Invoking a third party Web service with EGL and Web 2.0

Skill Level: Intermediate
Reginaldo Barosa Executive IT Specialist – IBM Boston
April, 2010

Abstract

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

Web 2.0 describes the changing trends in the use of World Wide Web technology and web design that aim to enhance creativity, communications, secure information sharing, collaboration and functionality of the web. Web 2.0 lets your company quickly develop *mashups*, each of which is a combination of software capability from different sources. For example, you might be developing an application to let travelers reserve rooms in one or another city. With Web 2.0, the application can provide access to a Google™ map for each hotel and can include, within the map, a weather forecast for the city. Neither the mapping software nor the forecast software was created with the other in mind, yet the two kinds of software are usefully integrated in a creative way.
EGL is converted automatically into JavaScript™. And implements Web 2.0 Because EGL supports Safari, the generated applications work on the iPhone as well in web browsers like Microsoft® Internet Explorer versions 6 through 8, Firefox 2 and 3, and Safari 2 and 3

Before you start

Learn what to expect from this tutorial, and how to get the most out of it.
Be sure that you have started the System z Sandbox and that IBM® Rational® Developer for System z is running under the Windows in the system z Sandbox.

About this series

Walk through this scenario and others online as part of the Assets entry point and the Skills entry point of the Enterprise Modernization Sandbox for IBM® System z®.

About this tutorial

This tutorial shows you how to create a Rich UI (User Interface) web page that will invoke a Web service built from another product and deployed outside of your environment using IBM® Rational® Business Developer, which supports service-oriented architecture (SOA) and Web 2.0.
SOA is a method of organizing applications in modular pieces (called services, including Web services). The services provide logic to the clients in the form of functions, similar to the way that Enterprise Generation Language (EGL) libraries make functions available to programs..

Objectives

Demonstrate how to create "state-of -the-art" web interfaces using Rich User Interfaces and Web 2.0 that invoke a Web service that is deployed somewhere and with any technology that you don't need to know.
For example the Web service that we will consume here was deployed into CICS and its running in a mainframe system with z/OS..

Prerequisites

You need to be familiar with basic Web development concepts.

System requirements

The System z Sandbox

Getting started

You must be connected to the system z sandbox and Rational Developer for System z with EGL must be running with a workspace opened.

CICS application was modernized into Web service

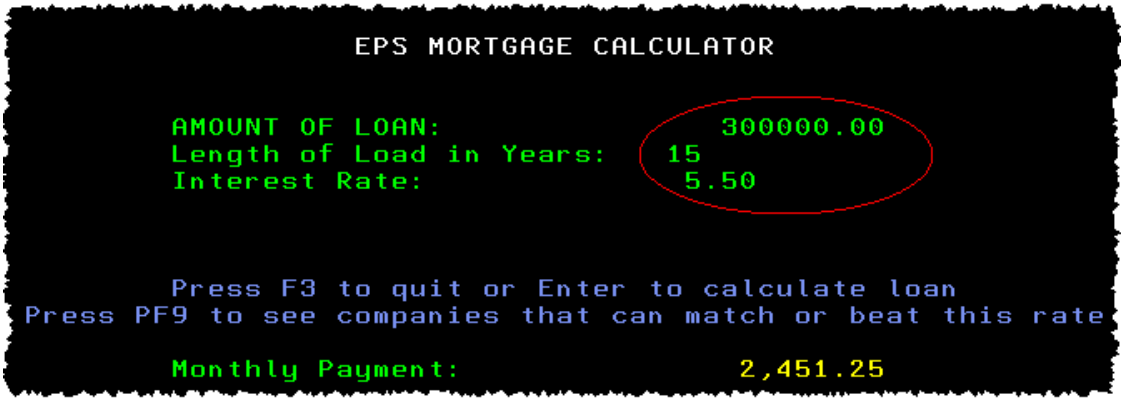
Since we want to show that the Rich UI client application that you will create is independent on the server implementation we decided to use an existing COBOL/CICS code that was transformed into Web service and deployed in our z/OS system located in Texas..

If you are interested in the CICS Services creation to see how this was done, refer to
IBM Enterprise Modernization for System z: Wrap existing COBOL programs as Web Services with IBM Rational Developer for System z

This CICS application has many functions but the one the one that you will use does a simple mortgage calculation.

Figure 1 shows this application interface when using a CICS terminal. The user types the Amount to loan, the length in years and the Interest Rate. The CICS server application returns the Monthly payment.

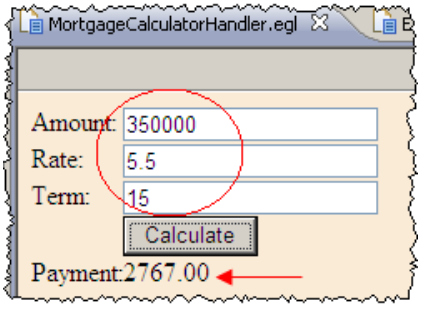
Figure 1. CICS application running in 3270 terminals



What you will learn from this tutorial

By completing the exercises in this tutorial, you will learn how to create and test EGL Rich UI components that will invoke Web Services. When you complete this tutorial your Rich UI Web page will be similar the one show in the Figure 2, instead of this ugly black screen on Figure 1.

Figure 2. Rich UI component created by EGL



Sequence of activities

These are the steps that you will perform in this tutorial:

1 **Create an EGL Rich UI project that will hold the EGL components**
2 **Import the WSDL into the EGL Rich UI project and test it**
3 **Create the Rich UI components.**
4 **Create the EGL code to consume the Web Service deployed into CICS**
5 **Complete the EGL Rich UI code to invoke the Web service**
6 **Test the Rich UI Application**
7 **Improve the code.**

Section 1 - Create an EGL Rich UI project that will hold the EGL components

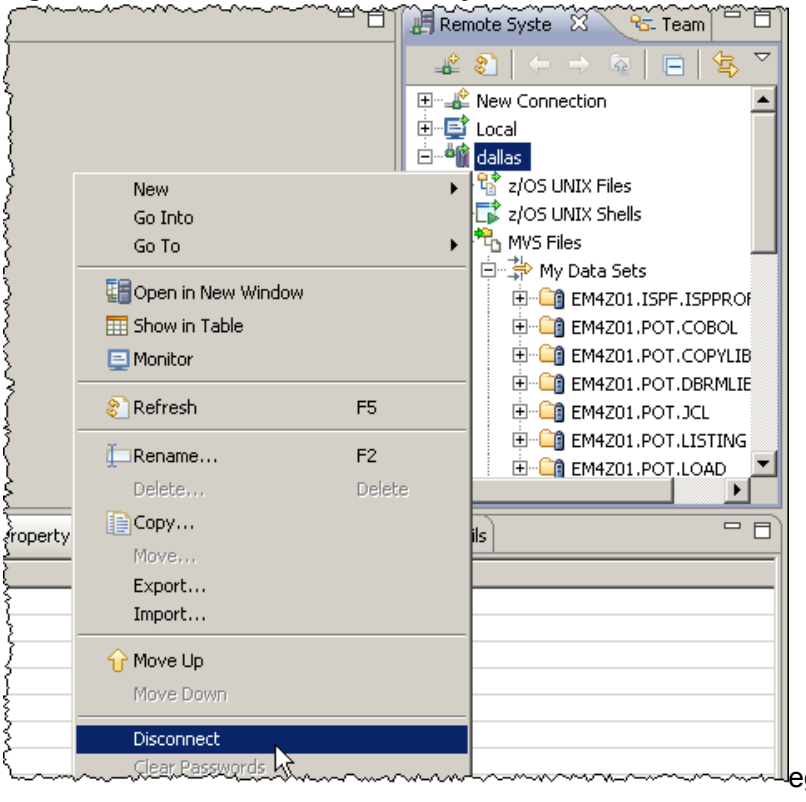
This example assumes that Rational Developer for System z with EGL is already started. Also note that you are operating under VMWARE, which may slow response time.

Disconnect from z/OS System

When The VMWARE session is started you will be connected to z/OS system and you will see the z/OS Projects perspective..

1. Because in this tutorial you will not use z/OS, you can disconnect to the z/OS system. Right click on **dallas** and select disconnect as shown in Figure 2.

Figure 2. Disconnection from z/OS system

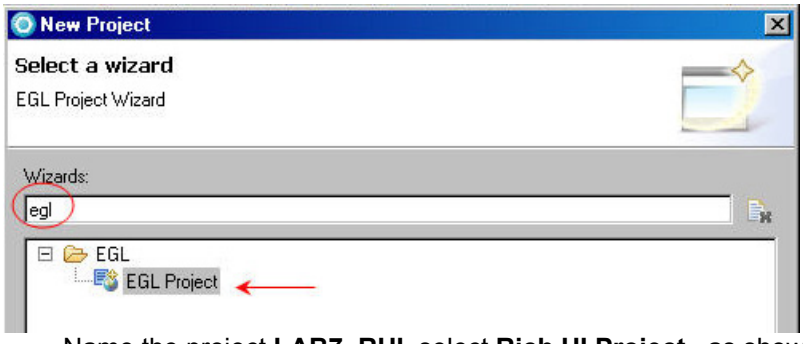


eglfile

Creating the new project

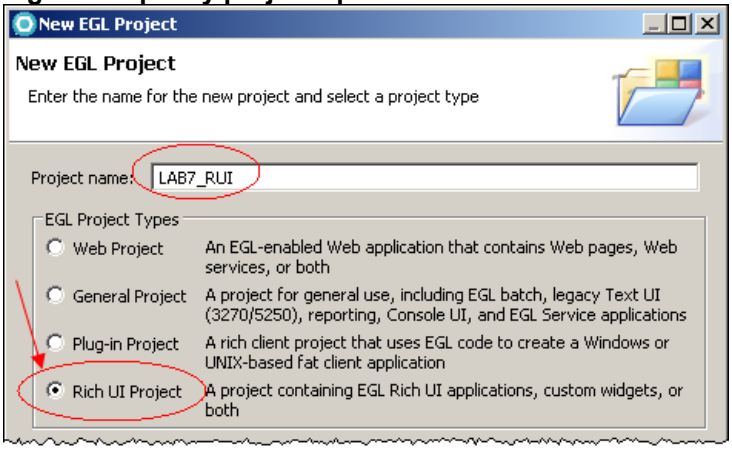
1. To create a new EGL Web project in the Workbench, click **File > New > Other** (or use Ctrl + N).
2. Type **egl** in the Wizards field, select **EGL Project**, as shown in Figure 3 and click **Next**.

Figure 3. Select EGL Project wizard



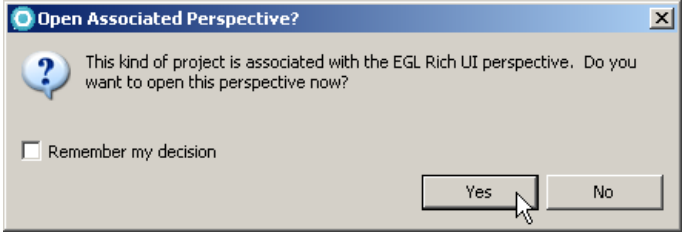
3. Name the project **LAB7_RUI**, select **Rich UI Project** , as shown in Figure 4 and click **Finish.** Remember that VMWARE will slow you down, be patient until this operation give you another dialog.

Figure 4. Specify project options



4. You need to use other perspective than *z/OS projects*, ; and if the *Confirm Perspective Switch* dialog displays with a message asking if you want to switch to the EGL Rich UI perspective, click **Yes,** as shown in Figure .5.

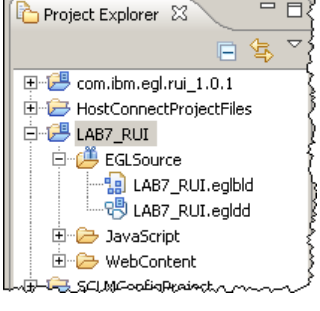
Figure 5. Open Rich UI perspective.



5. Expand the **LAB7_RUI > EGLSource** folder in the Project Explorer. Notice that two EGL build descriptor named *LAB7_RUI.eglbld* and the *LAB7_RUI.egldd* deployment descriptor was created.

6. Notice also that the wizard automatically create another project named **com.ibm.egl.rui_1.0.1**. (it may be already on your workspace)
This second project will be referred by your *LAB7_RUI* project.
Expand *LAB7_RUI* project to visualize what has been created as shown in the Figure 6.

Figure 6. LAB7_RUI project created



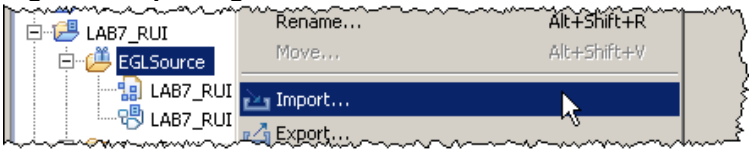
Section 2 - Import the WSDL and test it

Import the WSD into the Rich UI project

The CICS Web Service creator must provide the Web Service Definition Language (WSDL) to any client interest in use it. This WSDL is located in the file C:\EGL_POT\LAB7B \EPSCSMR.wsdl you will need to import it into your project.
You need to move this WSDL under EGLSource folder.

