

Create a mortgage portal with EGL Rich UI

Contents

Create a mortgage portal with EGL Rich	
UI	1
Introduction	. 1
Lesson 1: Plan the mortgage application	. 4
Sketch the interface	. 4
List the parts	. 5
Lesson checkpoint	. 7
Lesson 2: Set up the workspace	. 7
Import the portal and dialog widgets	. 7
Create an EGL project	. 7
Get the support files.	. 9
Change vour build path	11
Lesson checkpoint	11
Lesson 3: Create the mortgage calculation Service.	11
Create a Service part	12
Create a Record part	14
Lesson checkpoint	15
Lesson 4: Create the user interface for the calculator	15
Create a Rich UI Handler.	16
Construct the user interface	16
Lesson checkpoint	26
Lesson 5: Add code for the	20
Mortgage Calculator Handler functions	26
Change the Handler code	20
Add the calculate() function	20
Add the showProcessImage() function	27
Add the bideProcessImage() function	21
Add the coloulateMortgage() function.	20
Add the displayBogults() function	20
Add the display Results() function	20
Complete the error display code	28
	29
Lesson checkpoint	29
Lesson 6: Create the CalculationResultsHandler	•
	30
Publish the service results	30
Create the CalculationResultsHandler widget	30
	32
	32
Lesson 7: Create the main portal	33
Create the MainHandler widget	33
lest the portal	34
Lesson checkpoint	35
Lesson 8: Add a calculation history portlet	35
Create the History portlet	36
Lesson checkpoint	37
Lesson 9: Add the calculation history portlet to the	
main portal	37
Change the Results portlet	37
Change the main portal	38
Test the portal	38
Lesson checkpoint	41
Lesson 10: Create the UI for the Map portlet	42
Get a Yahoo! application ID	42
Get a Google Maps key	43
Create the Local Search Interface	43

Add records to the Interface file	44
Create the GoogleMap externalType	45
Create the UI for the MapLocatorHandler widget	46
Lesson checkpoint	48
Lesson 11: Create the source code for the Map	
portlet	48
Finish the source code for	
MapLocatorHandler.egl	48
Test the new portlet	50
Lesson checkpoint	51
Lesson 12: Add the Map portlet to the main portal	51
Change the main portal	51
Test the portal	51
Lesson checkpoint	52
Lesson 13: Install Anache Tomcat	52
Download and install the server	52
Lesson checkpoint	54
Lesson the Deploy and test the mortgage application	55
Edit the deployment descriptor	55
Deploy the Rich III application	55
Deploy the Kich Of application	56
	50
	59
Lesson 15: (Optional) Use validating forms in the	50
	59
Edit the Calculation portlet	59
lest the portlet	63
Test the portal in Preview view	64
Redeploy and test	64
Lesson checkpoint	65
Summary	65
Finished code for MortgageCalculationService.egl	
after Lesson 3	65
Finished code for MortgageCalculatorHandler.egl	
after Lesson 4	66
Finished code for MortgageCalculatorHandler.egl	
after Lesson 5	67
Finished code for CalculationResultsHandler.egl	
after Lesson 6	69
Finished code for MainHandler.egl after Lesson 7 .	69
Finished code for CalculationHistoryHandler.egl	
after Lesson 8	70
Finished code for MainHandler.egl after Lesson 9 .	71
Finished code for IYahooLocalService.egl after	
Lesson 10	72
Finished code for MapLocatorHandler.egl after	
Lesson 10	73
Finished code for MapLocatorHandler.egl after	
Lesson 11	73
Finished code for MainHandler.egl after Lesson 12	75
Finished code for MortgageCalculatorHandler.egl	
after Lesson 15	76
Notices	79
Index	83
	00

Create a mortgage portal with EGL Rich UI

Create a Rich UI portal application that calculates mortgage payments, compares principal to interest, maintains a history of calculations, and maps mortgage businesses based on a U.S. zip code.

Learning objectives

In this tutorial, you will learn how to complete these tasks:

- · Plan the application and design the interface
- Import a custom widget to manage portlets
- Write a service to calculate mortgage payments
- Create a portlet to request input for the calculation service and display the results
- · Create a pie chart to compare total principal to total interest
- Pass data between portlets by using the InfoBus widget
- Create a table that lists all calculations
- · Create a portlet to find mortgage businesses
- Create a Rich UI portal page to contain the individual portlets
- · Install and configure the Apache Tomcat Web server
- Deploy the Web page to the server and test the application
- · Replace the Calculation portlet with a version that uses validating forms

Time required

90 minutes

Other tutorials for Rational[®] EGL Community Edition:

Format a Rich UI logon page

Introduction

Create a Rich UI portal application that calculates mortgage payments, compares principal to interest, maintains a history of calculations, and maps mortgage businesses based on a U.S. zip code.

Portal applications are one of the most popular Web 2.0 user interfaces. These portals collect useful tools in independent areas of the Web page, which are called portlets. The following image shows the portal application that you will create in this tutorial:



Rational EGL Community Edition provides service-oriented architecture (SOA) capabilities. With SOA, you can build applications in which customers interact with Rich UI Web pages while services complete the complicated background calculations.

In this tutorial, you use two services:

- A dedicated EGL service that performs mortgage calculations
- A Web service that lists businesses of a specified type within a specified zip code

In addition, you use an external program to find an address on a map.

You create the dedicated calculation service in this tutorial. A dedicated service is available only to applications that reference the current project. Two protocols are typically used with Web services: SOAP and the Representational State Transfer (REST) protocol. To oversimplify, SOAP is highly extensible, and so allows for great rigor and complexity. REST is more focused on typical HTTP communication. This tutorial uses both protocols.

You can run the application with live data even before you deploy it to a Web project. Deploying the application is a final step that creates the HTML and other files that your customers use to run the application.

Learning objectives

In this tutorial, you will learn how to complete these tasks:

• Plan the application and design the interface

- Import a custom widget to manage portlets
- Write a service to calculate mortgage payments
- Create a portlet to request input for the calculation service and display the results
- · Create a pie chart to compare total principal to total interest
- Pass data between portlets by using the InfoBus widget
- Create a table that lists all calculations
- · Create a portlet to find mortgage businesses
- Create a Rich UI portal page to contain the individual portlets
- · Install and configure the Apache Tomcat Web server
- Deploy the Web page to the server and test the application
- Replace the Calculation portlet with a version that uses validating forms

Time required

This tutorial takes approximately 90 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

You can create the EGL files you need for this application in one of the following ways:

Line by line (most helpful)

Complete the individual lessons to explore the code in small, manageable chunks, learning important keywords and concepts. This method also requires the longest time commitment.

Finished code files

At the end of each lesson in which you create a file, there is a link to the complete code for that file, which you can copy into the EGL editor.

Skill level

Introductory

Audience

This tutorial is designed for people who know the basic concepts of programming, but have little experience with EGL. It is intended for people who work with Rich UI and generate JavaScriptTM.

System requirements

To complete this tutorial, you must have the following tools and components installed on your computer:

- Rational EGL Community Edition Version 1.0.1.
- A working Internet connection

Prerequisites

You do not need any experience with EGL to complete this tutorial. The Format a Rich UI logon page tutorial is a useful introduction, but is not required.

Expected results

You will create a working Rich UI application that calculates mortgages and finds mortgage lenders in a specified area.

Lesson 1: Plan the mortgage application

Design your application on paper before you begin coding.

The first step in planning an application is to list your objectives. This tutorial has the following objectives:

- Calculate mortgage payments based on a total amount, interest rate, and term.
- Display the total interest compared to the total principal in a pie chart.
- Display a history of prior calculations.
- Show a map with the locations of mortgage brokers in a specified zip code.
- Create individual, interactive portlets for the various features of the application.
- Use a combination of dedicated services and external Web services to complete standard tasks.

Sketch the interface

When you create an application, sketch the interface before you write code. Use this sketch as a guide when you create the components of the interface:



List the parts

Determine which EGL parts you need to do the job. For an EGL Rich UI interface, you usually need a combination of Services, Handlers, and widgets.

An EGL Rich UI Handler typically corresponds to a Web page. In this case, you also create Handlers that correspond to portlets. The part is called a Handler because it contains code to handle events that might occur on the page or portlet. Widgets provide content, such as boxes, text, fields, and other screen items. Services complete the behind-the-scenes work.

In this tutorial, you use the following existing components to build the application:

com.ibm.egl.rui.portal

Contains widgets for a portal page and individual portlets within the page. You will import this project in Lesson 2.

com.ibm.egl.rui.infobus

Provides communication between portlets. This package is part of the com.ibm.egl.rui project in IBM[®] Rational EGL Community Edition.

com.ibm.egl.rui.widgets

Contains the following widgets that you use in this tutorial:

- Box
- Combo
- HyperLink
- Image
- TextField
- TextLabel

This package is part of the com.ibm.egl.rui project in the product.

com.ibm.egl.rui.dojo.widgets

Contains the following widgets that you use in this tutorial:

- Button
- DojoGrid
- DojoGridColumn
- DojoPieChart
- PieChartData

The Dojo Toolkit is an open source collection of JavaScript tools. The com.ibm.egl.rui.dojo.widgets project is in the product.

Local Search Web Service

This Web service returns a list of businesses of a specified type within a specified zip code, including the latitude and longitude for each business. Yahoo! hosts the service.

Google Map Widget

This external program calls out a name and address on a map of the surrounding area. Google hosts the program.

In addition, you create the following parts:

MortgageCalculationService

A Web service that calculates monthly payments. Write this service as a dedicated EGL Service so that the code for the Service is part of the same project as your other application files.

MainHandler

The portal page that contains the individual portlets

MortgageCalculatorHandler

The portlet where the application calculates monthly payments

CalculationHistoryHandler

The portlet that displays an interactive list of previous calculations

CalculationResultsHandler

The portlet that displays the pie chart

MapLocatorHandler

The portlet that displays the locations of mortgage providers

Lesson checkpoint

In this lesson, you completed the following tasks:

- · Established the objectives for the application
- Sketched the application interface
- · Listed the required parts

In the next lesson, you download the external files that are required to run the application and create an EGL project to hold them.

Lesson 2: Set up the workspace

Before you write code for the portlets, import the files you need for the tutorial and create an EGL project.

Import the portal and dialog widgets

The portal widget manages both the main portal page and the individual portlet widgets in the page.

The dialog widget provides function calls that create Windows dialog boxes. Use the **showError()** function from the DialogLibrary in this plug-in to display error messages.

- Download com.ibm.egl.rui.portal_version at http://www.ibm.com/ software/rational/cafe/docs/DOC-3433.
- 2. Save the downloaded file on your hard disk.
- 3. Download and save com.ibm.egl.rui.dialog_version at http:// www.ibm.com/software/rational/cafe/docs/DOC-3434.
- 4. When both downloads are complete, import the plug-ins to your workspace. For each downloaded .zip file, complete the following steps:
 - a. From the upper menu in the EGL workspace, click **File** → **Import**.
 - b. In the Import window, expand General and click Existing Projects into Workspace.
 - c. In the next window, for the **Select archive file** field, browse to one of the files you downloaded. The project is displayed in the left pane of the window.
 - d. Click Finish.

Create an EGL project

The *project* is the fundamental unit of organization for EGL source files. It typically corresponds to an application. Within a project, you can create packages if you need greater organization. In this tutorial, you group the various kinds of parts in packages.

To create an EGL project:

1. If you are in a workbench perspective other than EGL, change to the EGL perspective by clicking **Window** → **Open Perspective** → **[Other]** → **EGL**. The perspective icon is in the upper right corner of the workbench.

Ĵ	E EGL	
		- 6

2. Click **File** → **New** → **EGL Project**, or click the **New EGL Project** icon on the menu bar.



- **3**. In the New EGL Project window, enter the following information:
 - a. In the Project name field, type the following name: MortgageEGLProject

🗐 New EGL Project		×
New EGL Project Create a new EGL project		Contraction of the
Project name: Mortgage Project location	EGLProject	
Location: D:\MyData	Norkspace_mortgage_tutorial\MortgageEGLProject Browse	
0	<back next=""> Einish Cancel</back>	

b. Click Finish.

EGL creates a project named MortgageEGLProject. Note the two folders inside the directory:

- One for EGL source code
- One for the Web content that you create



Get the support files

This tutorial provides several files that belong in the WebContent directory that EGL created:

- A cascading style sheet (CSS) file to format the finished Web page
- An animated GIF to indicate background activity in progress
- The JavaScript file for the Google Maps application

To add these support files to your application:

- Download the MortgageSupport.zip file to location that is easy to remember, such as your Desktop. You can download the file at http://www.ibm.com/ software/rational/cafe/docs/DOC-3435
- 2. From inside the workbench, import the contents of the archive:
 - a. In the Project Explorer view, select **MortgageEGLProject**. From the upper menu in the EGL workspace, click **File** → **Import**.
 - b. In the Import window, expand General, click Archive File, and click Next.
 - c. In the Archive File window, for the From archive file field, browse to the directory where you downloaded the archive file and click MortgageSupport.zip. The top level of the archive is displayed in the left pane of the window. Expand the directory structure to see the contents.

🗐 Import	
Archive file Import the contents of an archive file in zip or tar format from the	local file system.
From archive file: D:\temp\MortgageSupport.zip	Browse
Es icons E Ø icons E Ø icons	
Filter Types Select All Deselect All	
Into folder: MortgageEGLProject	Browse
(?) < Back Nex	t > Finish Cancel

- d. Because **MortgageEGLProject** was selected when you started the Import wizard, that project is displayed by default in the **Into folder** field.
- e. Click Finish.
- f. When an error message appears asking permission to overwrite the MortgageEGLProject.css file, click Yes. The default, empty style sheet is replaced with the downloaded version.

Overwrite 'i	MortgageEGLProje	ct.css' in folde	er	
MortgageE	GLProject\WebCor	ntent\css'?		

