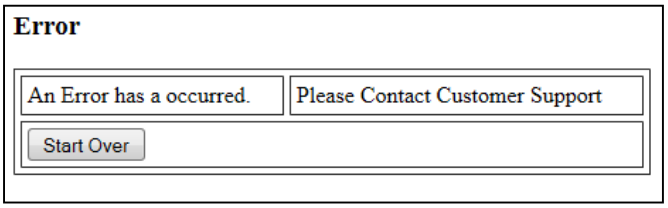

Figure 15

Implementing a Service Objects Web Service into a Java Project

Conclusion

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