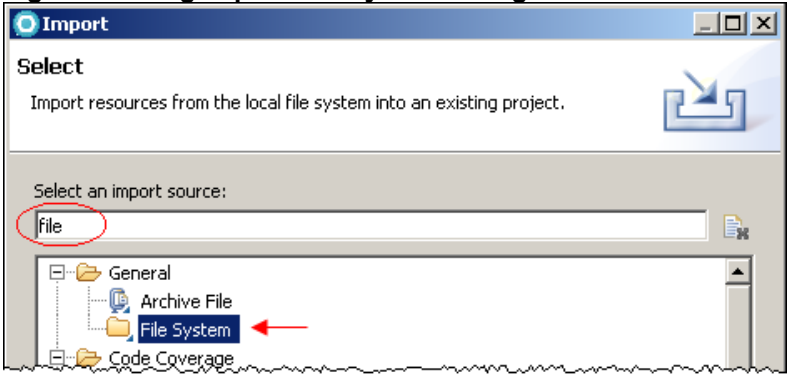
1. To import the WSDL, expand **LAB7_RUI** and **EGLSource** and then right click on **EGLSource** and select **Import….**

Figure 7. Importing the WSDL



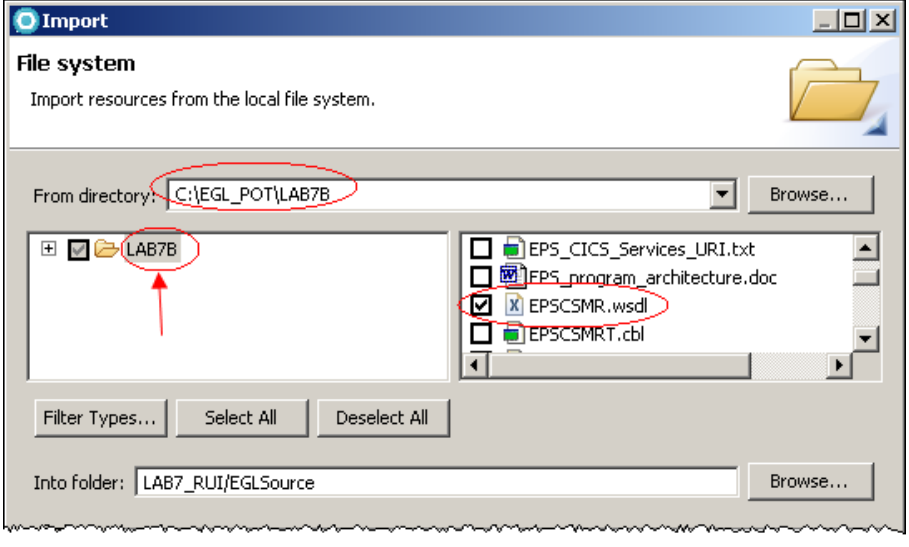
2. On the Import dialog type **file,** select **File System** and click **Next**

Figure 8. Using Import File System dialog



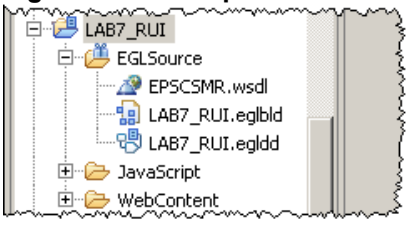
3. Navigate to **C:\EGL_POT\LAB7B** , click on **LAB7B** (not the whole box), select **EPSCSMR.wsdl** and click **Finish**.

Figure 9. Importing the WSDL File



4. Expand **EGLSource** folder to see the file copied there

Figure 10. The imported WSDL is now underEGLSource

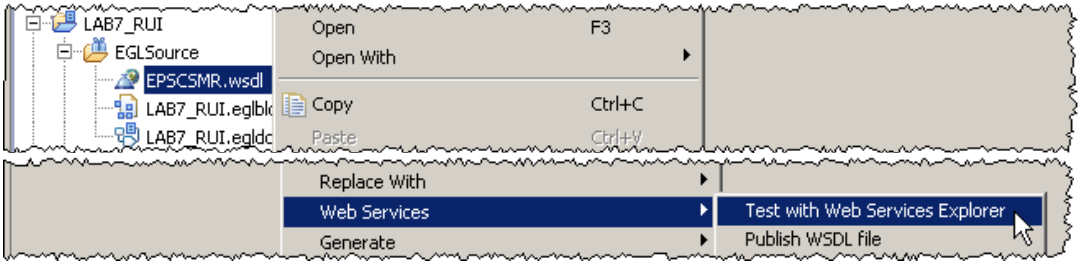


Test the WSDL imported to invoke the Web service

Rational developer products provide a nice and easy test facility to invoke Web services from existing WSDL components.

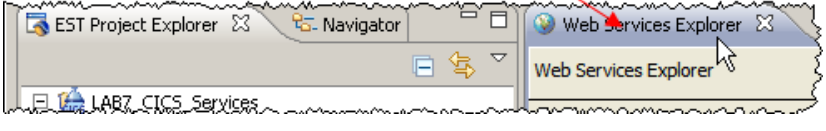
1 Using the Project Explorer view, right-click on **EPSCSMRD.wsdl** and select **Web Services > Test with Web Services Explorer**

Figure 11. Invoking Web Services Test Explorer



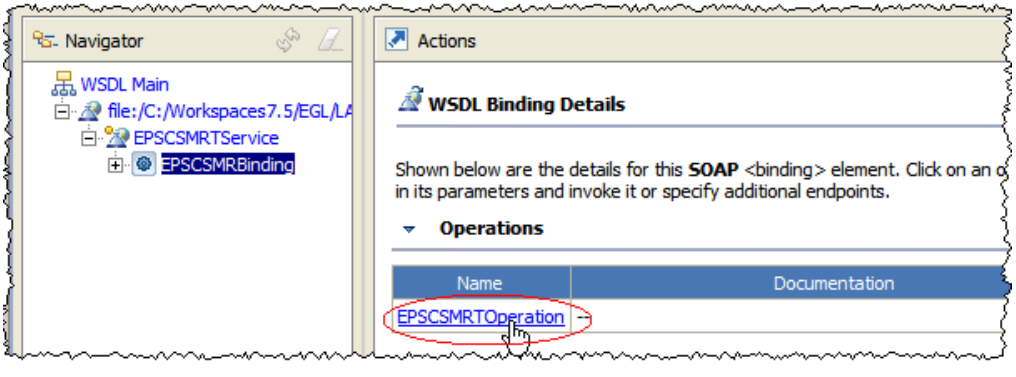
2 Be patient (you are under VMware). After a while, the *Web Services Explorer* will be launched in a Web Browser view.
Resize the view to see all areas of the Web Services Explorer. Or you just **double-click on the Web Services Explorer** title to display full screen.

Figure 12. Invoking Web Services Test Explorer



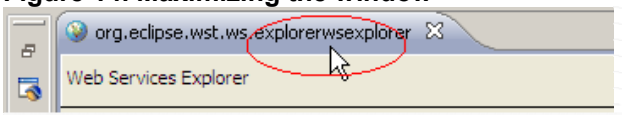
3 Click on the link for the **EPSCSMRTOperation** operation within the *Actions* section, (alternatively, you can click on the **+** sign beside *EPSCSMRBinding* and expand the view to display the *EPSCSMRTOperation*

Figure 13. Invoking Web Services Operation



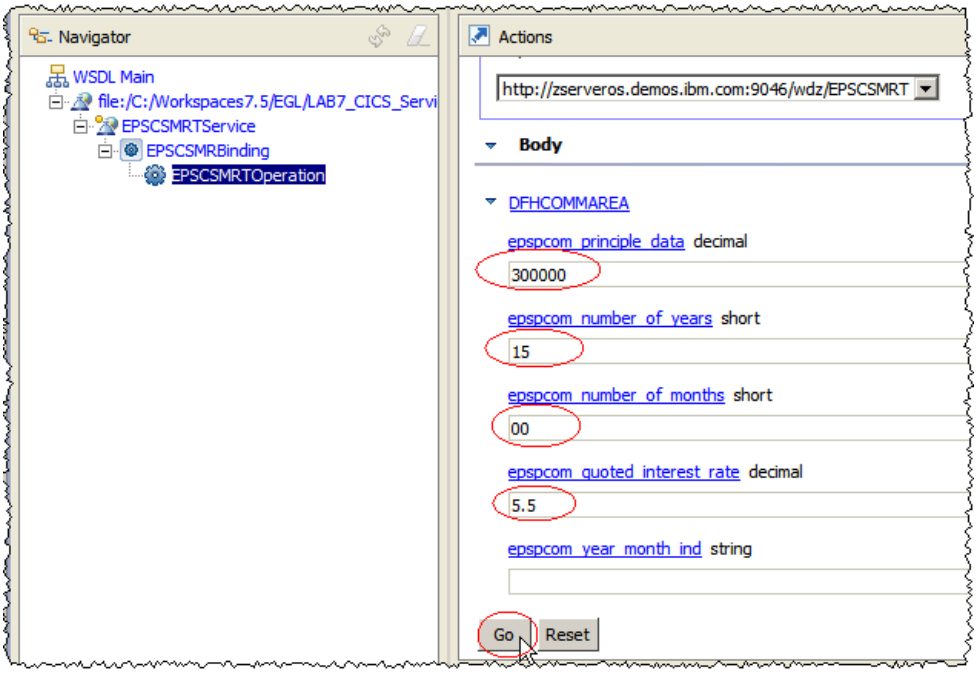
4 Be sure that this view is maximized. Just double click on the title.

Figure 14. Maximizing the window



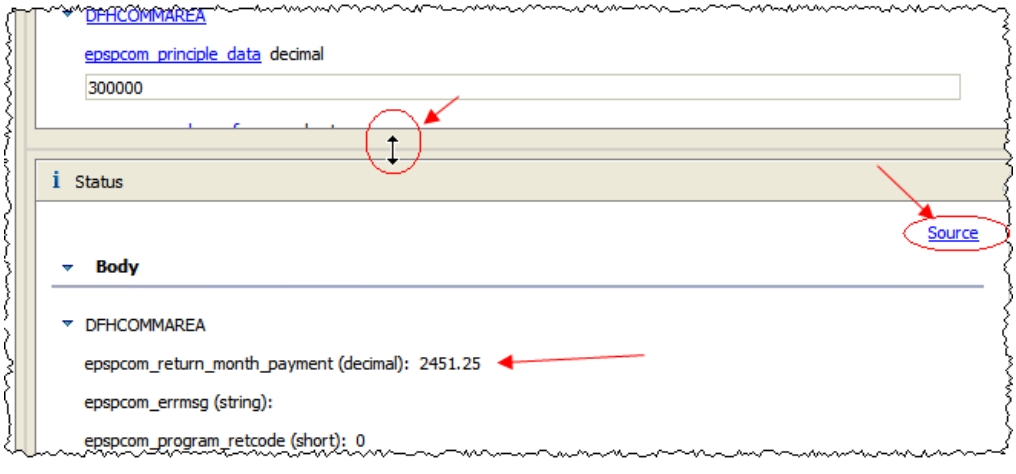
5 Notice that the Actions section is now replaced with information about the *EPSCSMRTOperation* operation and the parameters it takes. It also lists the endpoint associated with this request.
Type valid numbers like **300000, 15 , 00** and **5.5** and click **Go** to invoke the operation.