Change your build path

The EGL build path determines the projects that EGL examines when trying to resolve references in your programs. To add the projects you just downloaded to your build path:

- 1. Right click MortgageEGLProject in the Project Explorer view, then click Properties. On the left side of the Properties for MortgageEGLProject window, click **EGL Build Path**. EGL displays a list of the projects in your workspace.
- 2. Select the **com.ibm.egl.rui.portal** project and the **com.ibm.egl.rui.dialog** project. The finished build path window should look like the following image: These selections mean that when you organize imports, EGL will look in all of

type filter text	EGL Build Path
Resource BeanInfo Path Builders EGL Build Path EGL Default Build Descriptors	Select the check box for each project in the EGL build path:
Java Build Path Java Code Style Java Compiler Java Editor Java Editor Javadoc Location Project References Refactoring History Pun/Debug Settings	 ✓ ☞ com.ibm.egl.rui_1.0.3 ✓ ☞ com.ibm.egl.rui.dialog_1.0.1 ✓ ☞ com.ibm.egl.rui.dojo.runtime.google_1.4 □ ☞ com.ibm.egl.rui.dojo.samples_0.8 □ ☞ com.ibm.egl.rui.dojo.widgets_0.8 ✓ ☞ com.ibm.egl.rui.portal_1.0.2 □ ☞ com.ibm.egl.rui.samples.ce

the selected projects to resolve references.

3. Click OK.

Related reference

EGL build path

Lesson checkpoint

In this lesson, you completed the following tasks:

- · Imported the portal widget
- · Imported the dialog widget
- Created an EGL project for the Mortgage application
- · Imported support files for the application
- · Adjusted your build path

In the next lesson, you create a dedicated service to calculate a monthly mortgage payment.

Lesson 3: Create the mortgage calculation Service

Create a dedicated service to calculate monthly payments.

In this lesson, you create an EGL Service part. The *part* is a central concept in EGL, and the term means what it seems to: An EGL part is one separable piece of an EGL application. The Record, the Program, and the Library are examples of parts.

In EGL, *generatable parts* are the pieces of an EGL application that can be generated separately. You must place each generatable part in a separate source file, and the name of the part must be the same as the name of the file. The Service is a generatable part.

Like a Library, the Service contains functions that you call from other programs. You will use the Service you create in this lesson as a dedicated Service, which means that it is available only to applications that reference the current project. For a dedicated Service, you do not need to create an Interface part as you do for a Web Service.

Create a Service part

- In the Project Explorer window, right-click MortgageEGLProject, then click New → Service.
- 2. In the New EGL Service Part window, enter the following information:
 - a. In the EGL source file name field, enter the following name: MortgageCalculationService

EGL adds the .egl file extension.

- b. In the Package field, enter the following name: services
- c. Verify that **Create as Web (SOAP) service** and **Create as Web (REST) service** are cleared. You are creating a dedicated service, which does not implement an Interface, so make sure that the **Implements Interfaces** field is empty.

New EGL Service	part.		E
Source fol <u>d</u> er:	MortgageEGLProject\EGLSource	Browse	
Package:	services	Bro <u>w</u> se	
Service name: Implements Interfaces:	MortgageCalculationService		<u>A</u> dd
			Remove
	AP) service		
Create as web (SU			

- 3. Click **Finish**. EGL opens the new Service part in the editor.
- 4. Remove the boilerplate code from the file, leaving only the following lines:

package services;

service MortgageCalculationService

end

5. Add the following function before the end statement:

```
function amortize(inputData MortgageCalculationResult inOut)
amt MONEY = inputData.loanAmount;
// convert to monthly rate
rate DECIMAL(10, 8) = (1 + inputData.interestRate / 1200);
// convert to months
term INT = (inputData.term * 12);
// calculate monthly payment amount
pmt MONEY = (amt * (rate - 1) * Mathlib.pow(rate, term)) /
(MathLib.pow(rate, term) - 1);
totalInterest MONEY = (pmt * term) - amt;
// update result record
inputData.interest = totalInterest;
end
```

When you paste code from these instructions, the formatting may change. Press Ctrl+Shift+F to reformat the code. You can change the formatting rules by clicking **Window** \rightarrow **Preferences** \rightarrow **EGL** \rightarrow **Editor** \rightarrow **Formatter**.

Note:

EGL marks any code errors with a red X in the left margin and a wavy red line under the error. Move your cursor over the X to see an error message.

```
package services;
  service MortgageCalculationService
    IWN.VAL.3260.e 5/30 The type MortgageCalculationResult cannot be resolved. on Result in Out)
            amt MONEY = inputData.loanAmount;
            // convert to monthly rate
0
            rate DECIMAL(10, 8) = (1 + inputData.interestRate / 1200);
            // convert to months
0
            term INT = (inputData.term * 12);
            // calculate monthly payment amount
            pmt MONEY = (amt * (rate - 1) * Mathlib.pow(rate, term)) /
                     (MathLib.pow(rate, term) - 1);
            totalInterest MONEY = (pmt * term) - amt;
            // update result record
0
            inputData.monthlyPayment = pmt;
0
            inputData.interest = totalInterest;
        end
    end
```

Because you have not yet defined a type named MortgageCalculationResult, EGL cannot create the inputData variable based on that type. When you create this Record type in the next exercise, EGL will remove the error markers from the display.

6. Save the file by clicking **File** → **Save** or by pressing Ctrl-S.

```
Related concepts
```

Introduction to Service parts

Create a Record part

The amortize() function uses a MortgageCalculationResult record. You can define this record in the same file as the Service.

To create the Record part:

 Add the following code after the amortize() function in the MortgageCalculationService.egl file. The Record is a part, so you define it outside the Service part, that is, after the final end statement in the file:

```
record MortgageCalculationResult
    // user input
    loanAmount MONEY;
    interestRate DECIMAL(10,8);
    term INT;
    // calculated fields
```

```
// calculated fields
monthlyPayment MONEY;
interest MONEY;
end
```

2. Save the file. EGL should not display any more error markers in the code. If you see errors in your source file, compare your code to the file contents in "Finished code for MortgageCalculationService.egl after Lesson 3" on page 65. As you work through the tutorial, it is possible that you might see red Xs next to the project or next to one of the folders below it, yet not see any errors in the file itself.



If you encounter this situation, resolve it by clicking **Project** → **Clean**. In the Clean window, click **Clean projects selected below** and then click **MortgageEGLProject**.

Clean all projects	Clean projects selected below
☐ 🥵 com.ibm.eal.rui 1.0.	3
com.ibm.eql.rui.dialo	g 1.0.1
com.ibm.eql.rui.dojo	runtime.google 1.4
🛛 🕮 com.ibm.egl.rui.dojo	.samples_0.8
🛾 🕮 com.ibm.egl.rui.dojo	.widgets_0.8
🛛 🕮 com.ibm.egl.rui.port	al_1.0.2
] 进 com.ibm.egl.rui.samı	oles.ce
🛛 🕮 MortgageEGLProject	

Click **OK**. EGL rebuilds the selected project and the red X is removed from the Project Explorer view.

3. Close the file by clicking the X next to the file name in the tab at the top of the editor or by clicking **File** → **Close**.

Lesson checkpoint

You learned how to complete the following tasks:

- Create an EGL Service part
- Create an EGL Record part and add it to the source file for the Service
- Check for errors in your code

In the next lesson, you create the user interface for the first application portlet.

Lesson 4: Create the user interface for the calculator

Each portlet on the finished page is controlled by an EGL Rich UI Handler part.

EGL Rich UI follows the Visual Formatting Model of the World Wide Web Consortium (W3C). This recommendation includes concepts such as containing boxes, default positioning, and the flow of objects on the page. For more information, see http://www.w3.org/TR/CSS2/visuren.html.

In this lesson, you build the page without using absolute positioning of the boxes. With relative positioning, the page can adapt more easily to different screen resolutions, browsers, fonts, and other factors that affect display. Use the EGL visual editor to drag Rich UI components for the calculator to the portlet. If you are not familiar with EGL Rich UI, you might want to complete the Format a Rich UI logon page tutorial.

Create a Rich UI Handler

 In the MortgageEGLProject folder, right-click the EGLSource folder. Click New → Rich UI Handler, or click the New Rich UI Handler button on the menu bar.

📒 EGL - E	GL Comm	unity Ed	ition™		
File Edit	Navigate	Search	Project	Run	Window
] 👩 🔹 !	a 🖻]	f i	N] 3	ijz. →	0 - 0
Projec	t Explorer	x	New Ri	ch UI	Handler

- 2. In the New Rich UI Handler part window, enter the following information:
 - a. For EGL source file name, enter the following name: MortgageCalculatorHandler
 - b. For **Package**, enter the following name: handlers
 - c. Click Finish.

The new Handler opens in Design view in the EGL editor. EGL creates the handlers package for you in the EGLSource folder.

Construct the user interface

The user interface (UI) for the Calculator portlet collects the information required by the mortgage calculation service:

- The cost of the property
- The interest rate for the loan
- The length of the mortgage

To construct the UI for the calculator:

 In Design view, go to the Palette view, located by default to the right of the EGL editor. You should see palettes available for both EGL Widgets and Dojo Widgets. If you do not see the Dojo Widgets palette, click the Refresh palette button to the left of the Palette view.



2. Make sure the initial ui Box is selected. Selected objects are surrounded by a dotted line. Locate the **Properties** view, in the lower left of the workspace by default. It shares space with the **Outline** view, so you might need to click **Properties** to see the contents. Enter the following value for the **id** property: form

 -

The cascading style sheet (CSS) that you imported as part of the MortgageSupport.zip file contains the following entries for form:

```
#form input {
  font-family: verdana, tahoma, sans-serif;
}
#form td {
  padding: 4px;
}
```

The second specification, for a table cell, ensures that the various widgets that you place in this box will be separated by enough space to make them easy to read.

3. In the **EGL Widgets** palette, scroll down until you see the TextLabel widget. Drag a TextLabel widget onto the Design interface in the EGL editor.

When the entire ui box turns green, you can drop the widget on the interface.

	N
ß	Box (ui)
	P
ign Source Preview	

Note the thin yellow area at the top of the screen. If you dragged the widget there, it would be displayed outside and above the ui box. When you release the mouse button, the New Variable window opens.

 In the New Variable window, enter the following name: amountLabel

100			
Variable name: am	ountLabe	1	
✓ Prompt for a	variable r	name	
You may also	change t	his setting ir	n the preferences
		ок	Cancel

- 5. Click OK.
- 6. Make sure that the new amountLabel widget is still selected. In the Properties view, enter the following label name for the text field: Amount:

Properties Ev	vents
text	Amount:
id	
class	
color	
backgroundCo	olor
font	[
fontSize	
font///eight	(none)

7. Drag a TextField widget onto the UI to the right of the label.

22	Box (ui)

- Give this field the following name: amountField
- 9. Click OK.
- Change the text property of the amountField widget to the following value, which becomes the default value for the field: 180000

Do not include comma separators or a decimal point.

- 11. Create a label and field for the mortgage rate:
 - a. Add another TextLabel widget and assign it the following name: rateLabel

Because the mainBox widget where you are placing these labels and fields defaults to two columns, EGL automatically places the rateLabel widget on a new row.

- b. Change the **text** property of the rateLabel widget to the following label: Rate:
- Add a TextField widget with the following name: rateField

- d. Change the text property of the rateField widget to the following value, which becomes the default value for the field:
 5.2
- 12. Create a label and combo box for the mortgage term:
 - a. Add a TextLabel widget with the following name: termLabel

EGL places the widget on a new row.

- a. Change the text property to the following label: Term:
- b. Drag a Combo widget onto the line next to the termLabel widget.
- c. Give the combo the following name: termCombo
- d. In the Properties view, click the ellipsis (...) next to the **values** property. The values window opens.
- e. Select Combo and click Remove.

		Add
ombo		Remove
		Move up
		Move dow

- f. Type the following number in the **Add** field:
 - 5
- g. Click Add.
- h. Add each of the following numbers, clicking Add after each addition:
 - 10
 - 15 30

The values window should now look like the following image:

	Add
5	Remove
15 30	Move up
	Move down

- i. Click OK.
- j. Change the **selection** property of the termCombo widget to the following value, which becomes the default value for the field:

4

This number indicates the fourth item in the list, which is the number 30.

- **13**. Create a submit button and bind it to a stub function:
 - a. Drag another text label onto the UI to create a blank space, and assign the following name:

blankLabel

- b. In the Properties view, clear the text field.
- Drag a Box widget into the next available position and assign the following name to the variable.
 buttonBox

Use this box to hold your submit button and a graphic that indicates that a calculation is in process.

d. From the Dojo Widgets palette, drag a Button onto the buttonBox widget. Use the Dojo button because it is more stylish than the EGL version.

Rate: Us.2	 Box (ui)
Term 30 V	Box (buttonBox)

- e. Give the button the following name: calculationButton
- f. In the Properties view, change the **text** property to the following label:

Calculate

g. On the **Events** tab, select the onClick event in the column on the left. At the far right end of that row, click the plus sign to add a new function.

🗄 Outline 🔲 Prope	rties 🛛	~
🛄 DojoButton (calcu	lationButton)	
Properties Events		<u>*</u>
Event	Function	
onChange		
onClick		
onSelect	1	4
onFocusGained		
onFocusLost		
onKeyDown		

- In the New Event Handler window, enter the following name for the new function:
 calculate
- i. Click **OK**. EGL changes to the Source view and creates a stub function, that is, a function with no code, at the end of the file.

```
function start()
end
```

function calculate (event Event in)

end

end

- j. Click **Design** to return to Design view. The calculate function name is now displayed next to the onClick event. The function is bound to the button, so that when you click the **Calculate** button, EGL calls the calculate() function. You will add the code for this function later.
- 14. Add an animated image to indicate calculation in process.
 - a. Drag an Image widget from the EGL Widgets palette to the buttonBox widget.
 - b. Give the image the following name: processImage
 - c. In the Properties view, on the Properties page, assign a source for the image in the src field: icons/progress3.gif

This is the image you imported from the MortgageSupport.zip archive.

- d. Also in the Properties view, expand **Appearance**. From the list of options for the **visibility** field, select **hidden**. The image remains hidden until the **Calculate** button is clicked.
- 15. Create a field to hold the result of the calculation:
 - a. Drag a TextLabel to the next row on the screen. Place this label *outside* the buttonBox widget.



- b. Assign the following name: paymentLabel
- c. In the Properties view, add the following label for the text property: Payment:
- Drag a TextField widget to the next position on the screen and assign the following name:

paymentField

- 17. Drag an HTML widget to the next position on the screen.
 - Assign the following name: errorDisplay

This widget creates a text area where you can display error messages. Error messages are a normal part of processing and often indicate minor problems like using a comma separator when entering the Amount field.

- b. With the new errorDisplay widget selected, open the Properties view and clear the **text** property.
- c. Click the button next to **color**.

Properties	Events	-
src	I	
text	[
id	[
class	[
color		
background	Color	
	Press to	select a color

d. From the Color selection window, click **Name format** and scroll through the list of colors until you can click **Red**.

