

Welcome to the EGL Consuming Hats demo. In this demo we are using WSDC and HATS, but we could also use RDI SOA to accomplish the same end results.

We have already showed you in the previous part of this demo how to create a web service with the HATS tool. Here we take the HATS wsdl that HATS created and copy it into an EGL project and then create an application that accesses the web service with the information that is inside the wsdl file.

Here we are looking at the workbench and I am in the web perspective. We'll create a new EGL project and we can go ahead and copy the wsdl into that project.

We select the dynamic web project action from the pop up menu. The new dynamic project wizard will prompt us for the project name and configuration we want to use and the file name. We fill in this information and then go to the next page.

On this page we select some capabilities we need in our project so the wizard can generate the right files and directories in our web project. After we made our selections, we press the next button to get to the next page.

This page has all the defaults we need so we don't have to change anything, just click next. For this project we don't need to specify any additional EGL settings. So we can go ahead and finish up using the wizard. The wizard creates the project with all the folders, files, and directories that we need to make this a real J2EE web project.

Here we can see that our project was created in our project view. We now expand the GetCustInfo project to work with the individual files. You'll notice that inside the project there is an EGL source folder. Inside the EGL folder are all the packages that contain EGL logic, EGL data descriptions and EGL service descriptions. Inside this folder we will create a package and inside that package, we will copy the wsdl file.

Right mouse click on the EGL source folder and then select create a new package, which will then invoke the new package wizard. The new package wizard prompts us for the package name. We key in services and click the finish button. We already got the wsdl file that we received from the HATS folks and we now have to paste it into the services package. We expanded the services package and can now see the wsdl file inside this package.

Now we want to start the wizard that creates the interface from the wsdl file. So EGL knows about the service and knows about the service interfaces that users can create all the artifacts that are needed for EGL to invoke this service. We create EGL interfaces and Web Client Binding is the action we want to take here. The wizard opens and asks us which web services we want to add to this project since we only have one, we select the service that is listed here. Now we get a list of services that are listed in the wsdl file. There are two files getcustomerdetailsprocessws and getnestedbeannames. We are only interested in getcustomerdetailsprocessws. Click the next button.

We now have to identify the EGL descriptive file that will store the information that will link the wsdl file and the interface that gets created together. It has the binding information about this web service. Since we don't have a deployment descriptive file, we will just create a new one. Specify the name of the deployment descriptive file and click the finish button.

The wizard has opened up the EGL source file that got created from the wsdl information. I've highlighted the 2 things that are important. At the top we see the input parameters that this web service expects and that is the customer number which is a string, so we will give the service the customer number. On the bottom we see the output properties or output parameters. There is a lot of internal parameters in there, but we don't have to care about these. What we care about is the customer name, customer number and the customer phone fields. Later on in our user interface, we will take these fields and display the data in these fields from the user interface.

At the bottom of this source file you see the two functions that represent the services that are described in the wsdl file and since we are only interested in the `getcustomerdetailsprocessws` service, we will look at that and we can see that there is an input parameter specified and that input parameter has the data type of the record that I just showed you and was specified on top of this source file. There is also further on the right side an output parameter that describes the output record. That I have shown you as well that contains the customer name. We don't have to do anything with the source file, it is just for our information so that we know what got generated so we can use these pieces later on in our logic that invokes that interface.

We now want to create a web page so we right mouse click on the web content folder and we'll specify that we want to create a new web page inside this folder.

We will right click on the web content folder to select the new page wizard. We get prompted for the web page name and also for any templates we want to use on this page. We select the template and click the finish button to create the web page.

We now see the generated web page in the editor. The default editor of web pages is the page designer, which allows us to create the web page without knowing about html tags, it will create the html tags for us. We are not interested here in creating the page at the moment. We want to create the logic behind the page first. So we right mouse click on the page and say we want to go the jsf handler which contains the logic behind this page.

Here we can see the jsf handler for this page. There is already skeleton code for us created and we now have to go in and add the variables and the logic we need in our code. Here we create the variable `custnumber` which is of type string. Now we will create a stereotype that specifies what the heading should be on the user interface for this variable. Since I don't know all the attributes that are available for me for this data type, I just press `ctrl space` which will prompt me with everything that is available that starts with the letter `d`, which I already typed in.

The content assist tells me now that the three attributes that I could select DateFormat, displayname and displayuse. I need the display name, so I double click on it and it will fill the editor with that name. Displayname= was already filled in for me so I just had to add the string I want to add as a heading for this field which is enter customer number.