Figure 15. Typing values to test the Service



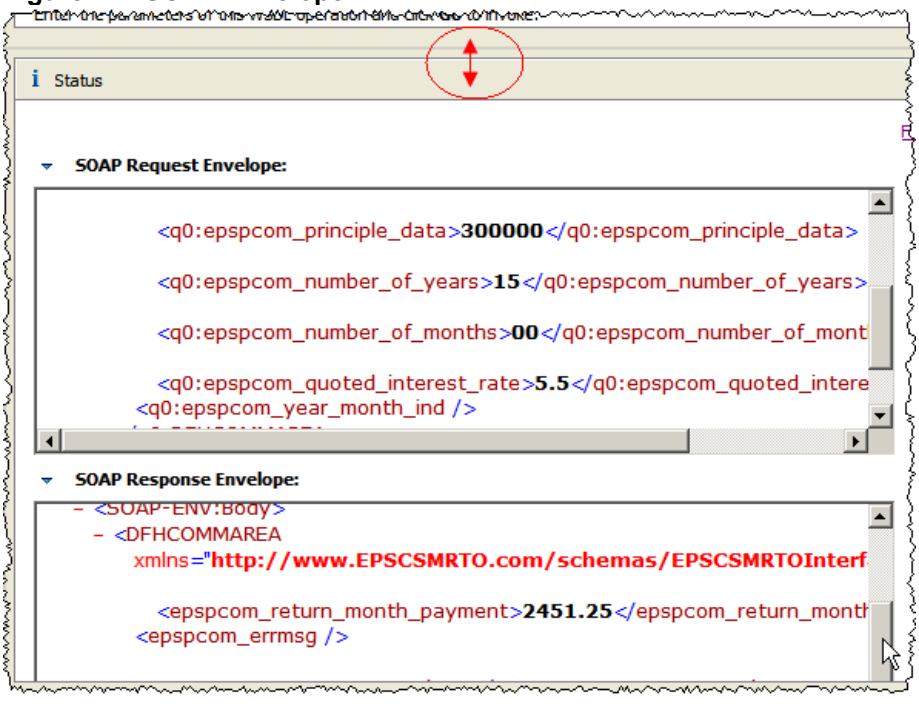
6 Resize the *Status* section and you should see results as shown in Figure 18.
Click on **Source** to see the SOAP Envelopes

Figure 16. Web Service results



7 Resize again and you will see the XML SOAP envelopes with the input and output messages as seen in Figure 17:

Figure 17. SOAP Envelope



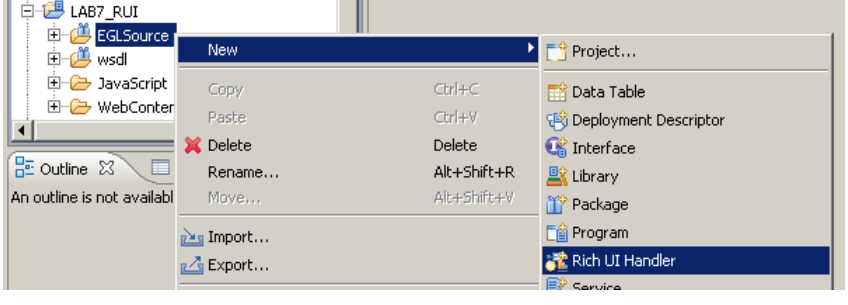
8 After you are done testing, close the web browser using (**Ctrl + Shift + F4**).

9 Reset the Enterprise Service Perspective: **Window > Reset perspective**

Section 3 - Create the Rich UI components

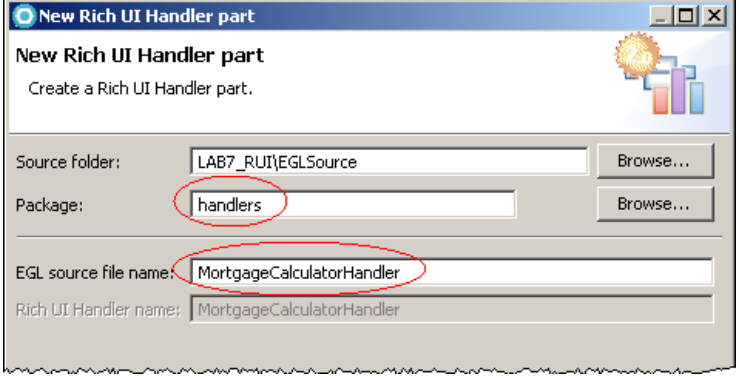
1 You will now create the EGL code that will invoke the Web Services. Right click on **EGLSource** and select **New > Rich UI Handler** amount

Figure 18. Starting Rich UI handler



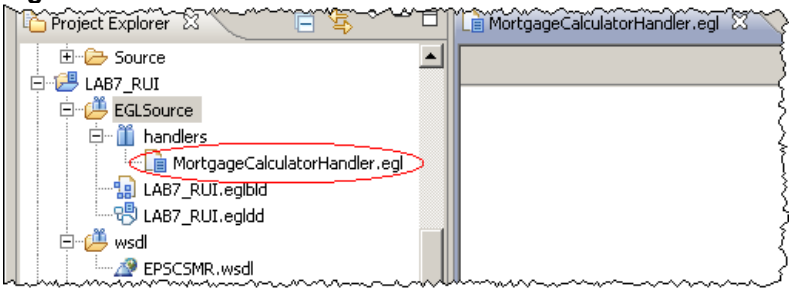
2 Type **`handlers`** as *Package* name and **`MortgageCalculatorHandler`** as *EGL source file name* and click **Finish**.

Figure 19. Creating EGL Rich UI handler



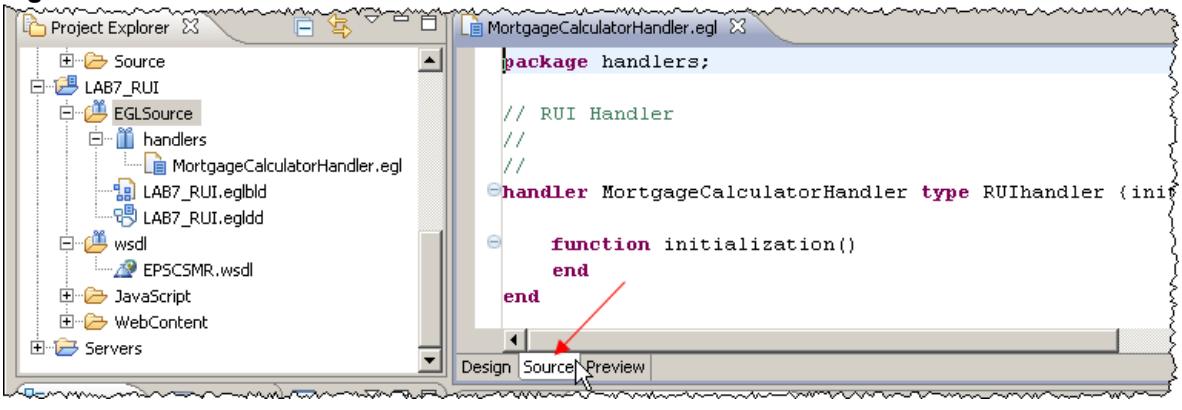
3 The EGL Rich UI editor will open with a blank page.
Expand **EGLSource** and **handlers** package to see the **MortgageCalculatorHandler.egl** code created.

Figure 20. The EGL handler is created



4 Click on the tab **Source** to see the EGL code generated . Note that the type *RUIhandler* is the EGL code where you will add the Rich UI controls and if desired the business logic or web services invocations.

Figure 21. Source tab shows the EGL code



Understanding how browsers handle a Rich UI application
The internal data areas used by the browser are represented as an inverted tree.
The tree is composed of a root -- named document -- and a set of elements, which are units of information. The top most element that is available to you is named body. The elements subordinate to body are specific to your application.
A set of rules describes both the tree and how to access the data that the tree represents. That set of rules is called the Document Object Model (DOM). We refer to the tree as the DOM tree. Refer to the Rational Developer for System z help for better explanation and examples of these definitions.

Using the Design surface to create a DOM tree
When you drag a widget from the palette to the Design surface, the areas that can receive the widget are called potential drop locations, and the color of those areas is **yellow** by default. When you hover over a potential drop location, the area is called a selected drop location, and the color of that area is **green** by default. You can customize the colors in the Workbench preferences.

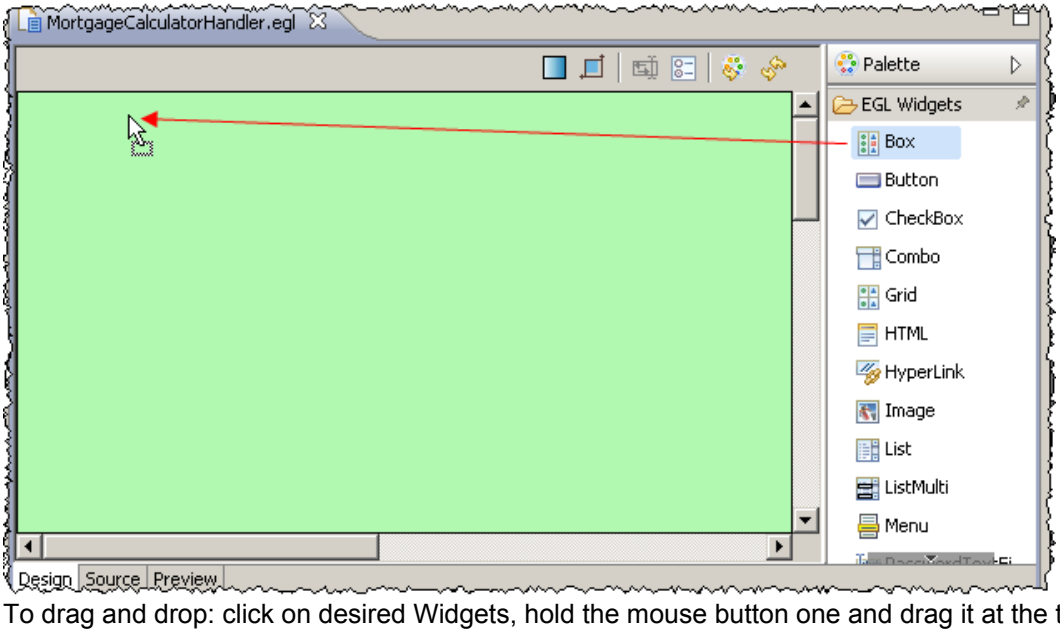
When you first drag a widget to the Design surface, the entire surface is a selected drop location, and the effect of the drop is to declare the widget and to identify it as the first element in the Rich UI handler's initialUI property. That property accepts an array of widgets at development time. The array is ultimately used to create a DOM tree, which is a runtime data structure. Specifically, the elements in the Rich UI handler's initialUI array become children of the document element, with the order of initialUI array elements at development time equivalent to the order of sibling DOM elements at run time.

When you drag another widget to the Design surface, you have the following choices:
1. You can place the widget adjacent to the initially placed widget. The effect on your source code is to declare the second widget and to identify it as another element in the initialUI array. Your placement of the new widget is either before or after the first widget and indicates where the widget is placed in the array.
2. If the initially placed widget was a container—for example, a box—you can place the second widget inside the first. The effect on your source code is to add an element to the children

property of the container. The effect is ultimately to add a child element to the DOM tree; specifically, to add a child element to the element that represents the container.
Your subsequent work continues to build the DOM tree. You can repeatedly perform drag-and-drop operations, with the placement of a widget determining what array is affected and where the widget is placed in the array. The drag-and-drop operation is an alternative to writing a widget declaration and array assignment in the code itself, whether in the Source tab of the Rich UI editor or in the EGL editor.

5 You will now add some Rich UI controls to the blank page.
Click on **Design** tab and using the mouse, drag and drop **Box** on top of the screen
Hold the mouse button one and drag it at the top of the screen

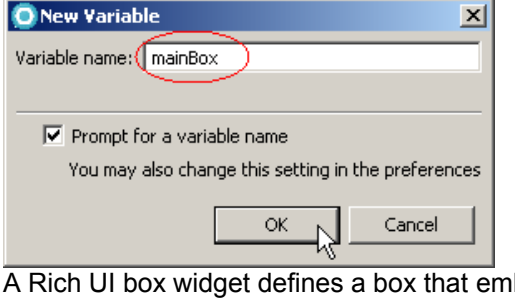
Figure 22. Drag and drop a Box widget



To drag and drop: click on desired Widgets, hold the mouse button one and drag it at the top of the screen..

6 The New Variable window will pop up. Type **mainBox** and click **OK**. It is a good practice to start giving names to the widgets to be created now, this way you can easily identify those in the EGL code that you will see soon..

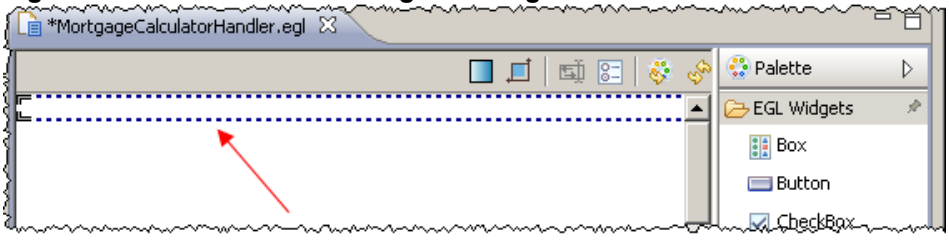
Figure 23. Adding a name to a Variable



A Rich UI box widget defines a box that embeds other widgets that you will create.

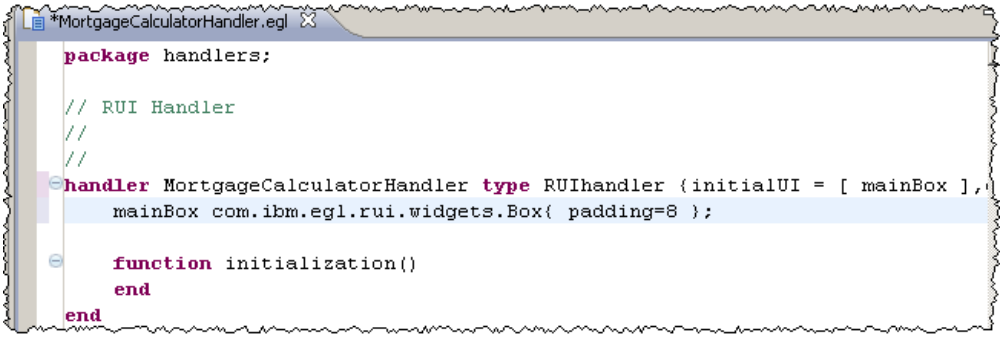
7 Note that a doted area is created. You will add the controls inside of this area.