Color	Format G RGB G Hexadecimal
Name	format
	PaleGreen
	PaleTurquoise
	PaleVioletRed
	PapayaWhip
	PeachPuff
	Peru
	Pink
	Plum
	PowderBlue
	Purple
-	Red
	DevelPhus
	SaddleBrown
	Salmon
]	
Custo	m

e. Click OK.

f. Save the file.

The completed interface should look like the following image:

	1 0
Г	٦
L	<u>د</u>
]
	r L

You will develop the rest of the portlet in Source view. Click the **Source** tab at the bottom of the editor pane. EGL created the code in the **Source** tab based on your

actions in the EGL visual editor. The code should match the file contents in "Finished code for MortgageCalculatorHandler.egl after Lesson 4" on page 66.

Lesson checkpoint

You learned how to perform the following tasks:

- Create a Rich UI Handler
- Use the EGL visual editor to create a user interface
- Use the Properties view to format the interface
- Use CSS to format the interface

In the next lesson, you add code to support the interface that you created in this lesson.

Lesson 5: Add code for the MortgageCalculatorHandler functions

Add functions in the MortgageCalculatorHandler part to support the user interface that you constructed in the previous lesson.

In this lesson, you will work exclusively with EGL source code.

The following exercises show the EGL code for each of the functions in MortgageCalculatorHandler.egl, with brief explanations of how the code works. Add each function, in order, before the final **end** statement in the file.

For the moment, ignore the start() function. You will add code in a later lesson so that this portlet can communicate with other portlets on the page.

This lesson uses the EGL Model-View-Controller (MVC) framework to manage errors. These functions are included in the com.ibm.egl.rui plug-in. MVC is a way of thinking about the various components that are involved in presenting data in a user interface. At its simplest, the *model* is a representation of data, such as a data field, the *view* is the user interface, and the *controller* is the mechanism that defines the relationship between the model and the view. You create an EGL Controller widget to associate the data in a model with the view, in this case, a screen element.

Change the Handler code

You must make a few minor changes to the main Handler section. Use the EGL MVC framework to handle any errors that occur during calculation.

- In the Design view for MortgageCalculatorHandler.egl, select the errorDisplay HTML widget. At the bottom of the editor, click the Source tab. The source editor opens with the cursor at the beginning of the declaration for errorDisplay.
- 2. Add the following lines above the errorDisplay declaration:

// use for error messages
error STRING = "";

This string holds any error messages that the Handler receives.

3. Add the following lines below the errorDisplay declaration:

// associate MVC with errorDisplay widget
errorController Controller{@MVC{model = error, view = errorDisplay}};

You are associating the built-in MVC framework with the errorDisplay widget. You will use this errorController widget in the setError() function later in this lesson.

4. Below the main declaration for the Handler, declare a variable to represent the service:

mortService MortgageCalculationService{};

You can use this variable to call the service within the program.

- 5. Remove the default **width** and **height** measurements from the ui Box declaration. Both fields are initially set to 400 pixels.
- 6. For a cleaner look, you can delete the comments (lines beginning with "//") that precede the main Handler declaration. These comments are part of the boilerplate Handler code that EGL creates.

Add the calculate() function

This function calls two other functions. The first makes the process GIF visible in order to indicate that the application is working. The second calls the service to complete the calculation.

To code the function in the EGL editor, paste the following lines to replace the calculate() stub function that you created in the last lesson:

```
function calculate(event Event in)
  showProcessImage();
  calculateMortgage();
end
```

Note: You do not need to provide arguments for the calculateMortgage() function. The necessary information is global to the Handler program. Before you call the service, you will place the necessary information in a record that the service function uses as an argument.

Add the showProcessImage() function

This function makes the animated processing GIF visible. To code the function in the EGL editor, paste the following lines before the final **end** statement:

```
function showProcessImage()
processImage.visibility = "visible";
end
```

Note: The visibility property is part of any Image widget.

Add the hideProcessImage() function

Because you have a function to make the processing GIF visible, you now need a function to make it invisible again. To code the function in the EGL editor, paste the following lines:

```
function hideProcessImage()
processImage.visibility = "hidden";
end
```

Note: The visibility property is part of any Image widget.

Add the calculateMortgage() function

This function calls a service to complete the calculation based on the values displayed in the UI. To code the function in the EGL editor, paste the following lines:

```
function calculateMortgage()
    // new copy of the input record
    inputRec MortgageCalculationResult{};
    // load with values from the ui
    inputRec.loanAmount = amountField.text as MONEY;
    inputRec.interestRate = rateField.text as DECIMAL(10,8);
    inputRec.term = termCombo.values[termCombo.selection] as INT;
    call mortService.amortize(inputRec) returning to displayResults
        onException handleException;
end
```

Note:

- inputRec is a local variable that is based on the MortgageCalculationResult record in the MortgageCalculationService source file. Passing a record that contains the necessary parameters is a common way of communicating with a Web service. The service then places its results in that same record.
- The EGL as operator casts one type as another. In this case, it converts text fields to numeric ones.
- **3**. The **call** statement here is a variation used with services only. It has the advantage of being asynchronous, so that the UI does not freeze while waiting for the service to respond.

Add the displayResults() function

This function adds a dollar sign to the result of the calculation, and hides the processing GIF. To code the function, paste the following lines before the final **end** statement:

```
function displayResults(retResult MortgageCalculationResult in)
paymentField.text = "$" + retResult.monthlyPayment as STRING;
hideProcessImage();
end
```

Complete the error display code

1. Before the final end statement, add the following functions:

```
private function setError(err STRING in)
error = err;
errorController.publish();
end
```

```
// catch-all exception handler
private function handleException(ae AnyException in)
setError("Error calling service: " :: ae.message);
end
```

Functions with the **private** modifier cannot be called outside the current program. The setError() function uses the errorController widget you created earlier.

- 2. Right-click an empty space in the editor. Click **Organize imports**. EGL adds **import** statements for all the undefined symbols that it can. If you see errors in your source file, compare your code to the file contents in "Finished code for MortgageCalculatorHandler.egl after Lesson 5" on page 67.
- 3. After you resolve any errors, save the file.

Test the calculator

You are now ready to test your first portlet.

- 1. Change to Preview view by clicking the **Preview** tab at the bottom of the editor. You can fully test your application in the Preview view, including services, databases, and the user interface. EGL displays the default values that you entered when you created the UI.
- 2. Click **Calculate**. EGL displays the monthly payment. Because you are using a local service, the calculation might be too fast for you to see the processing GIF.

Amount:	180000
Rate:	5.2
Term:	30
	Calculate
Payment:	\$988.40

3. Change the values for any of the three fields and click **Calculate** again. The Payment field changes accordingly.

Lesson checkpoint

You learned how to complete the following tasks:

- · Work in source mode in the EGL editor
- Use EGL Model-View-Controller functions
- Call an EGL Service inside a function

In the next lesson, you create a pie chart to compare the total principal to the total interest for a given calculation.

Lesson 6: Create the CalculationResultsHandler widget

Add a second portlet on the page to hold a pie chart that shows the relative proportions of principal and interest.

The CalculationResultsHandler does not perform calculations or make service calls. Instead, it relies on messages from the MortgageCalculatorHandler widget that you created in the previous lessons. The mechanism that you use to send and receive messages is the InfoBus library, which is part of the com.ibm.egl.rui plug-in.

To tell EGL that you want to receive messages about specific events, you call the InfoBus.subscribe() function. To send a message, you call the InfoBus.publish() function. In this lesson, you add the publish() function to the MortgageCalculatorHandler widget. Then you add the subscribe() function to a new Handler, which uses the information to create a pie chart.

Publish the service results

1. In the MortgageCalculatorHandler.egl file, find the displayResults() function that you created in the previous lesson. Add the following line before the **end** statement:

InfoBus.publish("mortgageApplication.mortgageCalculated", retResult);

The function now consists of the following code:

```
function displayResults(retResult MortgageCalculationResult in)
paymentField.text = "$" + retResult.monthlyPayment as STRING;
hideProcessImage();
InfoBus.publish("mortgageApplication.mortgageCalculated", retResult);
end
```

This code shows the payment amount in the payment field, hides the processing image, and publishes the results of the calculation in the retResult record. The information in that record is available to any widget that subscribes to the mortgageApplication.mortgageCalculated event.

Note: The names of these events are case-sensitive, so "mortgageApplication" is a different event than "MortgageApplication".

2. Organize imports to resolve the InfoBus label, then save and close the file.

Related reference

Rich UI Infobus

Create the CalculationResultsHandler widget

- 1. Create a Rich UI Handler in the handlers package, as you did in "Lesson 4: Create the user interface for the calculator" on page 15.
 - a. Give the Handler the following name:
 - CalculationResultsHandler b. Place the Handler in the following package:

handlers

The Handler opens in the Design view of the EGL editor.

 Drag a PieChart widget from the Dojo Widgets palette onto the initial uiBox widget for the Handler, and give the widget the following name: interestPieChart





- **3**. Change to Source view to complete the rest of the lesson. At the bottom of the editor, click the **Source** tab.
- 4. In the ui Box declaration, reduce the **height** property of the Box to 300 pixels. This adjustment creates more space for the other portlets.
- 5. You need only two PieChartData records. Replace the data field of the interestPieChart widget with the following code:

```
data = [
  new PieChartData{y=1, text="Principal", color="#99ccbb"},
  new PieChartData{y=0, text="Interest", color="#888855"}
]};
```

The y field contains the amount that is compared to the amounts in the other PieChartData records. You are setting up an initial display of 100 percent principal. This display is a placeholder that the application uses until the first calculation.

6. Set up your communication with the MortgageCalculatorHandler widget by adding the following line to the start() function:

InfoBus.subscribe("mortgageApplication.mortgageCalculated", displayChart);

This code calls the displayChart() function whenever the specified event occurs. You added this event to the MortgageCalculatorHandler widget in the previous exercise.

7. Before the final end statement, add the displayChart() function:

```
function displayChart(eventName STRING in, dataObject ANY in)
localPieData PieChartData[];
localPieData = interestPieChart.data;
resultRecord MortgageCalculationResult = dataObject as MortgageCalculationResult;
```

```
localPieData[1].y = resultRecord.loanAmount;
localPieData[2].y = resultRecord.interest;
interestPieChart.data = localPieData;
end
```

You receive the data record through the InfoBus subscription, as the dataObject parameter of the function.

To force the pie chart to refresh, you must set the data array equal to a new value. Here you create a new array of PieChartData Records and give them initial values from the interestPieChart widget. Then you update this new array with values from the incoming MortgageCalculationResult Record type. When you set the interestPieChart.data array equal to the new array, the pie chart on the screen is refreshed.

 Organize your imports and save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for CalculationResultsHandler.egl after Lesson 6" on page 69.

Test the pie chart

1. Change to Preview view. EGL displays a default pie chart showing 100% principal.



2. Close the file.

Lesson checkpoint

You learned how to complete the following tasks:

- Use the InfoBus widget to pass information between portlets
- Create a pie chart widget

In the next lesson, you add the two new portlets to a main portal page.
Lesson 7: Create the main portal

The main page uses the EGL portlet widgets to manage communication between the parts of the application.

Create the MainHandler widget

- 1. Create a Rich UI Handler in the handlers package, as you did in "Lesson 4: Create the user interface for the calculator" on page 15.
 - a. Give the Handler the following name:
 - MainHandler
 - b. Place the Handler in the following package: handlers

The Handler opens in the Design view of the EGL editor.

- 2. Code the main handler in Source view. At the bottom of the editor, click the **Source** tab.
- 3. Make the following changes to the declaration of the ui Box:
 - a. Remove the default width and height measurements.
 - b. Add a portal widget named mortgagePortal to the **children** property. You will declare this widget in the next step.
 - c. The revised declaration should look like the following line:

ui Box{ columns = 2, children = [mortgagePortal] };

4. Add the following line below the ui Box definition:

```
mortgagePortal Portal { columns = 2, columnWidths = [ 300, 650 ] };
```

5. Skip a line and add the following code:

```
calculatorHandler MortgageCalculatorHandler{};
resultsHandler CalculationResultsHandler{};
```

These lines declare variables that represent the two new widget types that you created in the previous lessons.

6. Skip a line and add the following code:

```
calculatorPortlet Portlet{children = [calculatorHandler.ui],
    title = "Calculator"};
resultsPortlet Portlet{children = [resultsHandler.ui],
    title = "Results", canMove = TRUE, canMinimize = TRUE};
```

Each new portlet variable is declared with the initial UI for the corresponding widget.

7. Add the two portlet variables to the main portal in the start() function:

```
function start()
mortgagePortal.addPortlet(calculatorPortlet, 1);
mortgagePortal.addPortlet(resultsPortlet, 1);
```

```
// Subscribe to calculation events
InfoBus.subscribe("mortgageApplication.mortgageCalculated", restorePortlets);
```

```
// Initial state is minimized
resultsPortlet.minimize();
end
```

As you did previously, you subscribe to the mortgageCalculated event in the MortgageCalculatorHandler widget to trigger an action. In this case, you call the restorePortlets() function. The main portal minimizes the Results portlet, which contains the pie chart, until you complete a new calculation.

8. Add the following function to manage the layout of the portlets:

```
function restorePortlets(eventName STRING in, dataObject ANY in)
if(resultsPortlet.isMinimized())
resultsPortlet.restore();
end
end
```

The restore() function is automatically available for any variable that is based on the portlet type.

9. Organize your imports and save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for MainHandler.egl after Lesson 7" on page 69.

Test the portal

Test the main portal to make sure that the results Portlet receives changes from the Calculation portlet.

1. At the bottom of the editor, click **Preview**. EGL displays the main portal and the two subsidiary portlets.

Calculate	or	- 🖸
Amount:	180000	
Rate:	5.2	
Term:	30 💌	
	Calculate	
Payment		
Results		년 🖸

2. Click **Calculate**. The animated GIF that indicates that processing is in progress is displayed. When the calculation finishes, the pie chart is displayed.

	or	0 10
Amount: Rate: Term:	180000 5.2 30 💌	
Payment	Calculate	
Results		- 8
	Interest Principal	

- **3**. Change any of the calculation values and click **Calculate** again. The pie chart reflects changes in the proportion of principal to interest.
- 4. Close the file.

Lesson checkpoint

You learned how to complete the following tasks:

- Create a portal widget
- Add the portlets that you created in previous lessons to the portal

In the next lesson, you add a portlet to record the calculations you perform.