Now we need to add the parameters for the input parameter and output parameter. I'm using the content assist here again. So I specify the name for the record and since I know that the parameter records in the service starting with g, I already put in the letter g and I will start with the content assist by pressing the ctrl space.

Content assist shows me 3 values that I can select from, all of them are actually defined in the service. I'm interested in the input record. I'll double click on it and that value gets added to my variable definition.

I do the same for the output parameter record. Again, I am using content assist and here you can see handy content assist comes in. You don't have to memorize the names that have been created somewhere else, you just use content assist. You just need to know what letter they start with and you are pretty well off in writing this code very fast.

I select the record for the out parameter. Now I created the function that will invoke the web service. When the push button on the page gets pressed.

In this function, I need to fill the customer number value that gets passed to the web service, so what I do here is write an assignment statement that assigns the value that I get from the web page to the customer number field inside the input parameter record.

Again, using content assist to get the name of the parameter or better the name of the field inside the record. Now I have our assignment statement finished and I can now invoke the web service.

I create a variable that has the name request detail that is of the type web service. I use content assist to get the name of the web service. The name of the web service is GetCustomerDetails. Content Assist selects that. I add it to my variable definition. I have to specify that this is a service so I use content assist again to add this attribute. Bind service is the property name, so I select that. EGL as well as many other languages requires a semi colon at the end of the statement.

Now I invoke the service and I need to specify the service inside the interface I want to invoke. There are 2 functions available and I need to select which one I want and content assist will help me. Content assist will show me the 2 services available. I'll select the getcustomerdetails and it already includes the 2 parameters already required for it. Just select that and it gets inserted. The statement has been completed and now we need to adjust the parameter names. We have to use the names we created in the source.

We add an if statement here and check if the customer name gets returned as blank if the customer name is blank then we know that the customer number is not in the database.

So then we display an error message and we coded the error message here customernot found. That completes the function.

We now go back to the page designer and create the user interface. In the interface we want to add the customer number entry field to the page. To do that we can actually use the definitions we have created in the jsf handler. So if we look at the page data view on the lower left side of the work bench, we see the page handler for that. We need to expand it and then we can get access to variables and functions that are defined in the jsf handler.

Now the jsf handler is expanded, we can see the customer number defined as string and we can also see our input parameter and our output parameter records. We can now grab the customer number and drag it onto the page.

When we grab variables from the page view onto the page designer, we'll get a confirmation dialog that will ask us about the type of field we want to add. For example, is this an input field or only an output field. We can now make these selections and click OK. The page designer will now actually position that field on the page.

Here we can see the results of our action. We now have our entry field on the page. We have our heading we specified in the EGL jsf handler, everything is here. We can go on to the push button that invokes the function to this page.

We go back to the page data view, and now we get the function we created getcustomerinfo and drag that onto the page. You can see we now have a push button on our page that can automatically invoke that function and the web service to get the customer information. We want to add a horizontal ruler to our page, so we go into the html page drawer on the pallets view on the right side of the work bench, grab the horizontal ruler and drag it onto the page.

Now we have a ruler on the page and we can go ahead and insert the output data onto the page. We go to the table control from the html pallet and drag that onto the page so that we can put the output data on the headings into this table to align them nicely.

Now we are ready to fill the table with the variables that we defined in our jsf handler so we can drag those variables into the table and put the headings in and we are done.

Again, the dialog will come up as we drag those variables onto the page for an input field or output field. We can also change the labels if we want.

Here is the result of our actions of our table filled with the input field as well as the heading. We now have to adjust this and we do that for the 3 fields we want to get out of the service and out of the interface. I won't go through all these steps, I'll just show you the end result.

Here is the completed table, we have our fields in here and we can now save the page and EGL will generate code that is needed and we can test our application. We save the JSP and go ahead and run it. We right mouse click on the jsp file and we select run on server.

Our web page will come up and we have our header there, we have our input fields, the push button is there and we have our table that contains the data that is coming back from the service. I keyed in a valid customer number, then press the push button and we should get some results back. And here we are, we invoked the web service that was created by the HATS tool and that invoked the 5250 application that went and grabbed the data from the database. There is RPG Cobol behind that application. We turned that into a record and HATS grabbed the data from this record and put it back into the web service as part of the output parameters and now we can show the data on our web page here. So that is how you can front end a 5250 wizard user interface. In EGL, this not only works for web services that have been create with HATS, any wsdl you get from somebody and you want to write a web service client for that web service, you can certainly use EGL to use this web service and write a user interface that fits in front of this web service.

I hope you have enjoyed this demo. Please try out EGL and web services and modernize your applications.

Thanks very much!