Figure 24. Doted area indicating Box widget created



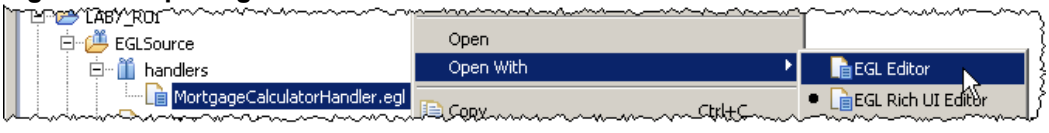
8 Click on the **Source** tab and you will see the EGL generated code. Please spend few minutes to understand this code.
Some points to note:
- *initialUI* specifies which widgets are children of the initial, DOM tree document element.
- *onConstructionFunction* specifies the on-construction EGL function, which is a handler function that is invoked when the handler starts running.

Figure 25. The EGL code generated



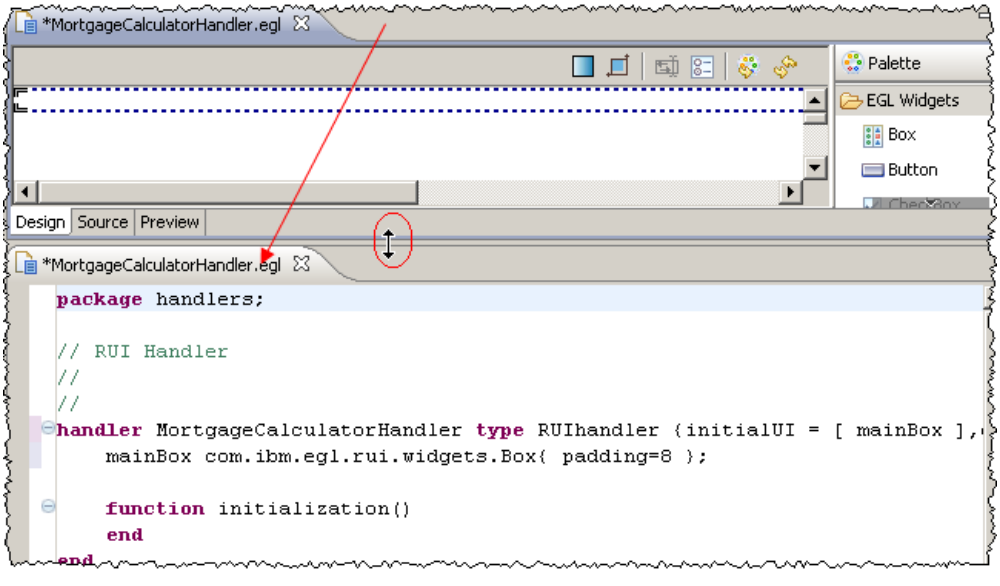
9 Notice that you can combine the EGL Rich UI editor and the EGL editor. This way at any visual change you will see the EGL code generated. This complements the features in the Rich UI editor by opening a single file in both the EGL Rich UI editor and the EGL editor.
Right click on **MortgageCalculatorHandler.egl** and select **EGL Editor** .

Figure 26. Opening the code with EGL Editor



10 Click on the **Design** tab of the first EGL Rich UI editor view and drag and drop the second EGL editor view under this *Design* view (Click and hold the title). Now you will have the *MortgageCalculatorHandler.egl* displayed in two ways as shown below.
At the top is the Design tab of the Rich UI editor, along with a palette that lists the available Widget types.
At the bottom is the EGL editor. Your work in either editor affects the same file and is reflected in the content displayed in the other editor.

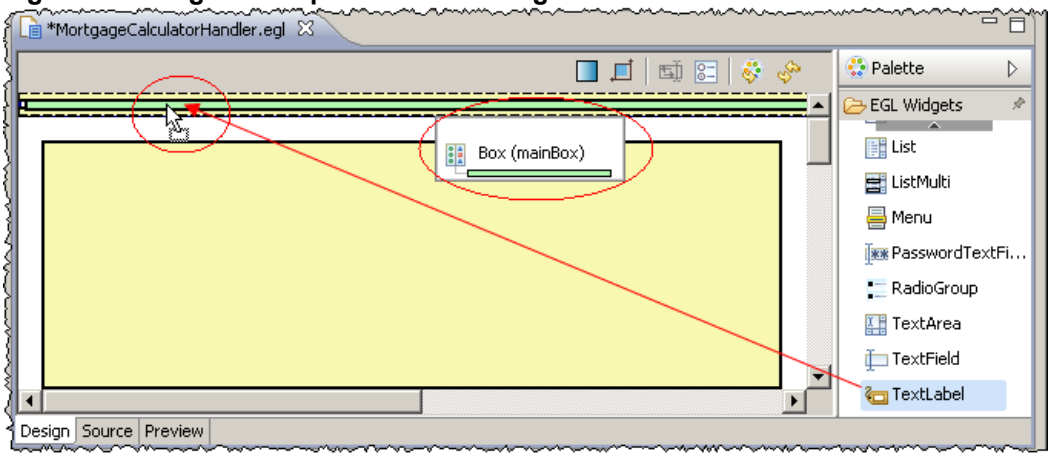
Figure 27. The second editor is opened



11 You will now start to add more EGL widgets.
Drag a **TextLabel** and drop it inside the box created before (**mainBox**) . Note that a picture shows the

location of this control and the green color indicates where this widget will be added. It is important that the green color is showing inside the *mainBox* .

Figure 28. Drag and drop a TextLabel widget



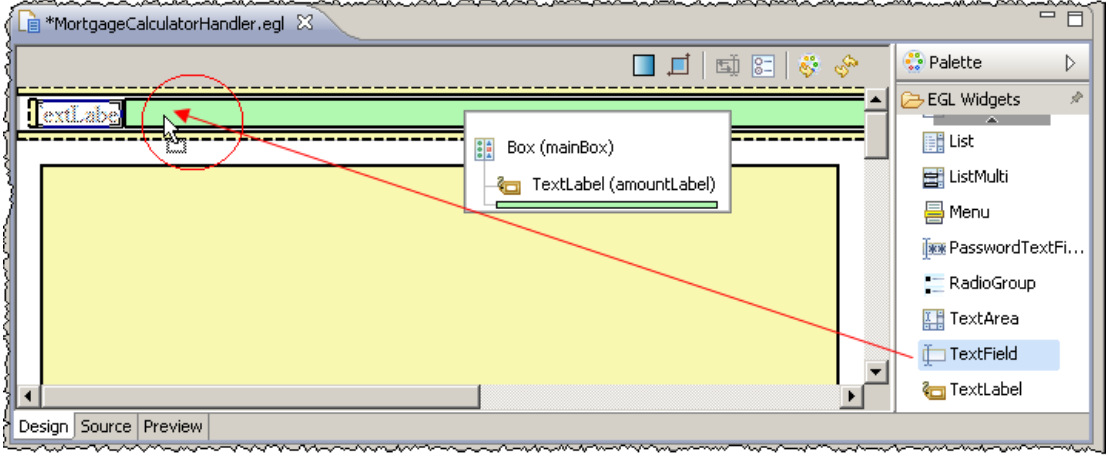
12 For the New Variable dialog, type **amountLabel** as name and click **OK**. You will change the text later. Notice in the **MortgageCalculatorHandler.egl** that *amountLabel* is created as a child of *mainBox* as shown below

Figure 29. EGL code generated

```
handler MortgageCalculatorHandler type RUIhandler {initialUI = [ mainBox ],onCon
    amountLabel com.ibm.egl.rui.widgets.TextLabel{ text="TextLabel" };
    mainBox com.ibm.egl.rui.widgets.Box{ padding=8,
    children = [ amountLabel ] };
```

13 You will need now a widget that will be an entry field.
Drag the widget **TextField** and drop inside the **mainBox** as shown below. Again be sure that the green color is inside the Box and notice the picture displayed.

Figure 30. Drag and drop a TextField



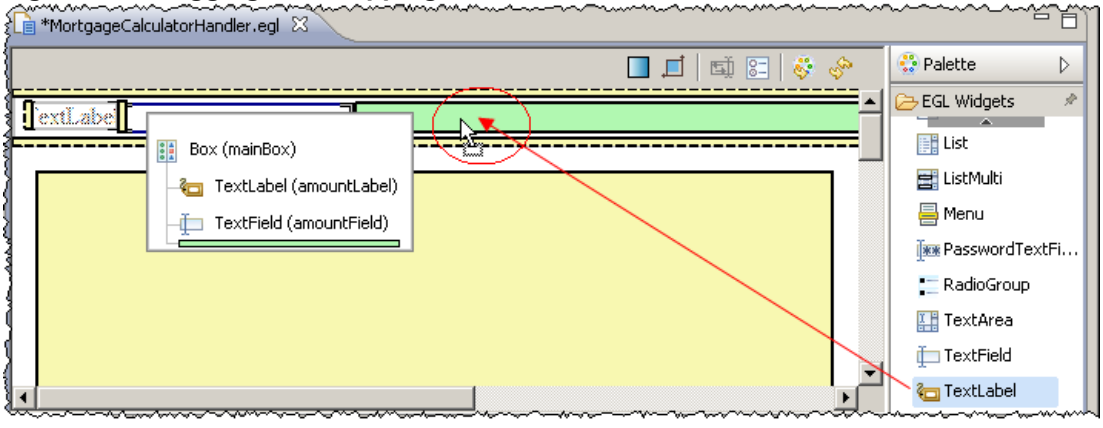
14 Type **amountField** as *Variable name* and click **OK.** This will be the field where the user will type the amount to be calculated.

Figure 31. Naming the variable

Note that in the *MortgageCalculatorHandler.egl* editor the *amountField* is added as a child of *mainBox* also. At this point you have created one label and one field You will need to repeat this for the second operand.

15 Drag a **TextLabel** and drop it inside the box (**mainBox**) after the *amountField* just created above. Again it is important that the green color is showing inside the *mainBox*

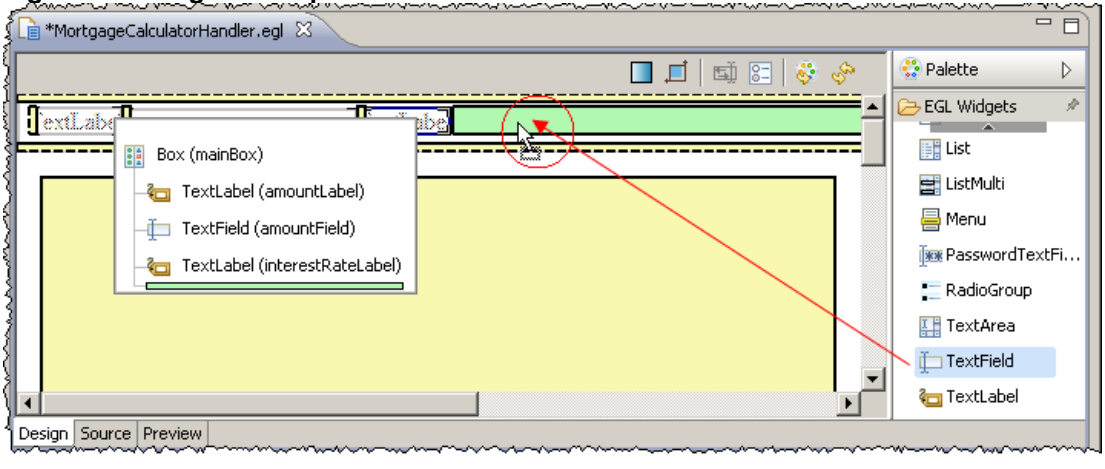
Figure 32. Dragging and dropping a TextLabel



16 Type **interestRateLabel** as name and click **OK**. Notice in the **MortgageCalculatorHandler.egl** *interestRateLabel* is created as a child of *mainBox* as shown below

You will need now another entry field widget. Drag the widget **TextField** and drop inside the **mainBox.** Again be sure that the green color is inside the *mainBox* and note the picture displayed.

Figure 33. Drag and drop a TextField

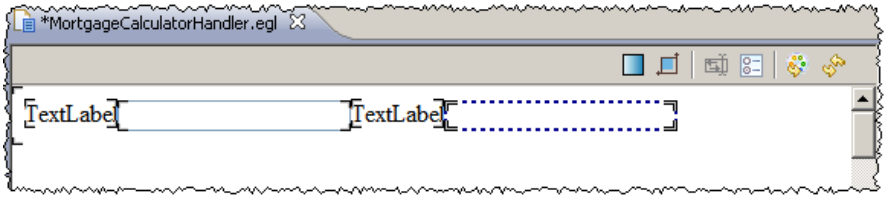


17 Type **interestRateField** as *Variable name* and click **OK** As you did it before, it is a best practice to name widgets that the EGL code will use.