Lesson 8: Add a calculation history portlet

Create a table where you can click a row to display a previous calculation.

In this lesson, you use the DojoGrid widget to create a table. Although the Grid widget is available in the **Dojo Widgets** palette, no properties are accessible in the Design view, so you code the widget in Source view.

Create the History portlet

- 1. Create a Rich UI Handler in the handlers package, as you did in "Lesson 4: Create the user interface for the calculator" on page 15.
 - a. Give the Handler the following name: CalculationHistoryHandler
 - b. Place the Handler in the following package: handlers

The Handler opens in the Design view of the EGL editor.

- 2. Code the calculation history Handler in Source view. At the bottom of the editor, click the **Source** tab.
- 3. Make the following changes to the declaration of the ui Box:
 - a. Remove the default width and height measurements.
 - b. Add a grid widget named historyGrid to the **children** property. You will declare this widget in the next step.
 - c. The revised declaration should look like the following line:

```
ui Box{ columns = 2, children = [ historyGrid ] };
```

4. Add the following declaration below the ui Box declaration:

```
historyGrid DojoGrid { behaviors = [ addSelectionListener ],
headerBehaviors = [ ],
columns = [
new DojoGridColumn { displayName = "Principal", name = "loanAmount", width = 60 },
new DojoGridColumn { displayName = "Rate", name = "interestRate", width = 60 },
new DojoGridColumn { displayName = "Years", name = "term", width = 60 },
new DojoGridColumn { displayName = "Payment", name = "monthlyPayment", width = 60 } ] };
```

A DojoGrid is a table with interactive capabilities. Note the addSelectionListener behavior. This code refers to a function that triggers an event when you click a row in the table. You will add that function later in this lesson.

5. Skip a line and add the following code:

// array to store calculation results
historyResults MortgageCalculationResult[0];

This code declares an array of the same Record type that you used in the other portlets to store the results of your calculations. This array stores each set of results after it is calculated.

6. Add the following code to the stub of the start() function:

// Subscribe to calculation events so history table (grid) can be updated
InfoBus.subscribe("mortgageApplication.mortgageCalculated", addResultRecord);

As before, you use InfoBus to notify the program when there is a new calculation. EGL responds by calling the addResultRecord() function.

7. Add the addResultRecord() function below the start() function:

// Update grid to show latest mortgage calculation

function addResultRecord(eventName STRING in, dataObject ANY in)

resultRecord MortgageCalculationResult = dataObject as MortgageCalculationResult;

```
historyResults.appendElement(resultRecord);
```

```
historyGrid.data = historyResults as ANY[];
end
```

As you did previously, you cast the incoming dataObject as a MortgageCalculationResult record. You append the new results to the array, and then replace the entire data field in the table widget. As you have seen before, replacing the data field causes the widget to refresh.

8. Add the following two functions to manage the selection of a calculation:

```
// Adds a listener to each cell
function addSelectionListener(grid DojoGrid in, cell Widget in,
   row ANY in, rowNumber INT in, column DojoGridColumn in)
   cell.setAttribute("row", rowNumber);
   cell.onClick ::= cellClicked;
end
// Publish event to InfoBus when previous calculation is selected
function cellClicked(e Event in)
   try
   row int = e.widget.getAttribute("row") as INT;
   InfoBus.publish("mortgageApplication.mortgageResultSelected", historyResults[row]);
   onException(ex AnyException)
   end
end
```

The Grid widget calls the addSelectionListener() function for each cell it creates. This function saves the row number as a free-form attribute of the cell, to be read later. The function also specifies a second function, cellClicked(), that is called when the user clicks the cell. The cellClicked() function reads the saved row number and then uses the InfoBus to publish the MortgageCalculationResult record associated with row.

- **9**. Organize your imports and save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for CalculationHistoryHandler.egl after Lesson 8" on page 70.
- 10. Close the file.

Lesson checkpoint

You learned how to complete the following tasks:

- Create a portlet that contains a DojoGrid table
- Use the behaviors of the table to trigger an event when a cell is clicked
- Add a results record after an InfoBus alert

In the next lesson, you add this portlet to the main portal.

Lesson 9: Add the calculation history portlet to the main portal

To add the new portlet to your page, you must change the Results portlet and the main portal.

Change the Results portlet

As of the end of the previous lesson, the CalculationResultsHandler widget checks for only a single event, mortgageApplication.mortgageCalculated. Now you also want to redisplay the pie chart when a row in the History portlet is

selected. That action generates the mortgageApplication.mortgageResultSelected event. You can use the asterisk wildcard character (*) to represent either event.

- In the EGL editor, open the CalculationResultsHandler.egl file and switch to Source view.
- In the start() function, find the following line: InfoBus.subscribe("mortgageApplication.mortgageCalculated", displayChart);
- Replace the specific event name with a wildcard character: InfoBus.subscribe("mortgageApplication.*", displayChart);

EGL now calls the displayChart() function whenever an event that begins with mortgageApplication. is published to the InfoBus.

4. Save and close the file.

Change the main portal

Add lines for the History portlet that are similar to the lines for the other two portlets:

- 1. In the EGL editor, open the MainHandler.egl file and click the **Source** tab.
- Immediately below the resultsHandler declaration, add a similar declaration for historyHandler:

historyHandler CalculationHistoryHandler{};

3. Immediately below the resultsPortlet declaration, add a similar declaration for historyPortlet:

```
historyPortlet Portlet{children = [historyHandler.ui],
    title = "History", canMove = TRUE, canMinimize = TRUE};
```

4. In the start() function, below the existing calls to addPortlet(), add the new portlet to the portal:

mortgagePortal.addPortlet(historyPortlet, 1);

5. As you did with the resultsPortlet, set the historyPortlet as initially minimized:

historyPortlet.minimize();

6. Add code for the historyPortlet to the end of restorePortlets() function:

```
if(historyPortlet.isMinimized())
historyPortlet.restore();
end
```

7. Save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for MainHandler.egl after Lesson 9" on page 71.

Test the portal

Test the main portal to make sure that the new History portlet is displayed and works correctly.

- 1. At the bottom of the editor, click **Preview**. EGL displays the main portal and the three subsidiary portlets.
- 2. Click **Calculate**. The animated GIF that indicates that processing is in progress is displayed. When the calculation finishes, the pie chart and history are displayed.

Calculato	or		0 🖸
Amount:	180000		
Rate:	5.2		
Term:	30 💌		
	Calcu	late	
Payment:	\$988.40	18	
Results			- 🖸
		Interest	
History			12.25
			0 🖸

- **3**. Change the **Term** of the mortgage to 5 years and click **Calculate** again. A second row is added to the history list.
- 4. Click a cell in the first row of the history list.

Calculato	or			1000
Amount:	180000			
Rate:	5.2			
Term:	5 💌			
	Calcul	ate		
Payment:	\$3413.34	4		
Results				- 83
	Principa	al	iterest	
History	Principa	al	aterest	0 2
History	Principa	lr al	Payment	- 0
History Principal 180000	Principa Rate 5.2	al Years 30	Payment 988.4	- 0

5. The pie chart displays the values for the selected row from the history list.

Calculato	or			0.0
Amount:	180000			
Rate:	5.2			
Term:	5 💌			
	Calcul	late		
Payment:	\$3413.3	4		
Results				- 🛙
	(Interest		
		Interest Principal		
History		Interest Principal		0 0
History	Rate	Interest Principal	Payment	- 0
History Principal 120000	Rate 5.2	Interest Principal	Payment 988:4	

Lesson checkpoint

You learned how to complete the following tasks:

- Subscribe to multiple events
- Create and update a portlet widget that contains a table

In the next lesson, you add a portlet that displays a map of mortgage companies in your area.

Lesson 10: Create the UI for the Map portlet

Create a portlet where you can enter a zip code and see a list of nearby mortgage companies and a map. Click the name of a company, and the map displays the location of the company.

This lesson relies on capabilities from two external Web sites:

- The Yahoo! Local Search Service provides information about businesses in a specific zip code
- The Google Maps API provides a map that you can imbed in a portlet to show an address

Each program requires a key, which you can obtain online. To get the keys, you must have accounts with both Google and Yahoo!. If you do not have accounts, you can easily create them for free. The Google key is only necessary if you deploy the application to a live server. You can use a dummy key for testing.

Get a Yahoo! application ID

The Local Search Service is part of the Yahoo! developer network. To use it, you must obtain a Yahoo! application ID:

1. Browse to the following location:

http://developer.yahoo.com/search/local/V3/localSearch.html

2. In the upper right corner, click Get an App ID.



- **3.** On the login page, enter your Yahoo! ID and password and click **Sign In**. If you do not have a Yahoo! ID, click **Sign up for Yahoo!** and complete the form. You can complete this process for free.
- 4. On the Yahoo! Developer Registration page, complete all the fields that are marked with asterisks:
 - a. Click Generic. This tutorial does not require user authentication.
 - b. For **Developer/Company Name**, enter the name associated with your Yahoo! ID.
 - c. For **Product Name**, enter the following name: Mortgage Tutorial
 - d. For **Contact email**, you can use the email address that is associated with your Yahoo! account (*yahoo_id*@yahoo.com).
 - e. For the Description of the application, enter the following description: Mortgage Tutorial

- 5. Click **Continue**. Yahoo! displays your 68-character application ID. Highlight this ID and press Ctrl+C to copy it into your copy buffer.
- **6**. Paste the ID into a text file in a location where you can easily find it, such as your Desktop. In the file, note that the ID is the Yahoo! key.

Get a Google Maps key

If you plan to display the map on a live Web site, the Google Maps API requires a key. If you are only interested in completing this tutorial, you can skip this exercise and use the API without a key.

To get a Google Maps key:

1. Browse to the following location:

http://www.google.com/apis/maps/signup.html

- 2. In the upper right corner of the page, click the **Sign in** link. If you have a Google account, enter your sign-in credentials. If you do not have an account, click **Create an account now** and complete the free registration process.
- **3**. At the bottom of the Sign Up for the Google Maps API page, complete the following fields:
 - a. Select I have read and agree with the terms and conditions.
 - b. If you plan to use the Google key on a live Web site, enter the URL for that site in the **My web site URL** field.
 - c. Click Generate API key. Google displays a page with your 84-character key and several code snippets. Note the snippet with the heading "JavaScript Maps API Example". This tutorial provides the JavaScript file that uses this snippet: WebContent/utils/GoogleMap.js. However, when you create your own application, you would use this code to create your own JavaScript.
- 4. Copy this key and paste it in the same file as your Yahoo! application ID. Make a note in the file that this is the Google key.

Create the Local Search Interface

When you use an external Web service, you create an Interface part to give EGL necessary information about the service. The Interface contains function prototypes, consisting of the function name and parameters for all the functions in the service, but not the function code. The Interface might also include any Record definitions that you use in calls to the service.

This is an unusual Interface because the getSearchResults() function prototype serves as an alias for an HTTP request. The **uriTemplate** property contains the URI for this request, with variables enclosed in braces.

More information about using the Local Search Interface is available on the Yahoo! site at the following location:

http://developer.yahoo.com/search/local/V3/localSearch.html

- 1. Create a new Interface part by right-clicking **MortgageEGLProject** and clicking **New** → **Interface**.
- 2. In the New EGL Interface Part window, complete the following fields:
 - a. For **EGL source file name**, enter the following name: IYahooLocalService
 - b. For **Package**, enter the following name: interfaces
 - c. Click Finish.
- **3**. Replace the contents of the file with the following code:

package interfaces;

```
interface YahooLocalService
function getSearchResults(appId string in, zipCode string in) returns(ResultSet)
{@GetRest{uriTemplate = "http://local.yahooapis.com/LocalSearchService/V3/
localSearch?appid={appId}&query=mortgage&zip={zipcode}&results=8",
responseFormat = XML};
end
```

After you have pasted the code, remove the newlines and tabs within the function prototype. This declaration must appear on a single line, with no spaces in the URI.

The function places the incoming parameters into a URI to call the service on the Yahoo! site. In addition to your Yahoo! ID and the zip code to look up, the URI also provides the keyword "mortgage" to use in the search and specifies a maximum of 8 results.

This is only one example of calling a service from EGL. Others have multiple function calls, and may handle the connection to the service through the EGL deployment descriptor file (for more information about the EGL deployment descriptor, see "Lesson 14: Deploy and test the mortgage application" on page 55).

4. Save the file.

Related reference

Interface part

Add records to the Interface file

The Interface requires three Record definitions. These Record definitions match the XML response that the Yahoo! service returns. You can see samples of this XML at http://developer.yahoo.com/search/local/V3/localSearch.html. The XML uses tags like "<Title>" that you convert into Record fields.

Because a Record is a part, place these definitions after the **end** statement for the Interface part.

1. Create the ResultSet Record by pasting the following code at the end of the IYahooLocalService.egl file:

```
record ResultSet{@XMLRootElement{name = "ResultSet", namespace = "urn:yahoo:lcl"}}
totalResultsAvailable STRING{@xmlelement { namespace = "urn:yahoo:lcl"}};
results Result[]{@XMLElement{name = "Result", namespace = "urn:yahoo:lcl"}};
end
```

The ResultSet Record includes an array of Result type Records. You create that Record type in the next step.

2. Create the Result Record by pasting the following code at the end of the file:

```
record Result
Title STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Address STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
City STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
State STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Latitude STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Longitude STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
rating Rating{@xmlElement{namespace = "urn:yahoo:lcl"}};
end
```

The Result Record includes a variable based on a Rating type Record. You create that Record type in the next step.

3. Create the Rating Record by pasting the following code at the end of the file:

```
record Rating
AverageRating STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
end
```

- 4. Save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for IYahooLocalService.egl after Lesson 10" on page 72.
- 5. Close the file.

Create the GoogleMap externalType

A type is like a blueprint. It contains information that you can use to create a variable that you can use in a program. It is an abstraction. A STRING, a Record, and a Service are all examples of types in EGL.

You can also use blueprints from outside EGL to create variables in an EGL program. One example is the GoogleMap widget, which is a map that you can embed in your Web page. You use the EGL **externalType** definition to give EGL the information it needs about such an outside blueprint.

The GoogleMap external type works with the GoogleMap.js file that is included in the MortgageSupport.zip file that you downloaded in Lesson 2.

- 1. Create a new EGL source file to hold the external type:
 - a. Select **MortgageEGLProject** in the Project Explorer view and click the **New EGL Source File** icon.