Note that in the *MortgageCalculatorHandler.egl* editor the *interestRateField* is also added as a child of *mainBox.*.

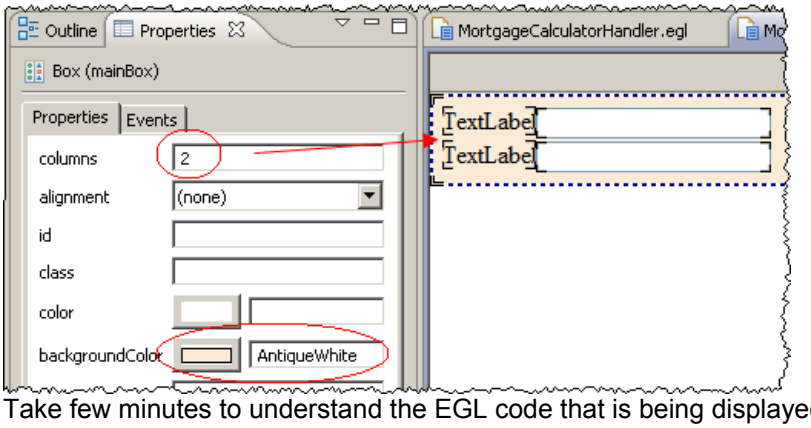
At this point you have created two labels and two fields. After this drop the picture will be as shown below. Notice that you have all widgets in one unique row. Its now time to start changing the Widgets properties.

Figure 34. Controls created



18 You will now change the properties of each widget created.
Select the **mainBox** (you will see a doted line when it is selected) and using the properties type **2** as *columns*, since you want to have two columns of widgets.
You want that each column includes both the *TextLabel* and the *TextField*.
To add a background color, click on the button near the **backgroundColor** click on **Name format**, select a color like **AntiqueWhite**, press **OK** and then press **enter**.
Save the code using **Ctrl + S**. The final result is shown in Figure 35.
Tip: If you are not interested in drag drop the components and prefer to type the code, go to the step 20 and we provide a file with all code you could copy/paste.

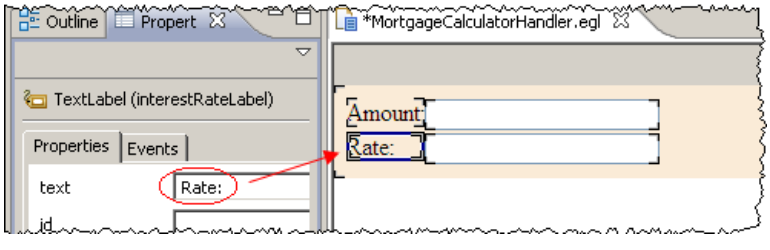
Figure 35. Changing mainBox properties



Take few minutes to understand the EGL code that is being displayed at each change you make it. If you are an experienced EGL Rich UI developer you could just type the code and verify the controls created.

19 You still need to change the two *TextLabel.*
Click on the first **TextLabel** and using the properties change the *text* to **Amount:** and press **enter.**
Repeat this for the second label and change the *TextLabel* to **Rate:**
The result is shown below.
If you do a mistake, you can use the undo (Edit > undo) to return the previous state.
Use **Ctrl + S** to save what you have done so far.
Note the EGL code that was generated.

Figure 36. Change text of interestRateLabel



20 You still need to add the labels, fields and one button. You may add those widgets via drag/drop and changing the properties or typing the EGL code.
It is your choice to make it typing or via drag drop. In this case Always be sure that the green color is under (inside) the *mainBox* and check the picture displayed to help on this

To make this coding task easy, we provided a file under folder **C:\EGL_POT\LAB7B\MortgageCalculator_Controls_only.egl** that you could copy and paste to your egl code.
But if you will build the presentation layer using drag and drop, you must name the widgets as shown below:

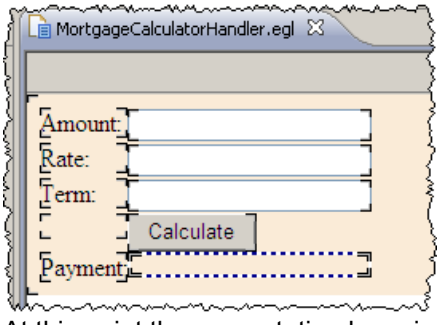
Widget type	*Variable Name*	*Text name*
TextLabel	**termLabel**	**Term:**
TextField	**termField**	none

TextLabel	**TextLabel** (default)	(erase TextLabel)
Button	**calculateButton**	**Calculate**
TextLabel	**paymentLabel**	**Payment:**
TextLabel	**paymentField**	(erase TextLabel)

Use **Ctrl + S** to save what you have done so far

21 The final design after adding all the controls will be as shown below**.**

Figure 37. All widgets created

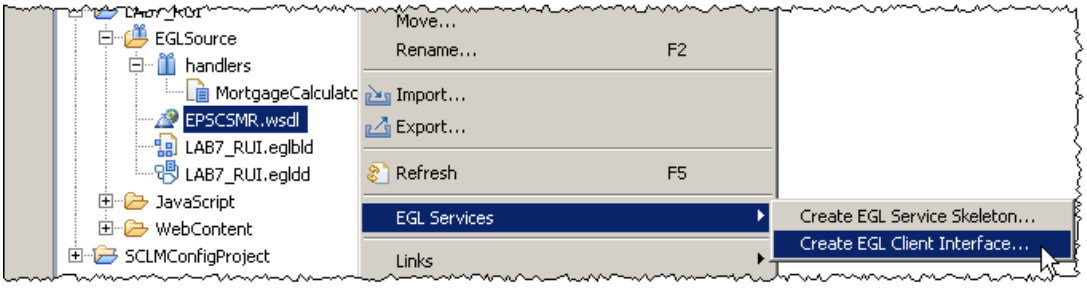


At this point the presentation layer is complete, you must now create the code to invoke the CICS Service and associate it with the button when clicked.

Section 4 - Create the EGL code to consume the Web Service deployed into CICS

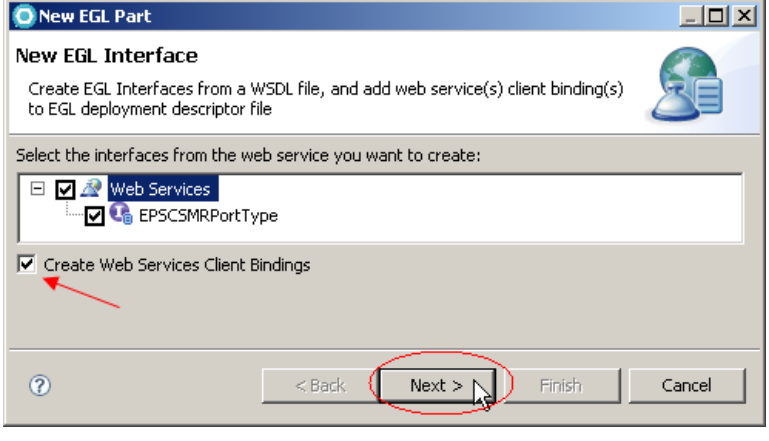
1 You will create the EGL code that will consume the Web Services.
Expand **LAB7_RUI**, **EGLSource,** right click on **EPCSMR.wsdl** and select **EGL Services → Create EGL Client Interface**…

Figure 38. Creating EGL Client Interface



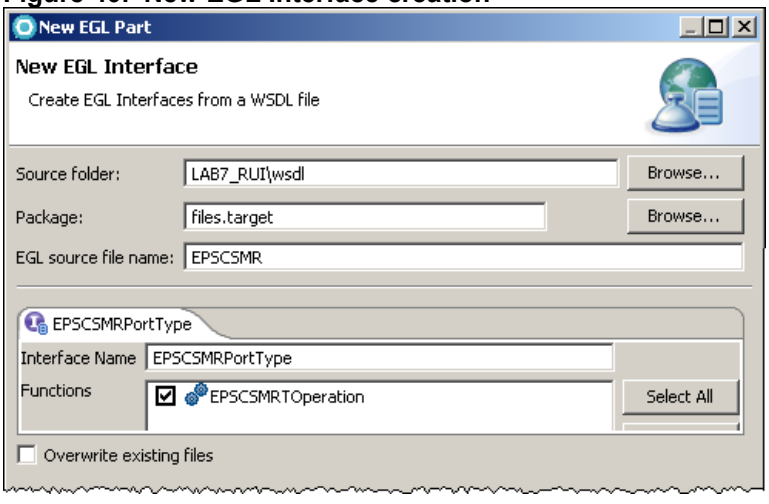
2 Click **Next>** and be sure that **Create Web Services Client Bindings** is selected.

Figure 39. New EGL Interface dialog



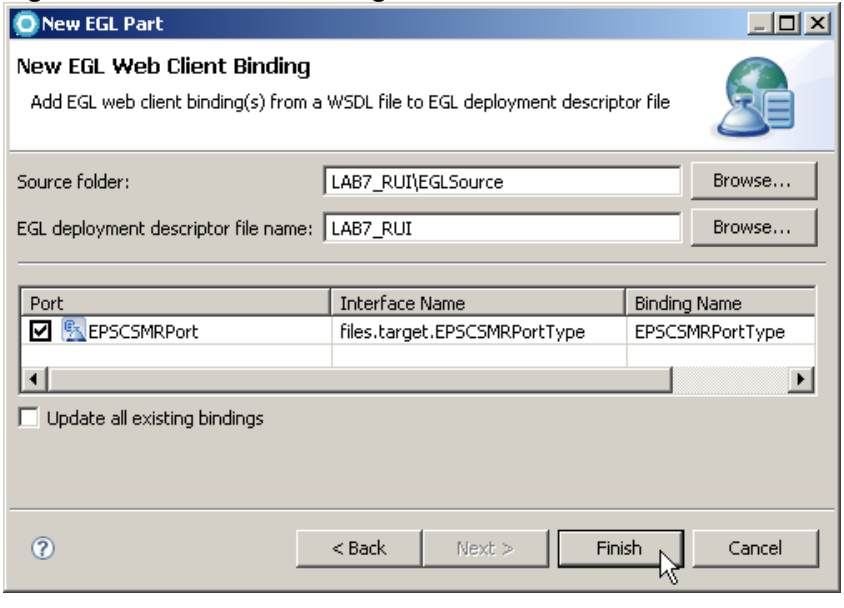
3 Accept all defaults and click **Next>.** Notice the EGL package name (*files.target*) that will be created. Also the EGL source name will be *EPSCSMR*.

Figure 40. New EGL Interface creation



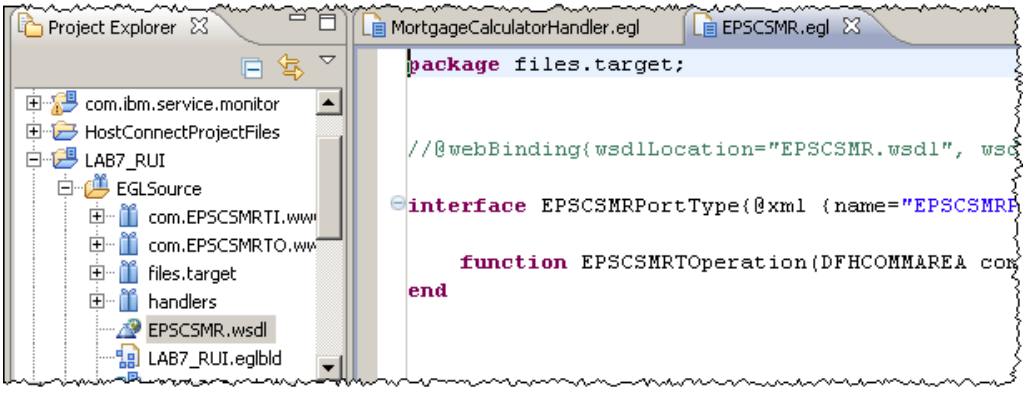
4 Again accept all defaults and click **Finish**.

Figure 41. EGL Client Binding to the Web Service



5 The EGL interface code will be generated and edited.

Figure 42. EGL Interface generated

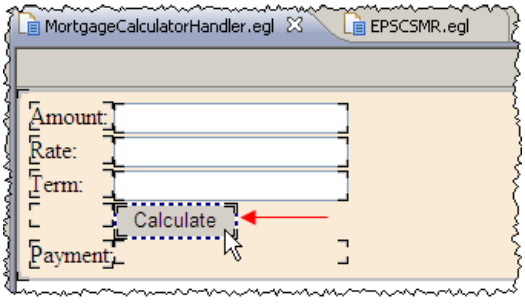


Section 5 Complete the EGL Rich UI code to invoke the Web service

Here you will create the code that will invoke the CICS Service and associate it with the button widget.

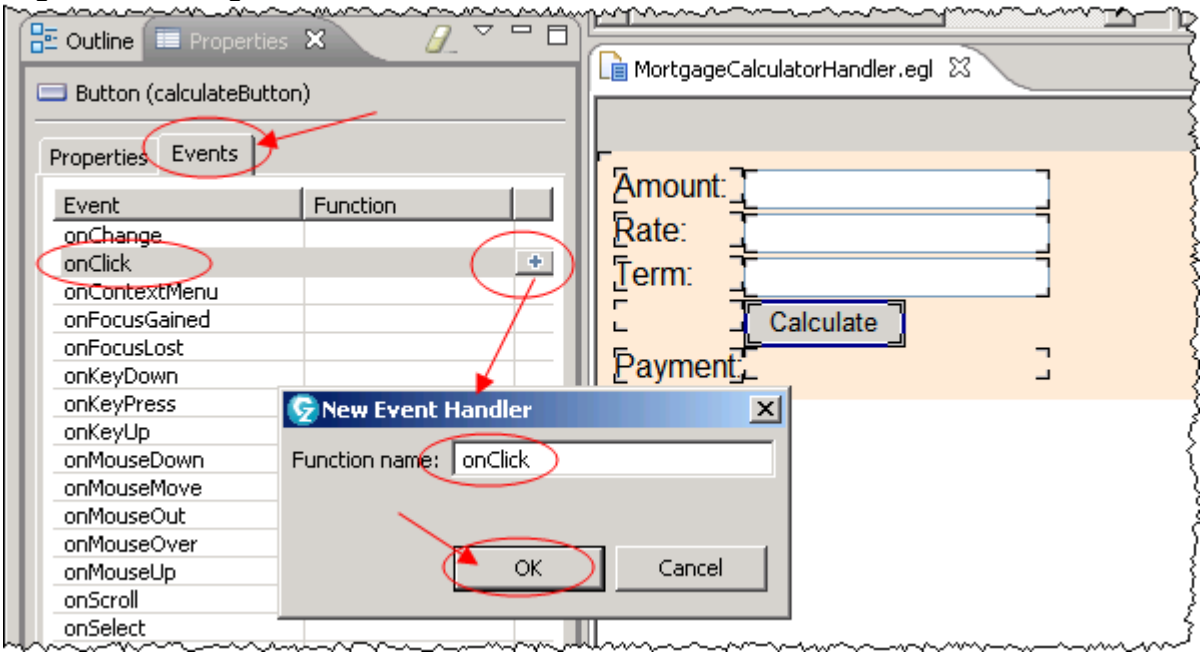
1 You need to create an event when the button is clicked to invoke the CICS Service and populate the field. This event will trigger the EGL function that performs this logic.
Using the *MortgageCalculatorHandler.egl* Rich UI Editor and **Design** tab, click button **Calculate** and be sure that the dotted marks are around the *calculateButton* and NOT the m*ainBox* as shown below.

Figure 43. Selecting the button widget



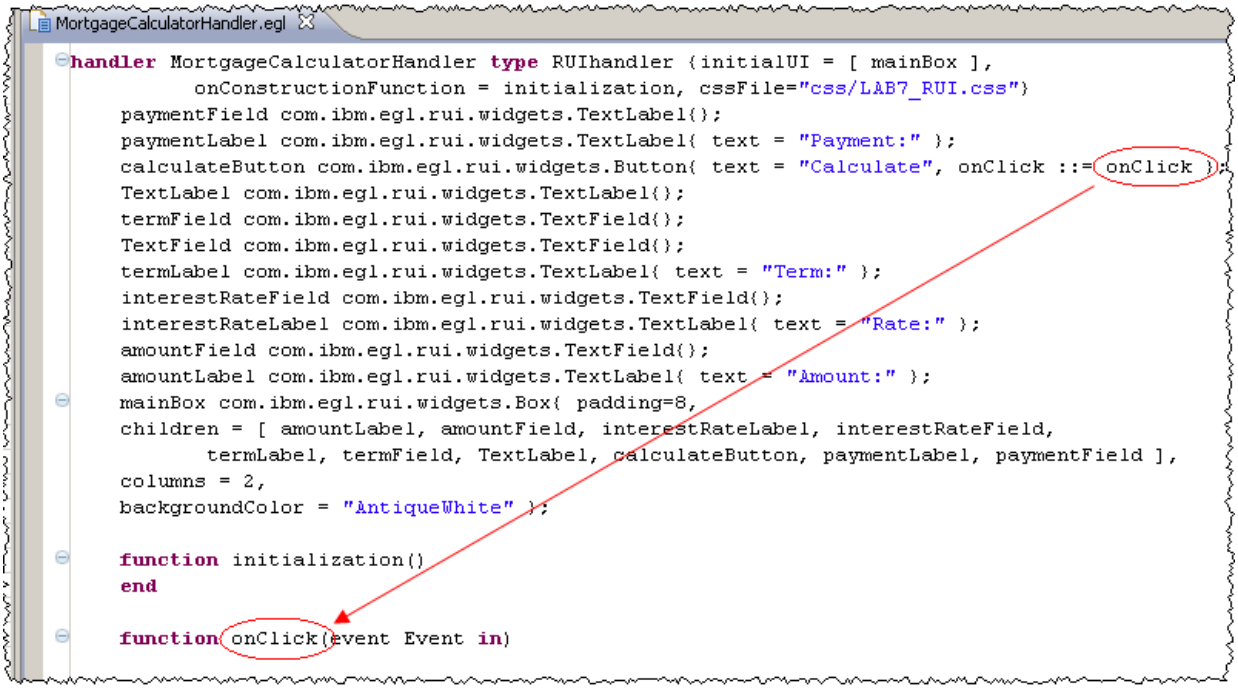
2 Having the *Calculate* button selected, go to the Properties view click on **Events** tab, select **onClick** and click the plus sign button 
Type **onClick** as *Function name* and click **OK**.
This function will be executed when the button is clicked.

Figure 44. Adding an event to the button



3 Save the EGL code generated using **Ctrl + S.** Figure below shows part of the EGL code generated. Note that the **onClick** event will execute the function **onClick**.
Also remember that when you resize your views the command **Window > Reset perspective** > **OK** will return the perspective to the default.

Figure 45. Function *onClick* generated is associated with even *onClick*

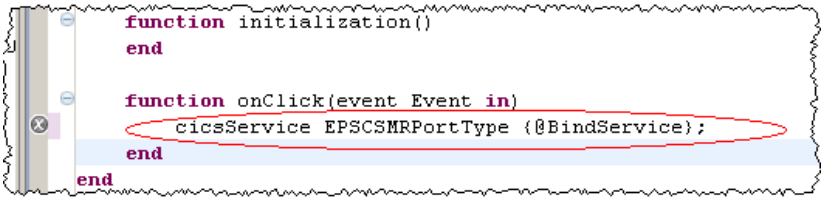


4 The hardest part is complete. Note that all could be done just by typing EGL code and this is probably what skilled EGL developers will do. But the drag and drop capabilities provide a nice way to learn EGL Rich UI coding.
You can test the widgets control created. Click the **Preview** tab.

5 At this point you will work with EGL code since you don't need any visual components. So you can close the second EGL editor and click on the **Source** Tab

6 When button is clicked you need to invoke the CICS Service. An EGL variable must be defined to bind to this service.
Under function **onClick** type a variable named **cicsService** and use **Ctrl + Space** to find **EPSCSMRPortType** and type the property **{@BindService}** as seen below.
Remember to use Ctrl + Space as code assist. It will make your life easier..

Figure 46. Defining a variable that binds to the Web Service



7 You now will define the input message.
With the cursor on the end of the line press **enter** to add a blank line.
Type i**nput** and **Ctrl + Space** and **DF** you will see the input message there, select it

Figure 47. Definition a variable that maps to the input message

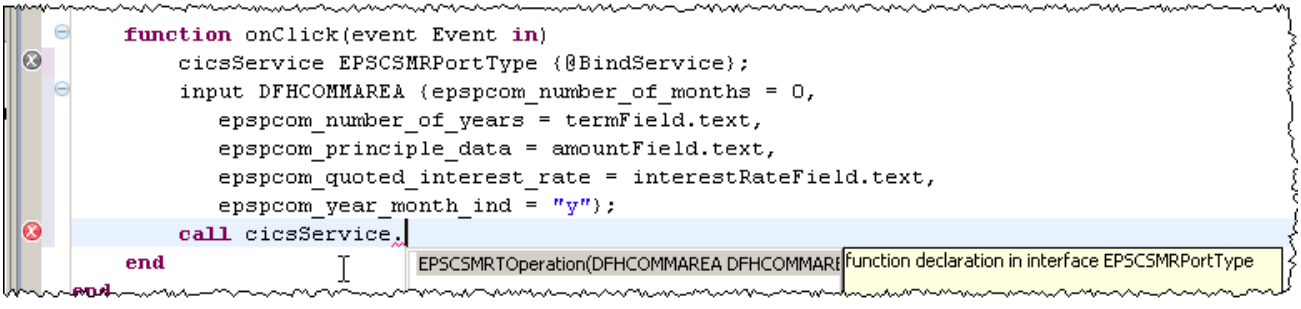
8 All data typed in the Rich UI field must now be assigned to the input message, this will be done now.
For example, if you want to get the value type in the field *amountField* you would code ***amountField.text.***

Using the **Ctrl + Space** code the input message variable as seen below. Note that the **.text** is not working in the code assist for now, but everything else is. You also maybe copy/paste this code from **C:\EGL_POT\LAB7B\MortgageCalculator_Complete.egl** if you are in a hurry. But understand that you are just moving the values to the input message variable.

```
input DFHCOMMAREA {epspcom_number_of_months = 0,
   epspcom_number_of_years = termField.text,
   epspcom_principle_data = amountField.text,
   epspcom_quoted_interest_rate = interestRateField.text,
   epspcom_year_month_ind = "y"};
```

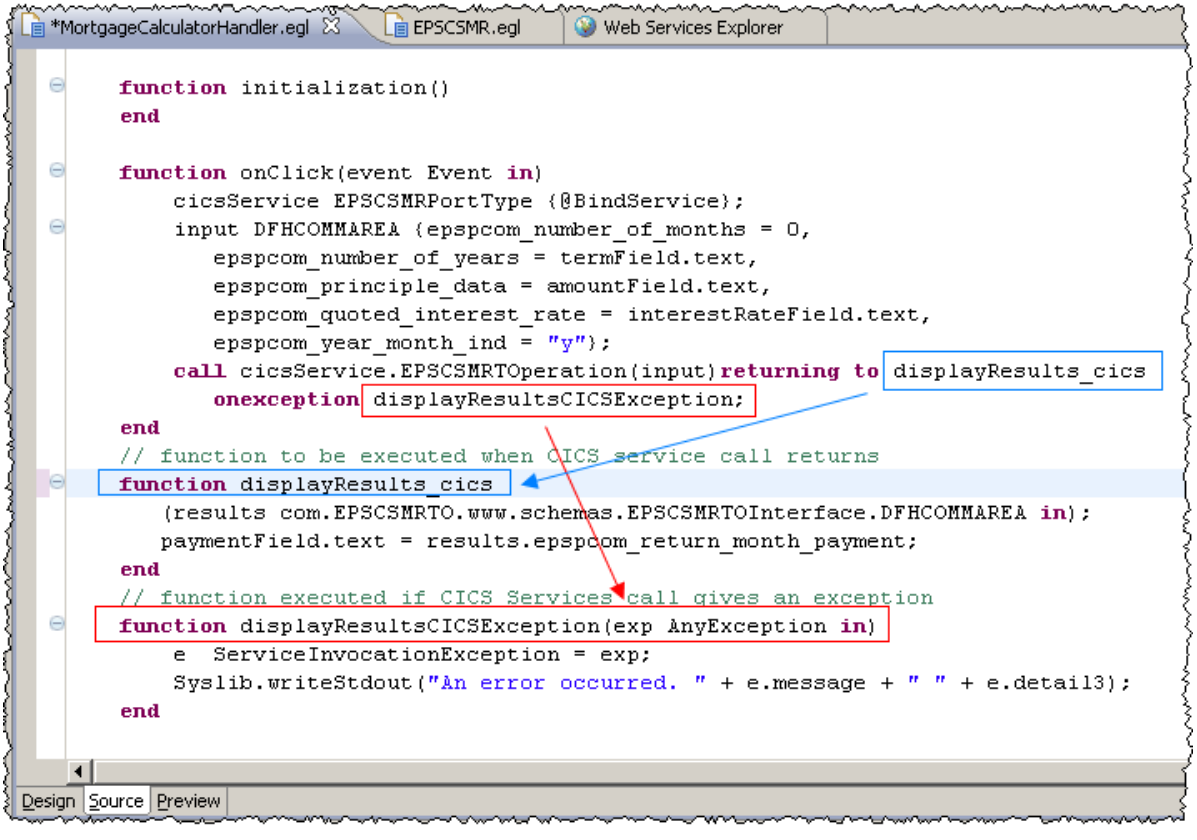
9 To invoke the web service you will define a call statement.
With the cursor on the end of the line press **enter** to add a blank line.
Type **call** and use **Ctrl + space**
Type a dot (.) after the **cicsService** and you will see the possible values (just one operation) as seen below

Figure 48. Using call statement to invoke a Web Service



10 The output message of the CICS Service is defined as the parameter of **displayResults_cics** function. Again the code assist may help you to create the code. The complete call to the CICS Web Service and the functions with the data returned and possible exception is shown below.
This code can be found at: **C:\EGL_POT\LAB7B\MortgageCalculator_Complete.egl** feel free to copy/paste.
Use **Ctrl + S** to save the code

Figure 49. Coding the functions when service is executed with success or not

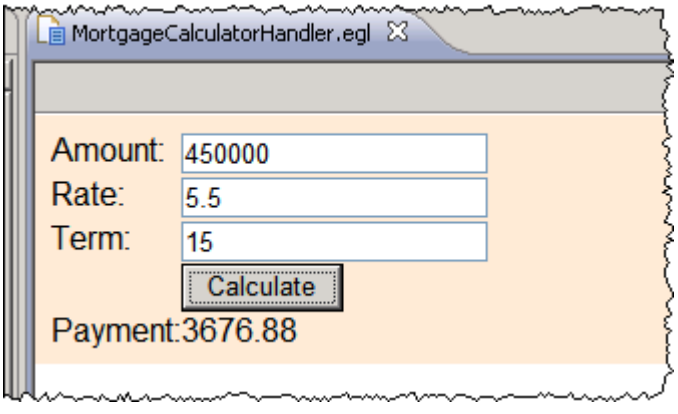


Section 6 Testing the Rich UI Application .