- b. Give the file the following name: GoogleMap
- c. Place the file in the following package: widgets
- Replace the boilerplate contents of the file with the following code: package widgets;

```
ExternalType GoogleMap extends Widget type JavaScriptObject {
  relativePath = "utils",
   javaScriptName = "GoogleMap"
  }
  function showAddress(address String in, description string in);
```

```
width STRING{@JavaScriptProperty{setMethod="setWidth", getMethod="getWidth"}};
height STRING{@JavaScriptProperty{setMethod="setHeight", getMethod="getHeight"};
```

end

In terms of how the file is used, this **externalType** definition is similar to the Interface part you created earlier. It includes a function prototype and two fields.

- **3**. Save and close the file.
 - Related reference

```
ExternalType part
```

Create the UI for the MapLocatorHandler widget

In the following instructions, as in any complex sequence of changes to a source file, remember to save the file periodically. To create the MapLocatorHandler widget:

- 1. Create a Rich UI Handler in the handlers package, as you did in "Lesson 4: Create the user interface for the calculator" on page 15.
 - a. Give the Handler the following name:
 - MapLocatorHandler
 - b. Place the Handler in the following package: handlers

The Handler opens in the Design view of the EGL editor.

- 2. Highlight the ui Box and make the following changes in the Properties view:
 - a. Change the columns property to 1.
 - b. Change the *id* property to form. This setting assigns styles from the style sheet for your application, WebContent/css/MortgageEGLProject.css.
- 3. Create a line of introductory text:
 - a. Drag a TextLabel widget from the EGL Widgets Palette to the ui Box and give it the following name: introLabel
 - b. In the Properties view, change the text property to the following phrase: Search for local mortgage businesses
- 4. Create a box for zip code entry:
 - a. Drag a Box widget into the next position and assign the following name: formBox
 - b. In the Properties view, change the **columns** property to 3.
- 5. Create a label for the zip code input field:
 - a. Drag a TextLabel widget into the new Box and assign the following name: zipLabel
 - b. In the Properties view, change the text property to the following text: Zip code:
- 6. Create a text field where the customer can enter a zip code:
 - Drag a TextField widget into the next position in the formBox widget and assign the following name: zipField
 - b. In the Properties view, add the following value in the text field, which becomes the default value for the field:
 10001

This value represents the zip code for midtown Manhattan.

- **c**. Click the **Events** tab and click the row for the onKeyDown event. Click the plus sign to add a function for the event.
- d. In the New Event Handler window, enter the following name for the new function:

checkForEnter

e. Click **OK**. EGL switches to Source view and displays the stub checkForEnter() function.

- f. Click **Design** to return to Design view. The checkForEnter function name is now displayed next to the onKeyDown event. The function is bound to the zipField field. You will add the code for this function later.
- 7. Add a button to initiate the search for the specified zip code:
 - a. Drag a Button widget from the Dojo Widgets Palette to the third position in the formBox Box. This button is actually a DojoButton and has the correct type in the source code. Assign the following name: zipButton
 - b. In the Properties view, make sure that you are on the Events page. Click the row for the onClick event. Click the plus sign to add a function for the event.
 - **c.** In the New Event Handler window, enter the following name for the new function:

buttonClicked

- d. Click **OK**. EGL switches to Source view and displays the stub buttonClicked() function.
- e. Click **Design** to return to Design view. The buttonClicked function name is now displayed next to the onClick event. The function is bound to the zipButton button. You will add the code for this function later.
- f. Click the Properties tab to switch back to the Properties page. Change the text property for the button to the following name: Search
- **8**. Drag a Box widget from the EGL Widgets palette into the next position *after* the formBox Box and give it the following name:

mapBox

o.cooa <mark>1</mark> 100	01 🔤 TextLabel (introLabel)	<u> </u>
	Box (formBox)	
	BOX (TORMBOX)	SS

- 9. Create a box to contain the list of mortgage companies.
 - a. Drag a second box onto the new mapBox Box and give it the following name:

listingBox

b. In the Properties view, set the **columns** property to 1. Expand the **Position** category and change the **width** property to 120.

You are finished working in EGL visual editor. In the next lesson, you will add source code to complete the portlet. Your editor pane looks like the following image:

	7
Search for local mongage businesses	
F	-
Zip code [10001] [Sea	rch
É.	1
F	1770
L_	

If you click the **Source** tab, you can see code that the EGL visual editor created. This code matches the code in "Finished code for MapLocatorHandler.egl after Lesson 10" on page 73.

Lesson checkpoint

You learned how to complete the following tasks:

- · Obtain keys from Yahoo! and Google to use their services
- · Create an Interface and Record definitions for the Local Search service
- · Create an externalType for the GoogleMaps widget
- Create a user interface for the Map portlet

In the next lesson, you add source code to complete the portlet.

Lesson 11: Create the source code for the Map portlet

Connect the service and the external widget to the user interface you created in the previous lesson.

Finish the source code for MapLocatorHandler.egl

- 1. Make sure that the MapLocatorHandler.egl file is open in the EGL editor. If you are in Design view, click the **Source** tab.
- 2. In the Handler declaration, change the **height** property of the ui Box to 450. This setting provides adequate space for the GoogleMap widget that you declare later.
- 3. Add a blank line below the Handler declaration, and then add the following line:

const YAHOO_APP_ID STRING = "app_id";

where *app_id* is the 68-character application ID you obtained from Yahoo! in the previous lesson. This ID must be surrounded by quotation marks.

 On the next line, declare a variable to represent the Yahoo! lookup service: lookupService YahooLocalService{@restbinding};

The **@restbinding** property indicates that EGL must look for binding information under the heading "YahooLocalService" in the deployment descriptor file. You will learn about the deployment descriptor file in "Lesson 14: Deploy and test the mortgage application" on page 55.

5. In the MapBox declaration, add a widget named localMap to the **children** property. The result looks like the following code:

```
MapBox Box{ padding=8,
    children = [ listingBox, localMap ] };
```

- After the listingBox declaration, declare the localMap widget: localMap GoogleMap{width = 500, height = 350};
- 7. Complete the start() function:

```
function start()
  search(); // show search results
end
```

The initial display shows results for the default zip code.

8. Complete the checkForEnter() function:

```
function checkForEnter(event Event in)
if(event.ch == 13)
search();
end
end
```

9. Complete the buttonClicked() function:

```
function buttonClicked(event Event in)
  search();
end
```

10. Add the search() function:

```
function search()
listingBox.setChildren([new Image{src = "icons/progress3.gif"}]);
localMap.showAddress(zipField.text, ""); // default map (no address)
// Call remote Yahoo Service and pass zip code
call lookupService.getSearchResults(YAHOO_APP_ID, zipField.text)
returning to showResults onException displayError;
end
```

The setChildren() function provides a shorthand way to display a progress GIF, as you did in the Calculator portlet. When you use the setChildren() function again later to display the search results in the same Box widget, those results replace the progress image.

The second field of the localMap.showAddress() function is used for an address. For the initial display, you do not provide an address.

11. Add the displayError() function that is specified in the search() function:

```
function displayError(ex AnyException in)
DialogLibrary.showError("Yahoo Service", "Cannot invoke Yahoo Local Service: "
    + ex.message, null);
end
```

The DialogLibrary is part of the com.ibm.egl.rui.dialog plug-in that you added to your workspace in Lesson 2, along with the portlet plug-in. It provides basic message functions for your Web page.

12. Add the showResults() function that is specified in the search() function:

```
function showResults(retResult ResultSet in)
linkListing HyperLink[0];
for(i INT from 1 to retResult.results.getSize() by 1)
newLink HyperLink{text = retResult.results[i].title, href = "#"};
newLink.setAttribute("address", retResult.results[i].Address + ", '
+ retResult.results[i].city + ", "
+ retResult.results [i].state);
newLink.setAttribute("title", retResult.results[i].Title);
newLink.onClick ::= mapAddress;
linkListing.appendElement(newLink);
end
listingBox.setChildren(linkListing);
end
```

Your call to the service returns an array of mortgage company addresses. The showResults() function reads through that array and creates a new array of business names in the form of hyperlinks, with a maximum of 8, to prevent overflowing the screen. The links point to the current page; all you need is the ability to register an onClick event for each name. That onClick event triggers the mapAddress() function, which you create in the next step.

13. Add the mapAddress() function specified in the showResults() function:

```
function mapAddress(e Event in)
    // Show the address on the map when the link is clicked
    businessAddress STRING = e.widget.getAttribute("address") as STRING;
    businessName STRING = e.widget.getAttribute("title") as STRING;
    localMap.showAddress(businessAddress, "<b>" + businessName + "</b>");
end
```

14. Organize your imports by pressing Ctrl+Shift+O, and save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for MapLocatorHandler.egl after Lesson 11" on page 73.

Test the new portlet

Because this portlet works independently, you can test it separately.

1. Make sure to save the file, and then click **Preview**. EGL displays the entry form with the zip code 10001 selected. A list of mortgage companies is displayed down the left side of the screen. On the right is a map of New York City.



Search for local mortgage businesses

If the page does not display properly, make sure that the GoogleMap.js file is in the WebContent/utils directory for your project.

- 2. Click any of the names in the left column. The map displays an indicator that shows the location of the business.
- 3. Enter a different zip code and click Search. The map displays a new location.
- 4. Close the file.

Lesson checkpoint

You learned how to complete the following tasks:

- Create and use a variable that is based on the Local Search service
- Create and use a variable that is based on the GoogleMaps widget

In the next lesson, you add the Map portlet to the main portal.

Lesson 12: Add the Map portlet to the main portal

To add the new portlet to your page, you must change the main portal.

Change the main portal

Add lines for the Map portlet that are similar to the lines for the other three portlets:

- 1. In the EGL editor, open the MainHandler.egl file and click the **Source** tab.
- Immediately below the historyHandler declaration, add a similar declaration for mapHandler:

mapHandler MapLocatorHandler{};

3. Immediately below the historyPortlet declaration, add a similar declaration for mapPortlet:

```
mapPortlet Portlet{children = [mapHandler.ui],
    title = "Map", canMove = FALSE, canMinimize = TRUE};
```

4. In the start() function, below the existing calls to addPortlet(), add the new portlet to the portal:

mortgagePortal.addPortlet(mapPortlet, 2);

In this case, you are adding the Map portlet to the second, wider column.

5. Save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for MainHandler.egl after Lesson 12" on page 75.

Test the portal

Test the main portal to make sure that the new Map portlet is displayed and that all portlets work correctly.

- 1. At the bottom of the editor, click **Preview**. EGL displays the main portal and the four subsidiary portlets.
- 2. Calculate at least two different mortgages and enter a five-digit U.S. zip code in the Map portlet.



3. Close the file.

Lesson checkpoint

There were no new tasks in this lesson.

In the next lesson, you add a portlet to display a map of mortgage companies for a specific area.

Lesson 13: Install Apache Tomcat

Use the Apache Tomcat open source Web server to display the generated version of your Web page.

Download and install the server

You can download and install the Tomcat Web server in the EGL workbench. If you have the Tomcat server on your system, you still must create a copy of the server for EGL to use.

To install and configure the server:

1. Locate the Servers view, which is by default at the lower right of the workbench. Right-click the empty space and click **New** → **Server**.

🔝 Problems 📮 Console 🙌 Server	rs 🕱 🔪 🕼 EGL Generat
Server 🔺	State
New 🕨 🎬 Server	
10.08.6	

- 2. In the Define a New Server window, expand **Apache** and click **Tomcat v6.0 Server**. Accept the default values for the other fields. Click **Next**.
- **3**. In the Tomcat Server window, click **Download and Install**. If the Tomcat server is already installed, click **Browse** and find the installation directory for the server, and then go to step 4 on page 54

🗐 New Server		
Tomcat Server Specify the installation directory		
Name:		
Apache Tomcat v6.0		
Tomcat installation directory:		
		Browse
	apache-ton	ncat-6.0.14 Download and Install
JRE:		<u></u>
Workbench default JRE		▼ Installed JREs
Ø	<back next=""></back>	Finish Cancel

Accept the terms of the license agreement. Browse to a directory for the application files, such as C:\Program Files\Apache. While it completes the installation, EGL displays the Define a New Server window again, this time with the installation directory specified. Progress is shown at the bottom right of the workbench.

- 4. When the installation is completed, click **Finish**.
- 5. Start the server by clicking the green circle with the white triangle at the top of the Server view.

0

Lesson checkpoint

In this lesson, you completed the following tasks:

- Downloaded the Apache Tomcat Web server, if necessary
- Installed Tomcat in your workspace

In the next lesson, you deploy the application to the Tomcat server and run it there.

Lesson 14: Deploy and test the mortgage application

During the deployment process, EGL creates HTML files and server-specific code to match your target environment.

Deployment is a two stage process:

- 1. Deploy your Handlers to a Web project
- 2. Deploy the Web project to a Web server, which in this case is the Apache Tomcat server you installed in the previous lesson

After you deploy the application, you can run it on the server rather than in Preview view.

Edit the deployment descriptor

EGL automatically creates a deployment descriptor file for each EGLSource folder. The deployment descriptor file manages the deployment process.

To edit the deployment descriptor:

- 1. In the EGLSource folder, double-click the MortgageEGLProject.egldd file. The EGL deployment descriptor opens in the editor. Because you created the various Handler parts with the wizard, EGL automatically added them to the list of Rich UI Handlers to deploy.
- 2. Because you are using a dedicated service, you do not need to add any information to the **Service Client Bindings** section. The list is empty.

				ien C
EGL Deployment Descriptor - Mortgage	GLProject			
Save this deployment descriptor after any ed	its, and then run Deploy	EGL Descriptor to deploy	to the target Web project.	
Select a Web project as the deployment targe	et.			
TerretNese	Mour	1. A.		
Target Name:	INEW.	· .		
 Services Deployment 		• Rich UI Deployment		
 Services Deployment Generate the following service parts as Web s 	ervices.	Rich UI Deployment Configure the deployment	: settings for Rich UI applicati	ions,
 Services Deployment Generate the following service parts as Web s 	ervices.	Rich UI Deployment Configure the deployment Rich UI Handler	settings for Rich UI applicati	ions.
 Services Deployment Generate the following service parts as Web s 	ervices.	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation	settings for Rich UI applicati	ions. Detail
 Services Deployment Generate the following service parts as Web s 	ervices. Detail	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.Calculation	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH	ions.
 Services Deployment Generate the following service parts as Web s 	ervices. Detail	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.MainHandler	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH MainHandler	ions.
 Services Deployment Generate the following service parts as Web s 	ervices.	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.MainHandler handlers.MapLocator	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH MainHandler MapLocatorHandler	ions.
 Services Deployment Generate the following service parts as Web s 	ervices.	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.MainHandler handlers.MapLocator handlers.MortgageC	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH MainHandler MapLocatorHandler MortgageCalculator	ions.
 Services Deployment Generate the following service parts as Web s 	Detail	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.Calculation handlers.MainHandler handlers.MapLocator handlers.MortgageC	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH MainHandler MapLocatorHandler MortgageCalculator	ions.
✓ Services Deployment Generate the following service parts as Web s	Detail	Rich UI Deployment Configure the deployment Rich UI Handler handlers.Calculation handlers.Calculation handlers.MainHandler handlers.MapLocator handlers.MortgageC	settings for Rich UI applicati HTML File Name CalculationHistoryHa CalculationResultsH MainHandler MapLocatorHandler MortgageCalculator	ions.