Now you can test the EGL Rich UI code.

1 Click on **Preview** tab and type three valid numeric fields, like **450000, 5.5** and **15** and click **Calculate**. You should have the results below

Figure 50. Executing the code



Note that you are invoking CICS Services from a browser client and no Application Servers are in place. This is all done by JavaScript™ using a supported browser. Cool isn't it?

Section 7 improving your code

You can see that this logic needs lots of improvement however you don't have time here to work on that. Try it: if you type a data that EGL code cannot transform into valid numeric data the Rich UI execution will fail. Try typing some letters there.. You will see the EGL error when trying to translate the values to *smallint* or *decimal* . When you have time you may add more logic on this code to avoid this problem.

One interesting aspect of Rich UI code (that in fact is a JavaScript capability) is the fact that you can have many events for the widgets that you create.

For example you could add and event associated with the fields that would verify if there is no exceptions when the focus is lost or when this field is changed. Using the Rational developer for System z with EGL and the Help page make a search on *onFocusLost* and you will have the description of the many events that might be used here.

Also EGL provides some service classes that may help to test the EGL Services code.
For example the " **tools.ServiceMonitor{};** "

When a service is invoked, it will be shown in the service monitor. We will also indicate if the service succeeded or failed. You can see this monitor on the code below.
Other nice capability is the fact that we can dynamically change the controls. For example, while the web service is being called we can indicate this changing the button content and disabling it. You also can see this in the code below.

1 This is an improved coding. The red rectangle indicates the new code added. See the monitor tools implemented as well the button being changed dynamically. You can find this code at **C:\EGL_POT\LAB7B\MortgageCalculator_Complete_and_Improved.egl** .
Note that if you copy/paste you will need to fix some errors due missing code.. You will do later.
Tips:
1. EGL adds the import statements when you have the code in the EGL Build Path. But in case you copied/paste the improved solution you may need to add this. The easiest way is using the EGL editor type **Ctrl+Shift+O** to execute the Organize Imports.

Figure 51. Complete EGL code

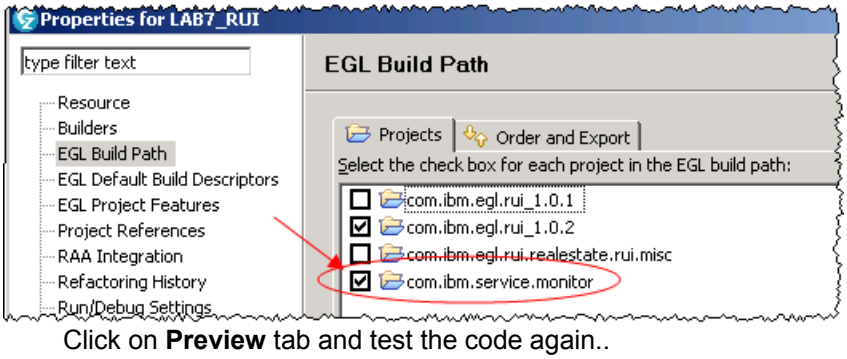
MortgageCalculatorHandler.egl

```
    backgroundColor = "AntiqueWhite" );
    // Create service monitor
    monitor tools.ServiceMonitor{};
    function initialization()
    end
    function onClick(event Event in)
        cicsService EPSCSMRPortType {@BindService};
        input DFHCOMMAREA {epspcom_number_of_months = 0,
           epspcom_number_of_years = termField.text,
           epspcom_principle_data = amountField.text,
           epspcom_quoted_interest_rate = interestRateField.text,
           epspcom_year_month_ind = "y"};
    // Disable the button while the call is in progress
        calculateButton.text = "Calling..."    ;
        calculateButton.disabled = true;
        call cicsService.EPSCSMRTOperation(input)returning to displayResults_cics
           onexception displayResultsCICSException;
    end
    // function to be executed when CICS service call returns
    function displayResults_cics
       (results com.EPSCSMRTO.www.schemas.EPSCSMRTOInterface.DFHCOMMAREA in);
       // Enable the button but change to "Calculate Again"
       calculateButton.text = "Calculate Again" ;
       calculateButton.disabled = false;
       paymentField.text = results.epspcom_return_month_payment;
    end
    // function executed if CICS Services call gives an exception
    function displayResultsCICSException(exp AnyException in)
        e  ServiceInvocationException = exp;
        Syslib.writeStdout("An error occurred. " + e.message + " " + e.detail3);
    end
end
```

2 Verify the LAB7_RUI project properties.

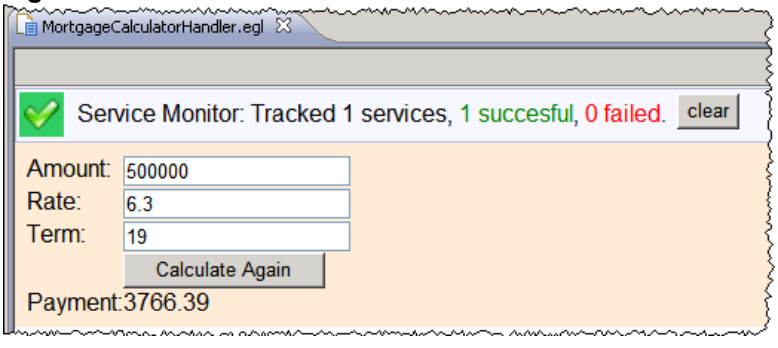
Right click on **LAB7_RUI** and select **Properties.** As shown in the Figure 52, be sure that your project is pointing to the project that has the service monitor. Also remember that you may need to add the EGL import statements on your code.

Figure 52. Defining EGL Build Path properties for the project



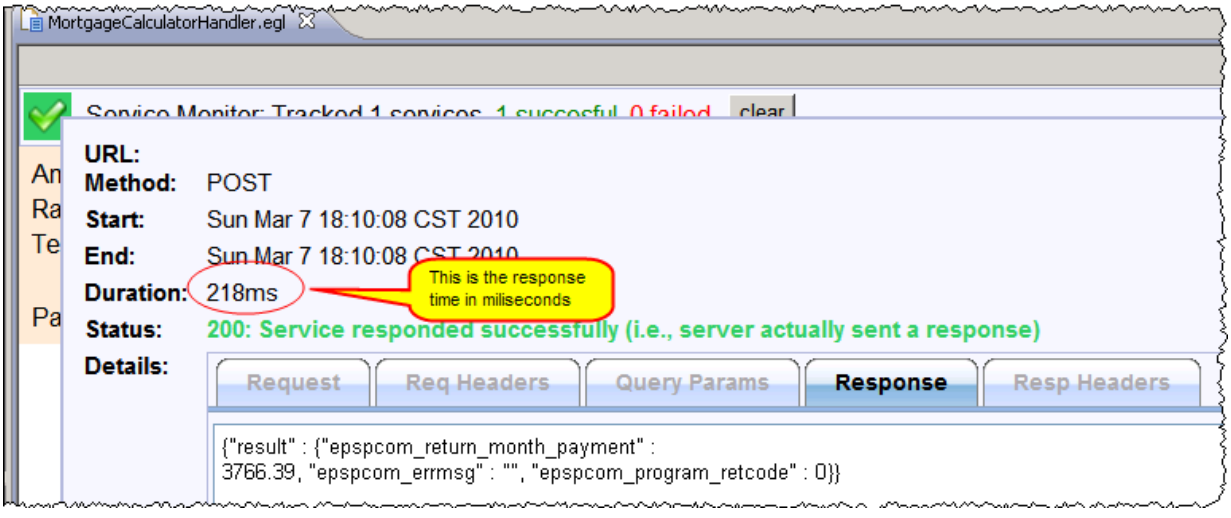
3 Click on **Preview** tab and test the code again..

Figure 53. Execution in Preview mode



4 Click in the green icon and see some data being transmitted and received to CICS .

Figure 54. Data being transmitted



Note: In case the preview does not work, close the egl editor , right click on **MortgageCalculatorHandler.egl** and select **Open with** > **EGL Rich UI Editor** > click **Preview** Tab.

(Optional) Improving service call exception handler

EGL refers to program errors as exceptions. An exception can occur when an EGL-generated program performs any of the following actions:

- Accesses a file, queue, or database
- Calls another program or service
- Invokes a function
- Performs an assignment, comparison, or calculation

You can choose which exceptions you want to handle, by type, or handle all exceptions with the same code. To handle an exception means that you do not allow the current program to terminate, but provide special processing for the error.

The mechanism that EGL uses to handle errors is the try block. Any statement that throws an exception inside a try block causes the program to look for a matching *onException* block within that same try block. If an *onException* block exists that references the exception thrown, control passes to the code within that block.

If a service call ends due to an error, you get a *ServiceInvocationException*. This record includes the following additional fields:

- **source :** EGL, NATIVE, or WEB, depending on the type of service invocation.
- **detail1 :** f the source field is set to WEB, the value here is the "FaultCode" value of the SOAP fault. If the source is set to EGL or NATIVE, the detail1 field is blank.
- **detail2** f the source field is set to WEB, the value here is the "SOAPActor" value of the SOAP fault. If the source is set to EGL or NATIVE, the detail2 field is blank.
- **detail3** f the source field is set to WEB, the value here is the "Diagnostic" value of the SOAP fault. If the source is set to EGL or NATIVE, the detail3 field is blank.

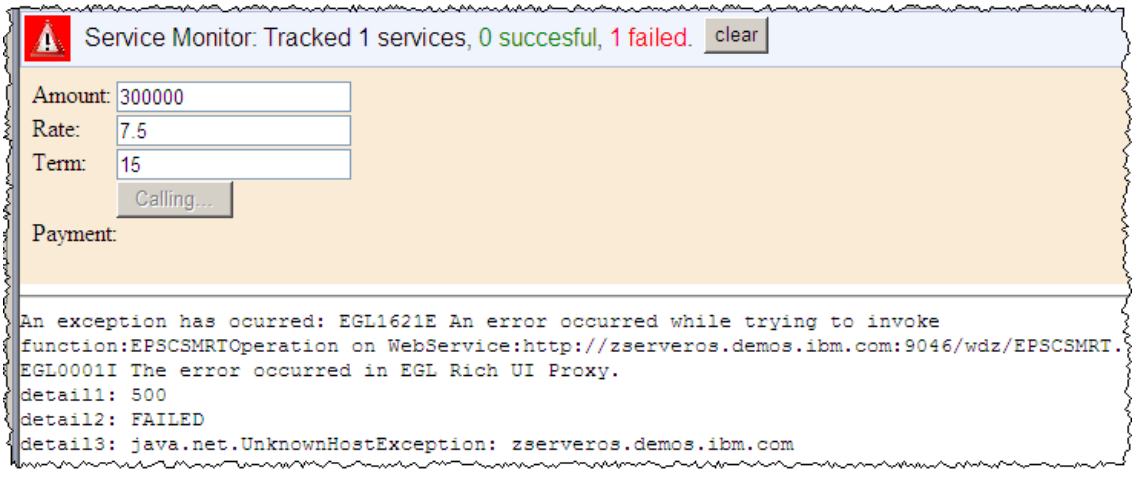
1 To have better information about the service call on our code we can improve the exception handler adding the code below:

Figure 55. Call exception handler improved

```
function displayResultsCICSException(exp AnyException in)
    s string = "An exception has ocurred: " + exp.message + "<br>";
    if (exp isa ServiceInvocationException )
     s += "detail1: " + (exp as ServiceInvocationException).detail1 + "<br>";
     s += "detail2: " + (exp as ServiceInvocationException).detail2 + "<br>";
     s += "detail3: " + (exp as ServiceInvocationException).detail3 + "<br>";
    end
    if (exp isa ServiceBindingException)
        s += "ServiceBindingException" ;
    end
    Syslib.writeStdout(s);
end
```

1 In a situation where the connection is broken, you would have the error below..

Figure 56. Exception handler when connection is broken

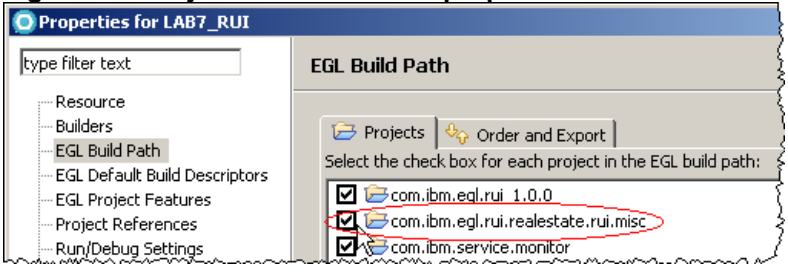


You can find this code exported as Project Interchange File at **C:\EGL_POT\LAB7B\solution\lab7B_solution_with_better_error_handling.zip** select all projects. Use **Project > Clean** > Clean the LAB7_RUI project if necessary.