3. Next to the **Target Name** field, click **New**. The Dynamic Web Project wizard opens.

4. For the **Project Name**, enter the following name: MortgageWeb

Any Web project is acceptable. You are creating a simple one for the purposes of the tutorial.

- 5. Click **Finish**. EGL redisplays the deployment descriptor and creates the Web project.
- 6. Save and close the deployment descriptor.

Deploy the Rich UI application

After you edit the deployment descriptor, the deployment process is automatic:

- 1. In the EGLSource folder, right-click the MortgageEGLProject.egldd file.
- 2. Click **Deploy EGL Descriptor**. The deployment process is automatic. At the end of the process, the Tomcat server might show a "Restart" status.

		and the second	CONTRACTOR CONTRACTOR
Server * S	tate	Status	
😐 🚆 Tomcat v6.0 Server at localhost 🖁	Started	Restart	

3. Restart the server by clicking the white arrow in the green circle in the upper

right of the **Servers** window **O** . Alternatively, you can right-click the server name and click **Restart**. When the server has restarted, the status is "Synchronized."

Run the generated code

- In the MortgageWeb/WebContent folder, find the generated HTML version of the MainHandler page. The name of the file is MainHandler-en_US.html.
- 2. Right-click the file name and click **Run As** → **Run on Server**



The Run On Server window opens.

3. In the Run On Server window, select the Tomcat 6.0 Server and click **Always use this server when running this project**. Click **Finish**.

😰 Run On Server	
Run On Server	
Select which server to use	
How do you want to select the server?	
Choose an existing server	
C Manually define a new server	
Select the server that you want to use:	
type filter text	
E 🗁 localhost	
🖥 Tomcat v6.0 Server at localhost 🛛 🖁 Started	
	11
Apache Tomcat v6.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 Web modules.	
Always use this server when running this project	
(2) CRack Next Next Next Next Next Next Next Next	Cancal
	Caricel

4. If you see a page not found error (404), see if the server is showing a Restart status. If so, restart the server and refresh the page. The page opens.

A Problems Servers	Co Vill Cor Generation	r Results Mig Ede Depidy Results	NY.	V P L P L
Server 🔺	State	Status		
🖽 🚆 Tomcat v6.0 Server at localhos	: 🖡 Started	Restart		

5. Test the application by calculating mortgages that are based on different rates, amounts, and terms. Verify that clicking a row in the History portlet displays the appropriate information in the Results portlet. Change the zip code in the Map Portlet and make sure the links cause the map to update.

Lesson checkpoint

You learned how to complete the following tasks:

- · Edit a deployment descriptor to deploy a Rich UI handler
- Run the application on the Web server

In the next lesson, which is optional, you use the EGL MVC framework to improve the user interface for the Calculator portlet.

Lesson 15: (Optional) Use validating forms in the Calculation portlet

Replace the simplified code in MortgageCalculatorHandler.egl with code that uses the EGL Model-View-Controller (MVC) framework.

You can use EGL Rich UI to define a form for gathering input from a user. Each field in the form can contain a label, a widget for displaying the value, and if necessary, a set of validation routines to verify the user input, You can use the ValidatingForm type to provide basic error checking and formatting tasks. Use a form that is based on this type to replace the **Amount**, **Rate**, and **Term** fields and labels in your original version of the Calculation portlet. Use a second form to replace the **Payment** field.

Edit the Calculation portlet

- 1. Open MortgageCalculatorHandler.egl in Source view.
- 2. In the ui Box declaration, change the value of the **columns** property to 1.
- 3. In the ui Box declaration, replace the following widgets with a new structure named inputForm:
 - amountLabel
 - amountField
 - rateLabel
 - rateField
 - termLabel
 - termCombo

At this point, the ui Box declaration looks like the following line:

```
ui Box{ columns = 1, children = [ inputForm, blankLabel, buttonBox, paymentLabel,
    paymentField, errorDisplay ], id = "form" };
```

Be sure to keep the **id** property set to "form" so that the UI does not look crowded.

- 4. On the same line, remove the blankLabel variable. You will add it to the buttonBox widget later.
- 5. On the same line, replace the paymentLabel and paymentField variables with a new variable named paymentForm. The finished ui Box declaration looks like the following line:

ui Box{ columns = 1, children = [inputForm, buttonBox, paymentForm, errorDisplay], id = "fo

6. Instead of setting default values individually, create a variable that is based on the MortgageCalculationResult record to hold the default values, and add it below the :

```
inputRec MortgageCalculationResult{
  loanAmount = 180000,
  interestRate = 5.2,
```

```
term = 30,
monthlyPayment = 0,
interest = 0
};
```

7. Create the inputForm variable and define the fields within it:

```
inputForm ValidatingForm{width=530, entries = [
    new FormField{displayName="Amount:", controller=amountController},
    new FormField{displayName="Rate:", controller=rateController},
    new FormField{displayName="Term:", controller=termController}
]};
```

The validating form is part of the MVC framework. You can specify functions for a validating form to screen the user input.

- 8. Create the controllers that you referred to in the inputForm declaration.
 - a. Create the amountController variable:

```
amountController Controller{@MVC{model = inputRec.loanAmount, view = amountField}};
amountField TextField{};
```

You create two associations here, as you did in the errorController widget "Lesson 5: Add code for the MortgageCalculatorHandler functions" on page 26. The storage area (the "model") for the amount is the loanAmount field in the inputRec record. The display area (the "view") is a TextField widget, just like you used in the original version of this portlet.

b. Create the rateController variable:

```
rateController Controller{
  formatters ::= formatRate, unformatters ::= unformatRate,
  @MVC{model = inputRec.interestRate, view = rateField}
};
rateField TextField{};
```

This code is essentially the same as the code for the amountController variable, except that you also specify two functions:

- formatRate(), which adds a percent sign before you display the rate
- unformatRate(), which strips the percent sign before you store the rate in the record

You will add these functions later in this lesson.

c. Create the termController variable:

```
termController Controller{
  retrieveModelHelper = getTermModel,
  retrieveViewHelper = getTermCombo,
  publishHelper = setTermCombo,
  commitHelper = setTermModel,
  @MVC{model = inputRec.term, view = termCombo}};

const TERM_VALUES String[] = ["5","10","15","30"];
termCombo Combo{ values = TERM_VALUES, selection = 4 };
```

Again, you create an association with a model and a view, and you specify function calls for formatting. You will create the get and set functions later in this lesson.

- 9. Delete the declarations for the following variables:
 - amountLabel
 - amountField
 - rateLabel
 - rateField

- termLabel
- termCombo

Your code now looks like the following image:

```
MortgageCalculatorHandler.egl
  Chandler MortgageCalculatorHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start, css
       mortService MortgageCalculationService{};
0
       ui Box{ columns = 1, children = [ inputForm, buttonBox, paymentForm, errorDisplay ], id = "form" };
       inputRec MortgageCalculationResult{
           loanAmount = 180000,
           interestRate = 5.2,
           term = 30,
           monthlvPavment = 0,
           interest = 0
       3;
       inputForm ValidatingForm{width=530, entries = [
new FormField{displayName="Amount:", controller=amountController},
           new FormField(displayName="Rate:", controller=rateController},
           new FormField{displayName="Term:", controller=termController}
       1};
       amountController Controller {@MVC{model = inputRec.loanAmount, view = amountField}};
       amountField TextField{};
       rateController Controller{
0
           formatters ::= formatRate, unformatters ::= unformatRate,
           @MVC{model = inputRec.interestRate, view = rateField}
       3 :
       rateField TextField{};
       termController Controller{
retrieveModelHelper = getTermModel,
           retrieveViewHelper = getTermCombo,
           publishHelper = setTermCombo,
           commitHelper = setTermModel,
           @MVC{model = inputRec.term, view = termCombo}};
       const TERM VALUES String[] = ["5", "10", "15", "30"];
       termCombo Combo{ values = TERM_VALUES, selection = 4 };
```

```
blankLabel TextLabel{};
```

 Add the blankLabel widget to the children property of the buttonBox widget: buttonBox Box{ padding=8,

children = [blankLabel, calculationButton, processImage] };

11. Remove the existing paymentLabel and paymentField declarations and replace them with the following code:

```
paymentField TextLabel{};
paymentField TextLabel{};
paymentController Controller{@MVC{model = inputRec.monthlyPayment, view = paymentField}};
paymentForm ValidatingForm{width=530, entries = [
    new FormField{displayName = "Payment:", controller = paymentController}
]
};
```

In the previous form, you used TextField widgets for the view. Here, you use a TextLabel widget because the field is used for output only; the user is not permitted to change this information.

12. Change the start() function to initially hide the payment form:

```
function start()
    hidePaymentForm();
end
```

13. After the functions that show and hide the process image, add functions to show and hide the payment form:

```
function showPaymentForm()
paymentForm.visibility = "visible";
end
function hidePaymentForm()
paymentForm.visibility = "hidden";
end
```

- 14. In the calculateMortgage() function, delete all the lines except the last one, the call statement. You do not need to create the inputRec record because it is now a global variable.
- 15. In the calculateMortgage() function, the call statement now becomes part of the following code for the function:

```
function calculateMortgage()
if (inputForm.isValid())
inputForm.commit();
inputForm.publish();
call mortService.amortize(inputRec) returning to displayResults
        onException handleException;
else
    hideProcessImage();
end
end
```

16. In the displayResults() function, replace the first line, which begins with "paymentField.text", with the following lines:

```
inputRec.monthlyPayment = retResult.monthlyPayment;
paymentForm.publish();
```

The paymentField widget is tied to the inputRec.monthlyPayment field through the paymentController MVC controller. After you change the inputRec field and publish the form, the field is automatically updated.

17. Also in the displayResults() function, add the following function call after the hideProcessImage() call:

showPaymentForm();

This function makes the form, and therefore the payment field, visible.

- 18. Before the final end statement in the handler, add the formatting functions.
 - a. The formatRate() function adds a percent sign to the rate amount. The formatRate() function removes the percent sign.

```
function formatRate(input string in) returns(string)
return(input+ "%");
end
function unformatRate(input string in) returns(string)
len int = strlib.characterLen(input);
if (input[len:len] == "%")
return(input[1:len-1]);
end
return(input);
end
```

b. The setTermCombo() function reads a number in STRING form and matches it to a value from the termCombo widget:

```
function setTermCombo(value STRING in)
    // Set the term combo box to the index corresponding by the value of the selected term
    for (i int from 1 to TERM_VALUES.getSize() by 1)
    if(TERM_VALUES[i] == value)
        termCombo.selection = i;
```

```
exit for;
      end
     end
    end
c. The getTermCombo() function returns the currently selected term in
   STRING form:
   function getTermCombo() returns(STRING)
     // Return the selected index of the term combo box
     return (termCombo.selection);
    end
d. The setTermModel() function validates and stores the index of a term
   value:
    function setTermModel(value string in)
     // value represents the index in the combo
     if(value >= 1 && value <= TERM_VALUES.getSize())</pre>
      inputRec.term = TERM_VALUES[value as int];
     end
    end
e. The getTermModel() function returns the current value of the term in
   STRING form:
```

```
function getTermModel() returns(STRING)
    // returns the term value
    return(TERM_VALUES[termCombo.selection]);
end
```

19. Organize your imports and save the file. If you see errors in your source file, compare your code to the file contents in "Finished code for MortgageCalculatorHandler.egl after Lesson 15" on page 76.

Test the portlet

1. Click the **Source** tab at the bottom of the editor pane. The interface is displayed with the default values.

Amount:	\$180,000.00]
Rate:	5.2%	j
Term:	30 💌	
Calc	culate	

2. Click Calculate. The payment form is displayed.

Amount:	\$180,000.00	
Rate:	5.2%	
Term:	30 💌	
Calc	culate	

Payment: \$988.40

- **3**. Change any of the input fields and recalculate. You can see a number of differences from the previous version of the portlet:
 - The amount is formatted with the appropriate currency symbol, separator, and decimal. This happens automatically because MVC associates the field with the loanAmount field of the inputRec record, and you declared loanAmount as a MONEY type.
 - The rate is formatted with a percent sign. This happens in the formatRate() function.
 - You can use a currency symbol, percent sign, separator, or decimal in your input. These characters would have caused an error in the previous version.
 - The payment form is not visible until you perform a calculation.
 - The payment field is formatted as currency because it is associated with a MONEY type field in the inputRec record.
 - You cannot overwrite the payment amount.
- 4. Close the file.

Test the portal in Preview view

- 1. In the EGLSource/handlers directory, open the MainHandler.egl file and click the **Preview** tab. The new Calculate portlet is displayed.
- 2. Click Calculate.
- **3**. Change any of the values and click **Calculate** again. The Results and History portlets function just as they did before.
- 4. Close the file.

Redeploy and test

Because you changed one of the deployed files, you must redeploy the project.

- In the EGLSource directory, right-click the EGL deployment descriptor file, MortgageEGLProject.egldd, and then click Deploy EGL Descriptor. The deployment process is automatic.
- In the MortgageWeb/WebContent directory, right-click the MainHandleren_US.html file and click Run As → Run on Server. As before, you might need to restart the server and refresh the page.
- **3**. Verify that the new version of the portlet is displayed, and that everything functions correctly.

Lesson checkpoint

You learned how to complete the following tasks:

- Use the EGL MVC framework to improve the Calculator interface
- · Redeploy the application to reflect program changes

You have completed the tutorial.