(Optional) Add a nice graphic widget to your code

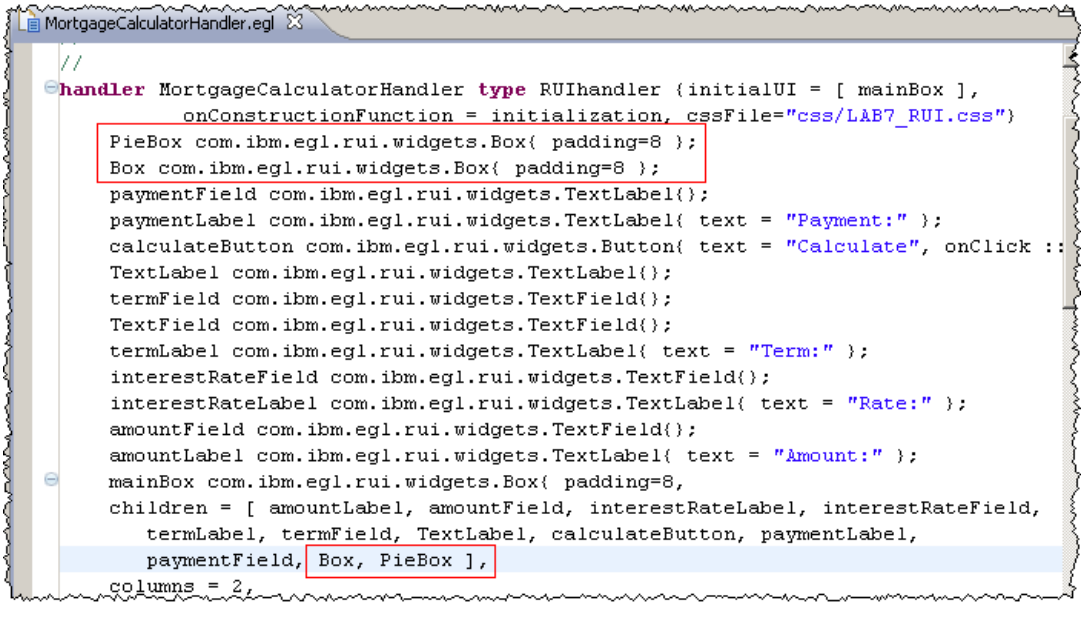
2 Check **LAB7_RUI** project properties and be sure that your project is pointing to the project that has the service monitor. Also remember that you may need to add the EGL import statements on your code

Figure 57. Project EGL Build Path properties



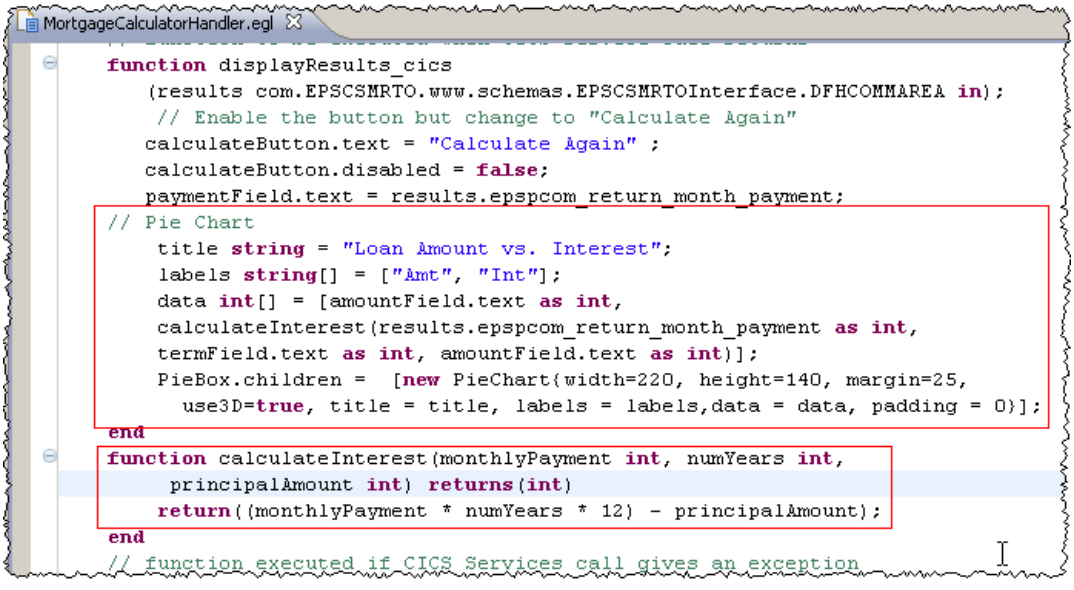
3 Edit your code and add the EGL code as shown below in the red rectangle

Figure 58. Code to define a new Box widget



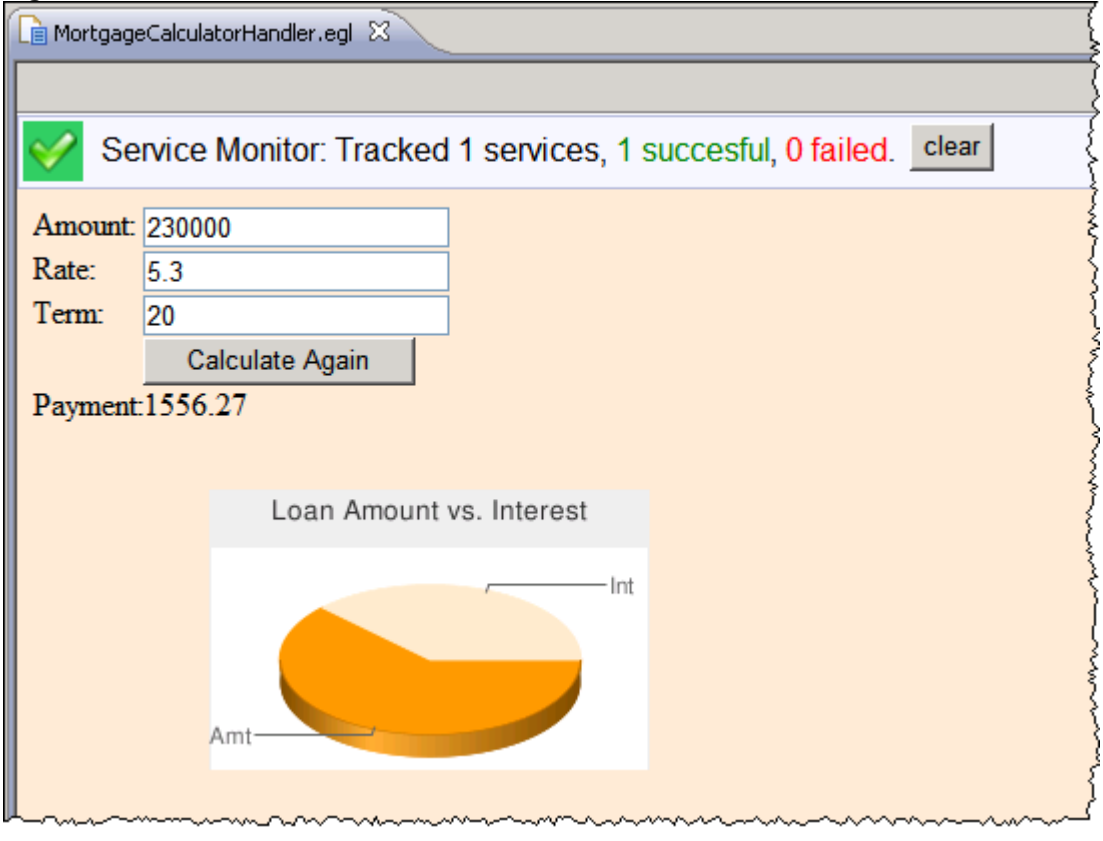
4 And

Figure 59. Code to invoke a Pie Chart widget



5 Click on Preview tab and add some valid data, you will see the new widget in action.

Figure 60. Source tab shows the EGL code



6 A complete project Interchange file can be found at:
C:\EGL_POT\LAB7B\ lab7B_solution_with_graphic_better_error_handling.zip

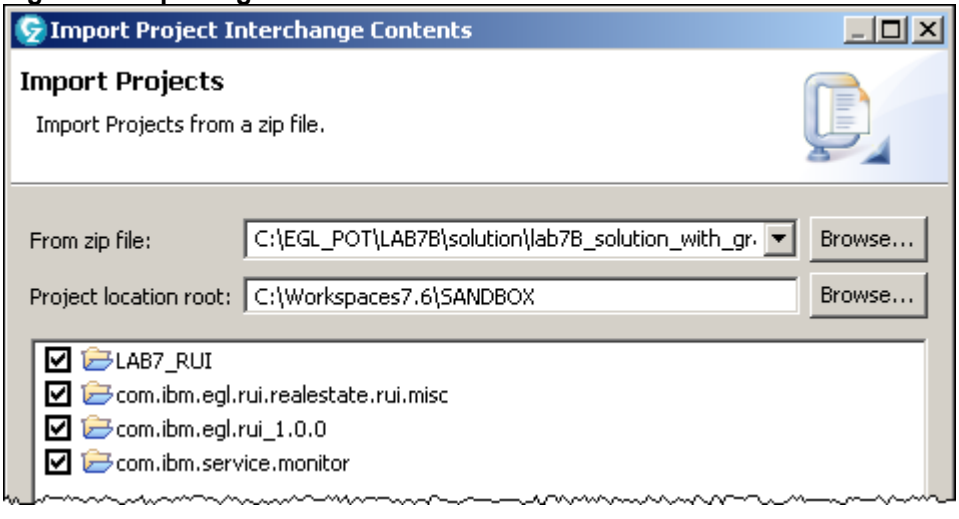
Solution location

If you could not complete the tutorial, do not get frustrated. If you missed one step or selected a bad choice in any of the wizards, you would have problems.

In that case, you can load the solution to your workspace by selecting **File > Import > Project Interchange** and using the solution located at
C:\EGL_POT\LAB7B\ lab7B_solution_with_graphic_better_error_handling.zip.

Select all projects as show in Figure 61 and click **Finish**.
For the tests you need to deploy the EAR file as seen in Section 6..

Figure 61 Importing the solution.



Resources

Learn

Get more information about EGL visiting EGL Café
Find out more about IBM Enterprise Modernization Solutions.
Learn more about IBM Rational Developer for System z.
Watch a demo of Rational Developer for System z.
Read Unleash the power of mainframe assets into SOA.
Visit the Rational page on developerWorks to find technical resources and learn about best practices for the Rational Software Delivery Platform.
Subscribe to The Rational Edge weekly newsletter.
Subscribe to the IBM® developerWorks® newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, Web casts and events.
Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

Get an evaluation version of Rational Developer for System z.
Download trial versions of other IBM Rational software.
Download these IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Tivoli®, and WebSphere®.

Discuss

Join the Architecture forum on developerWorks to get connected with others and take advantage of their expertise and experience to get you tips that can help you as a developer or architect to use the principles of service-oriented architecture (SOA).
Check out developerWorks blogs. and get involved in the developerWorks community.

About the author

Reginaldo W. Barosa is an IBM Certified Application Development Specialist. He provides sales support, helping customers with enterprise transformation solutions and development tools, such as IBM WebSphere Developer for System z. Before joining IBM US, Reginaldo worked for 27 years in IBM Brazil. He has co-authored IBM Redbooks and has written books, as well as other articles for IBM developerWorks. He holds a degree in electrical engineering from Instituto Mauá de Tecnologia, Sao Paulo, Brazil.