Summary

You have completed the Create a mortgage portal with EGL Rich UI tutorial.

In addition to explaining the basic steps required to create a portal page, this tutorial provided opportunities to practice the following skills:

- Designing your work on paper before coding
- Creating and deploying a service
- Creating a Rich UI Web page by writing source code in the EGL editor
- Using a service and a widget from an external provider

Lessons learned

You completed the following tasks:

- Plan the application and design the interface
- Import a custom widget to manage portlets
- Write a service to calculate mortgage payments
- Create a portlet to request input for the calculation service and display the results
- · Create a pie chart to compare total principal to total interest
- Pass data between portlets by using the InfoBus widget
- Create a table that lists all calculations
- Create a portlet to find mortgage businesses
- Create a Rich UI portal page to contain the individual portlets
- · Install and configure the Apache Tomcat Web server
- Deploy the Web page to the server and test the application
- Replace the Calculation portlet with a version that uses validating forms

Additional resources

EGL Rich UI follows the Visual Formatting Model of the World Wide Web Consortium (W3C). For more information, see the W3C site at http://www.w3.org/TR/CSS2/visuren.html.

Finished code for MortgageCalculationService.egl after Lesson 3

The following code is the complete text of the MortgageCalculationService.egl file at the end of Lesson 3. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version. package services;

service MortgageCalculationService

```
function amortize(inputData MortgageCalculationResult inOut)
 amt MONEY = inputData.loanAmount;
 // convert to monthly rate
 rate DECIMAL(10, 8) = (1 + inputData.interestRate / 1200);
 // convert to months
 term INT = (inputData.term * 12);
  // calculate monthly payment amount
 pmt MONEY = (amt * (rate - 1) * Mathlib.pow(rate, term)) /
    (MathLib.pow(rate, term) - 1);
  totalInterest MONEY = (pmt * term) - amt;
  // update result record
 inputData.monthlyPayment = pmt;
 inputData.interest = totalInterest;
end
end
record MortgageCalculationResult
 // user input
loanAmount MONEY;
interestRate DECIMAL(10,8);
term INT;
 // calculated fields
monthlyPayment MONEY;
interest MONEY;
end
   Related tasks
```

"Lesson 3: Create the mortgage calculation Service" on page 11 Create a dedicated service to calculate monthly payments.

Finished code for MortgageCalculatorHandler.egl after Lesson 4

The following code is the complete text of the MortgageCalculatorHandler.egl file after Lesson 4. The code includes tabs, comments, and blank lines. This code was created by the EGL visual editor. If you see errors in your source file, compare your code with this version.

package handlers;

// RUI Handler

```
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.TextLabel;
import com.ibm.egl.rui.widgets.TextField;
import com.ibm.egl.rui.widgets.Combo;
import dojo.widgets.DojoButton;
import com.ibm.egl.rui.widgets.Image;
import com.ibm.egl.rui.widgets.HTML;
11
//
handler MortgageCalculatorHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start
ui Box{ columns = 2, width = "400", height = "400", children = [ amountLabel, amountField, rateLabe
 id = "form" };
 amountLabel TextLabel{ text = "Amount:" };
 amountField TextField{
 text = "180000"
}:
 rateLabel TextLabel{ text = "Rate:" };
 rateField TextField{
 text = "5.2"
};
```

```
termLabel TextLabel{ text = "Term:" };
termCombo Combo{ values = ["5","10","15","30"],
selection = 4 };
blankLabel TextLabel{};
buttonBox Box{ padding=8,
children = [ calculationButton, processImage ] };
calculationButton DojoButton{ text = "Calculate", onClick ::= calculate };
processImage Image{
 src = "icons/progress3.gif",
 visibility = "hidden"
}:
paymentLabel TextLabel{ text = "Payment:" };
paymentField TextField{};
errorDisplay HTML{
 color = "Red"
};
function start()
end
function calculate(event Event in)
end
end
   Related tasks
```

"Lesson 4: Create the user interface for the calculator" on page 15 Each portlet on the finished page is controlled by an EGL Rich UI Handler part.

Finished code for MortgageCalculatorHandler.egl after Lesson 5

The following code is the complete text of the MortgageCalculatorHandler.egl file after Lesson 5. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version. package handlers;

```
import com.ibm.egl.rui.mvc.Controller;
import com.ibm.egl.rui.mvc.MVC;
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.Combo;
import com.ibm.egl.rui.widgets.HTML;
import com.ibm.egl.rui.widgets.Image;
import com.ibm.egl.rui.widgets.TextField;
import com.ibm.egl.rui.widgets.TextLabel;
import egl.ui.rui.Event;
import dojo.widgets.DojoButton;
import services.MortgageCalculationResult;
import services.MortgageCalculationService;
```

handler MortgageCalculatorHandler type RUIhandler {initialUI = [ui],onConstructionFunction = sta

```
mortService MortgageCalculationService{};
ui Box{ columns = 2, children = [ amountLabel, amountField, rateLabel, rateField, termLabel, term
amountLabel TextLabel{ text = "Amount:" };
amountField TextField{
    text = "180000"
};
rateLabel TextLabel{ text = "Rate:" };
rateField TextField{
    text = "5.2"
};
termLabel TextLabel{ text = "Term:" };
termCombo Combo{ values = ["5","10","15","30"],
selection = 4 };
blankLabel TextLabel{};
```

```
buttonBox Box{ padding=8,
children = [ calculationButton, processImage ] };
 calculationButton DojoButton{ text = "Calculate", onClick ::= calculate };
processImage Image{
 src = "icons/progress3.gif",
 visibility = "hidden"
};
paymentLabel TextLabel{ text = "Payment:",
class = "spacer" };
paymentField TextField{};
 // use for error messages
error STRING = "":
errorDisplay HTML{ color = "Red" };
// associate MVC with errorDisplay widget
errorController Controller{@MVC{model = error, view = errorDisplay}};
function start()
end
 function calculate(event Event in)
 showProcessImage();
 calculateMortgage();
end
 function showProcessImage()
 processImage.visibility = "visible";
end
 function hideProcessImage()
 processImage.visibility = "hidden";
 end
 function calculateMortgage()
 // new copy of the input record
 inputRec MortgageCalculationResult{};
 // load with values from the ui
 inputRec.loanAmount = amountField.text as MONEY;
 inputRec.interestRate = rateField.text as DECIMAL(10,8);
 inputRec.term = termCombo.values[termCombo.selection] as INT;
 call mortService.amortize(inputRec) returning to displayResults
    onException handleException;
end
 function displayResults(retResult MortgageCalculationResult in)
 paymentField.text = "$" + retResult.monthlyPayment as STRING;
 hideProcessImage();
end
 private function setError(err STRING in)
 error = err:
 errorController.publish();
end
// catch-all exception handler
private function handleException(ae AnyException in)
 setError("Error calling service: " :: ae.message);
end
end
```

Related tasks

"Lesson 5: Add code for the MortgageCalculatorHandler functions" on page 26 Add functions in the MortgageCalculatorHandler part to support the user interface that you constructed in the previous lesson.
Finished code for CalculationResultsHandler.egl after Lesson 6

The following code is the complete text of the CalculationResultsHandler.egl file after Lesson 6. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version. package handlers;

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.widgets.Box;
import egl.ui.color;
import dojo.widgets.DojoPieChart;
import dojo.widgets.PieChartData;
import services.MortgageCalculationResult;
handler CalculationResultsHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = sta
ui Box{ columns = 2, width = "400", height = "300", children = [ interestPieChart ] };
interestPieChart DojoPieChart{
  radius = 100,
 width = "300".
 height = "300",
 labelOffSet = 50,
 fontColor = "white",
 data = [
  new PieChartData{y=1, text="Principal", color="#99ccbb"},
  new PieChartData{y=0, text="Interest", color="#888855"}
]};
function start()
 InfoBus.subscribe("mortgageApplication.mortgageCalculated", displayChart);
end
function displayChart(eventName STRING in, dataObject ANY in)
 localPieData PieChartData[];
 localPieData = interestPieChart.data;
 resultRecord MortgageCalculationResult = dataObject as MortgageCalculationResult;
 localPieData[1].y = resultRecord.loanAmount;
 localPieData[2].y = resultRecord.interest;
 interestPieChart.data = localPieData;
end
end
```

Related tasks

"Lesson 6: Create the CalculationResultsHandler widget" on page 30 Add a second portlet on the page to hold a pie chart that shows the relative proportions of principal and interest.

Finished code for MainHandler.egl after Lesson 7

The following code is the complete text of the MainHandler.egl file after Lesson 7. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version.

package handlers;

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.portal.Portal;
import com.ibm.egl.rui.widgets.portal.Portlet;
import egl.ui.columns;
handler MainHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start, cssFile="c
ui Box{ columns = 2, children = [ mortgagePortal ] };
```

Create a mortgage portal with EGL Rich UI 69

```
mortgagePortal Portal { columns = 2, columnWidths = [ 300, 650 ] };
 calculatorHandler MortgageCalculatorHandler{};
resultsHandler CalculationResultsHandler{};
 calculatorPortlet Portlet{children = [calculatorHandler.ui], title = "Calculator"};
 resultsPortlet Portlet{children = [resultsHandler.ui],
  title = "Results", canMove = TRUE, canMinimize = TRUE};
 function start()
 mortgagePortal.addPortlet(calculatorPortlet, 1);
 mortgagePortal.addPortlet(resultsPortlet, 1);
 // Subscribe to calculation events
 InfoBus.subscribe("mortgageApplication.mortgageCalculated", restorePortlets);
 // Initial state is minimized
 resultsPortlet.minimize():
 end
 function restorePortlets(eventName STRING in, dataObject ANY in)
 if(resultsPortlet.isMinimized())
  resultsPortlet.restore();
 end
end
end
   Related tasks
```

"Lesson 7: Create the main portal" on page 33 The main page uses the EGL portlet widgets to manage communication between the parts of the application.

Finished code for CalculationHistoryHandler.egl after Lesson 8

The following code is the complete text of the CalculationHistoryHandler.egl file after Lesson 8. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version.

package handlers;

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.widgets.Box;
import egl.ui.columns;
import egl.ui.displayname;
import egl.ui.rui.Event;
import egl.ui.rui.Widget;
import dojo.widgets.DojoGrid;
import dojo.widgets.DojoGridColumn;
import services.MortgageCalculationResult;
handler CalculationHistoryHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start
ui Box{ columns = 2, children = [ historyGrid ] };
historyGrid DojoGrid { behaviors = [ addSelectionListener ],
  headerBehaviors = [ ],
  columns = [
   new DojoGridColumn { displayName = "Principal", name = "loanAmount", width = 60 },
  new DojoGridColumn { displayName = "Rate", name = "interestRate", width = 60 },
new DojoGridColumn { displayName = "Years", name = "term", width = 60 },
   new DojoGridColumn { displayName = "Payment", name = "monthlyPayment", width = 60 } ] };
 // array to store calculation results
historyResults MortgageCalculationResult[0];
 function start()
  // Subscribe to calculation events so history table (grid) can be updated
```

InfoBus.subscribe("mortgageApplication.mortgageCalculated", addResultRecord); end // Update grid to show latest mortgage calculation function addResultRecord(eventName STRING in, dataObject ANY in) resultRecord MortgageCalculationResult = dataObject as MortgageCalculationResult; historyResults.appendElement(resultRecord); historyGrid.data = historyResults as ANY[]; end // Adds a listener to each cell function addSelectionListener(grid DojoGrid in, cell Widget in, row ANY in, rowNumber INT in, column DojoGridColumn in) cell.setAttribute("row", rowNumber); cell.onClick ::= cellClicked; end // Publish event to InfoBus when previous calculation is selected function cellClicked(e Event in) trv row int = e.widget.getAttribute("row") as INT; InfoBus.publish("mortgageApplication.mortgageResultSelected", historyResults[row]); onException(ex AnyException) end end end **Related tasks**

"Lesson 8: Add a calculation history portlet" on page 35 Create a table where you can click a row to display a previous calculation.

Finished code for MainHandler.egl after Lesson 9

The following code is the complete text of the MainHandler.egl file at the end of Lesson 9. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version.

package handlers;

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.portal.Portal;
import com.ibm.egl.rui.widgets.portal.Portlet;
import egl.ui.columns;
handler MainHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start, cssFile="c
ui Box{ columns = 2, children = [ mortgagePortal ] };
mortgagePortal Portal { columns = 2, columnWidths = [ 300, 650 ] };
calculatorHandler MortgageCalculatorHandler{};
resultsHandler CalculationResultsHandler{};
historyHandler CalculationHistoryHandler{};
calculatorPortlet Portlet{children = [calculatorHandler.ui], title = "Calculator"};
resultsPortlet Portlet{children = [resultsHandler.ui],
   title = "Results", canMove = TRUE, canMinimize = TRUE};
historyPortlet Portlet{children = [historyHandler.ui],
   title = "History", canMove = TRUE, canMinimize = TRUE};
 function start()
 mortgagePortal.addPortlet(calculatorPortlet, 1);
 mortgagePortal.addPortlet(resultsPortlet, 1);
 mortgagePortal.addPortlet(historyPortlet, 1);
```

```
// Subscribe to calculation events
InfoBus.subscribe("mortgageApplication.mortgageCalculated", restorePortlets);
// Initial state is minimized
resultsPortlet.minimize();
historyPortlet.minimize();
end
function restorePortlets(eventName STRING in, dataObject any in)
if(resultsPortlet.isMinimized())
resultsPortlet.restore();
end
if(historyPortlet.isMinimized())
historyPortlet.restore();
end
end
end
end
```

Related tasks

"Lesson 9: Add the calculation history portlet to the main portal" on page 37 To add the new portlet to your page, you must change the Results portlet and the main portal.

Finished code for IYahooLocalService.egl after Lesson 10

The following code is the complete text of the IYahooLocalService.egl file at the end of Lesson 10. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version. package interfaces;

```
interface YahooLocalService
function getSearchResults(appId string in, zipCode string in) returns(ResultSet){@GetRest{uriTempla
end
record ResultSet{@XMLRootElement{name = "ResultSet", namespace = "urn:yahoo:lcl"}}
totalResultsAvailable STRING{@xmlelement { namespace = "urn:yahoo:lcl"}};
results Result[]{@XMLElement{name = "Result", namespace = "urn:yahoo:lcl"}};
end
record Result
Title STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Address STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
City STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
State STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Latitude STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
Longitude STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
rating Rating{@xmlElement{namespace = "urn:yahoo:lcl"}};
end
record Rating
AverageRating STRING{@xmlElement{namespace = "urn:yahoo:lcl"}};
end
   Related tasks
```

"Lesson 10: Create the UI for the Map portlet" on page 42 Create a portlet where you can enter a zip code and see a list of nearby mortgage companies and a map. Click the name of a company, and the map displays the location of the company.

Finished code for MapLocatorHandler.egl after Lesson 10

The following code is the complete text of the MapLocatorHandler.egl file at the end of Lesson 10. The code includes tabs, comments, and blank lines. For the most part, this code was created by the EGL visual editor. If you see errors in your source file, compare your code with this version.

package handlers;

```
// RUI Handler
```

import com.ibm.egl.rui.widgets.Box; import com.ibm.egl.rui.widgets.TextLabel; import com.ibm.egl.rui.widgets.TextField; import dojo.widgets.DojoButton;

```
//
//
```

handler MapLocatorHandler type RUIhandler {initialUI = [ui],onConstructionFunction = start, cssF

```
ui Box{ columns = 1, width = "400", height = "400", children = [ introLabel, formBox, mapBox ],
id = "form" };
introLabel TextLabel{ text = "Search for local mortgage businesses" };
formBox Box{ padding=8,
columns = 3,
children = [ zipLabel, zipField, zipButton ] };
zipLabel TextLabel{ text = "Zip code:" };
zipField TextField{
 text = "10001", onKeyDown ::= checkForEnter
};
zipButton DojoButton{ text = "Search", onClick ::= buttonClicked };
mapBox Box{ padding=8,
children = [ listingBox ] };
listingBox Box{ padding=8,
columns = 1,
width = "120" };
function start()
end
function checkForEnter(event Event in)
end
function buttonClicked(event Event in)
end
end
```

```
Related tasks
```

"Lesson 10: Create the UI for the Map portlet" on page 42 Create a portlet where you can enter a zip code and see a list of nearby mortgage companies and a map. Click the name of a company, and the map displays the location of the company.

Finished code for MapLocatorHandler.egl after Lesson 11

The following code is the complete text of the MapLocatorHandler.egl file at the end of Lesson 11. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version. package handlers;

import com.ibm.egl.rui.widgets.Box; import com.ibm.egl.rui.widgets.HyperLink;

```
import com.ibm.egl.rui.widgets.Image;
import com.ibm.egl.rui.widgets.TextField;
import com.ibm.egl.rui.widgets.TextLabel;
import com.ibm.egl.rui.widgets.dialog.DialogLibrary;
import egl.core.restbinding;
import eql.ui.rui.Event;
import dojo.widgets.DojoButton;
import interfaces.ResultSet;
import interfaces.YahooLocalService;
import widgets.GoogleMap;
handler MapLocatorHandler type RUIhandler {initialUI = [ ui ],onConstructionFunction = start, cssFil
const YAHOO APP ID STRING = "app id";
lookupService YahooLocalService{@restbinding};
ui Box{ columns = 1, width = "400", height = "450", children = [ introLabel, formBox, mapBox ],
id = "form" };
 introLabel TextLabel{ text = "Search for local mortgage businesses" };
 formBox Box{ padding=8,
children = [ zipLabel, zipField, zipButton ],
columns = 3 };
zipLabel TextLabel{ text = "Zip code:" };
zipField TextField{
 text = "10001", onKeyDown ::= checkForEnter
};
zipButton DojoButton{ text = "Search", onClick ::= buttonClicked };
mapBox Box{ padding=8,
children = [ listingBox, localMap ] };
listingBox Box{ padding=8,
columns = 1,
width = "120" }:
localMap GoogleMap { width = 500, height = 350 };
function start()
 search(); // show search results
 end
 function checkForEnter(event Event in)
 if(event.ch == 13)
  search();
  end
end
function buttonClicked(event Event in)
 search();
 end
 function search()
 listingBox.setChildren([new Image{src = "icons/progress3.gif"}]);
 localMap.showAddress(zipField.text, ""); // default map (no address)
 // Call remote Yahoo Service and pass zip code
 call lookupService.getSearchResults(YAHOO APP ID, zipField.text)
  returning to showResults onException displayError;
 end
 function displayError(ex AnyException in)
 DialogLibrary.showError("Yahoo Service", "Cannot invoke Yahoo Local Service: " + ex.message, null)
end
 function showResults(retResult ResultSet in)
 linkListing HyperLink[0];
 for(i INT from 1 to retResult.results.getSize() by 1)
  newLink HyperLink{text = retResult.results[i].title, href = "#"};
  newLink.setAttribute("address", retResult.results[i].Address + ", "
     + retResult.results[i].city + ", "
    + retResult.results [i].state);
```

```
newLink.setAttribute("title", retResult.results[i].Title);
newLink.onClick ::= mapAddress;
linkListing.appendElement(newLink);
end
listingBox.setChildren(linkListing);
end
function mapAddress(e Event in)
// Show the address on the map when the link is clicked
businessAddress STRING = e.widget.getAttribute("address") as STRING;
businessName STRING = e.widget.getAttribute("title") as STRING;
localMap.showAddress(businessAddress, "<b>" + businessName + "</b>");
end
end
Related tasks
```

"Lesson 11: Create the source code for the Map portlet" on page 48 Connect the service and the external widget to the user interface you created in the previous lesson.

Finished code for MainHandler.egl after Lesson 12

The following code is the complete text of the MainHandler.egl file at the end of Lesson 12. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version.

package handlers;

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.portal.Portal;
import com.ibm.egl.rui.widgets.portal.Portlet;
import egl.ui.columns;
handler MainHandler type RUIhandler {initialUI = [ui],onConstructionFunction = start, cssFile="css
ui Box{ columns = 1, children = [mortgagePortal]};
mortgagePortal Portal {columns = 2, columnWidths = [300, 650]};
calculatorHandler MortgageCalculatorHandler{};
resultsHandler CalculationResultsHandler{};
historyHandler CalculationHistoryHandler{};
mapHandler MapLocatorHandler{};
calculatorPortlet Portlet{children = [calculatorHandler.ui], title = "Calculator"};
resultsPortlet Portlet{children = [resultsHandler.ui],
  title = "Results", canMove = TRUE, canMinimize = TRUE};
historyPortlet Portlet{children = [historyHandler.ui],
  title = "History", canMove = TRUE, canMinimize = TRUE};
mapPortlet Portlet{children = [mapHandler.ui],
  title = "Map", canMove = false, canMinimize = true };
function start()
 mortgagePortal.addPortlet(calculatorPortlet, 1);
 mortgagePortal.addPortlet(resultsPortlet, 1);
 mortgagePortal.addPortlet(historyPortlet, 1);
 mortgagePortal.addPortlet(mapPortlet, 2);
  // Subscribe to calculation events
  InfoBus.subscribe("mortgageApplication.mortgageCalculated", restorePortlets);
 // Initial state is minimized
 resultsPortlet.minimize();
 historyPortlet.minimize();
 end
```

```
function restorePortlets(eventName STRING in, dataObject any in)
if(resultsPortlet.isMinimized())
resultsPortlet.restore();
end
if(historyPortlet.isMinimized())
historyPortlet.restore();
end
end
end
```

Related tasks

"Lesson 12: Add the Map portlet to the main portal" on page 51 To add the new portlet to your page, you must change the main portal.

Finished code for MortgageCalculatorHandler.egl after Lesson 15

The following code is the complete text of the MortgageCalculatorHandler.egl file after Lesson 15. The code includes tabs, comments, and blank lines. If you see errors in your source file, compare your code with this version.

```
package handlers;
```

```
import com.ibm.egl.rui.infobus.InfoBus;
import com.ibm.egl.rui.mvc.Controller;
import com.ibm.egl.rui.mvc.FormField;
import com.ibm.egl.rui.mvc.MVC;
import com.ibm.egl.rui.mvc.ValidatingForm;
import com.ibm.egl.rui.widgets.Box;
import com.ibm.egl.rui.widgets.Combo;
import com.ibm.egl.rui.widgets.HTML;
import com.ibm.egl.rui.widgets.Image;
import com.ibm.egl.rui.widgets.TextField;
import com.ibm.egl.rui.widgets.TextLabel;
import egl.ui.displayname;
import egl.ui.rui.Event;
import dojo.widgets.DojoButton;
import services.MortgageCalculationResult;
import services.MortgageCalculationService;
```

handler MortgageCalculatorHandler type RUIhandler {initialUI = [ui],onConstructionFunction = start

```
mortService MortgageCalculationService{};
ui Box{ columns = 1, children = [ inputForm, buttonBox, paymentForm, errorDisplay ], id = "form" };
```

```
inputRec MortgageCalculationResult{
 loanAmount = 180000,
 interestRate = 5.2,
 term = 30.
 monthlyPayment = 0,
 interest = 0
};
inputForm ValidatingForm{width=530, entries = [
new FormField{displayName="Amount:", controller=amountController},
new FormField{displayName="Rate:", controller=rateController},
new FormField{displayName="Term:", controller=termController}
1};
amountController Controller{@MVC{model = inputRec.loanAmount, view = amountField}};
amountField TextField{};
rateController Controller{
 formatters ::= formatRate, unformatters ::= unformatRate,
 @MVC{model = inputRec.interestRate, view = rateField}
};
rateField TextField{};
termController Controller{
```

```
retrieveModelHelper = getTermModel,
 retrieveViewHelper = getTermCombo,
 publishHelper = setTermCombo,
 commitHelper = setTermModel,
 @MVC{model = inputRec.term, view = termCombo}};
const TERM VALUES String[] = ["5","10","15","30"];
termCombo Combo{ values = TERM VALUES, selection = 4 };
blankLabel TextLabel{};
buttonBox Box{ padding=8,
children = [ blankLabel, calculationButton, processImage ] };
calculationButton DojoButton{ text = "Calculate", onClick ::= calculate };
processImage Image{
src = "icons/progress3.gif",
visibility = "hidden"
};
paymentField TextLabel{};
paymentController Controller{@MVC{model = inputRec.monthlyPayment, view = paymentField}};
paymentForm ValidatingForm{width=530, entries = [
 new FormField{displayName = "Payment:", controller = paymentController}
]
};
// use for error messages
error STRING = "";
errorDisplay HTML{
color = "Red"
};
// associate MVC with errorDisplay widget
errorController Controller{@MVC{model = error, view = errorDisplay}};
function start()
hidePaymentForm();
end
function calculate(e Event in)
 showProcessImage();
 calculateMortgage();
end
function showProcessImage()
processImage.visibility = "visible";
end
function hideProcessImage()
processImage.visibility = "hidden";
end
function showPaymentForm()
paymentForm.visibility = "visible";
end
function hidePaymentForm()
paymentForm.visibility = "hidden";
end
function calculateMortgage()
 if (inputForm.isValid())
  inputForm.commit();
  inputForm.publish();
  call mortService.amortize(inputRec) returning to displayResults
     onException handleException;
 else
 hideProcessImage();
 end
end
```

Create a mortgage portal with EGL Rich UI 77

```
function displayResults(retResult MortgageCalculationResult in)
 inputRec.monthlyPayment = retResult.monthlyPayment;
 paymentForm.publish();
 hideProcessImage();
 showPaymentForm();
 InfoBus.publish("mortgageApplication.mortgageCalculated", retResult);
 end
private function setError(err STRING in)
 error = err;
 errorController.publish();
end
// catch-all exception handler
private function handleException(ae AnyException in)
 setError("Error calling service: " :: ae.message);
end
 function formatRate(input string in) returns(string)
 return(input+ "%");
end
 function unformatRate(input string in) returns(string)
 len int = strlib.characterLen(input);
 if (input[len:len] == "%")
  return(input[1:len-1]);
 end
 return(input);
 end
 function setTermCombo(value STRING in)
 // Set the term combo box to the index corresponding by the value of the selected term
 for (i int from 1 to TERM VALUES.getSize() by 1)
  if(TERM VALUES[i] == value)
   termCombo.selection = i;
   exit for;
  end
 end
end
 function getTermCombo() returns(STRING)
 // Return the selected index of the term combo box
 return (termCombo.selection);
end
 function setTermModel(value string in)
 // value represents the index in the combo
 if(value >= 1 && value <= TERM VALUES.getSize())</pre>
  inputRec.term = TERM_VALUES[value as int];
 end
end
function getTermModel() returns(STRING)
 // returns the term value
 return(TERM VALUES[termCombo.selection]);
end
end
   Related tasks
```

Related tasks

"Lesson 15: (Optional) Use validating forms in the Calculation portlet" on page 59

Replace the simplified code in MortgageCalculatorHandler.egl with code that uses the EGL Model-View-Controller (MVC) framework.

Notices

© Copyright IBM Corporation 2000, 2010.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan, Ltd. 1623-14, Shimotsuruma, Yamato-shi Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software 3600 Steeles Avenue East Markham, ON Canada L3R 9Z7

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Trademark acknowledgments

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/ copytrade.shtml.

Index

Α

Apache Tomcat installing 52 applications running from generated code 56

С

calculate() 27 calculateMortgage() 28 CalculationHistoryHandler overview 36 CalculationHistoryHandler.egl source code 70 CalculationResultsHandler.egl source code 69

D

deployment descriptor, EGL editing 55 displayResults() 28 Dojo Widget library overview 5

Ε

expenseGrid overview 30

G

generatable parts overview 11 generated code running 56

Η

Handler, EGL Rich UI definition 5 hideProcessImage() 28 IYahooLocalService.egl source code lesson 9 72

Μ

MainHandler overview 33 MainHandler.egl source code lesson 12 75 lesson 7 69 lesson 9 71 MapLocatorHandler overview 42 MapLocatorHandler.egl source code lesson 10 73 lesson 11 73 MortgageCalculationService.egl source code Lesson 3 65 MortgageCalculatorHandler.egl source code lesson 15 76 lesson 4 66 lesson 5 67 MVC forms 59

Ρ

packages creating 7 parts overview 11 portal widget importing 7 portals definitions 1 portlet definitions 1 Project Interchange File overview 3 projects creating 7

R

Record parts creating 14 REST protocols definitions 1 Rich UI Handler definition 5 Rich UI Handler partparts creating 16 Rich UI Handlers deploying 55

S

Service parts creating 12 overview 11 showProcessImage() 27 sketch UI 4 SOAP protocols definitions 1

Т

Tomcat Web server installing 52 tutorials Format logon page 3

W

widgets overview 5