



Format a Rich UI login page

Contents

Format a Rich UI logon page. 1

Lesson 1: Design your Web page	1
Chart the page structure	3
Lesson checkpoint	4
Lesson 2: Meet the Rich UI widgets and handlers . .	5
Widget properties.	5
Essential widgets	6
The Rich UI Handler	6
Lesson checkpoint	7
Lesson 3: Create an EGL framework	7
Create an EGL Rich UI project	7
Create a Rich UI Handler	9
Lesson checkpoint	10
Lesson 4: Use box widgets to lay out a page . . .	10
Use box widgets for layout	10

Add controls to the layout	14
Lesson checkpoint	17
Lesson 5: Use columns to lay out a page	17
Use columns for layout	17
Lesson checkpoint	20
Lesson 6: Format the page	20
Use properties to format boxes	21
Use properties to format content	22
Use CSS to format content	23
Lesson checkpoint	24
Lesson 7: Work with EGL source code	24
Create a text field in source code	24
Bind a simple function to the Submit button . .	26
Lesson checkpoint	28
Summary	28

Format a Rich UI logon page

In this tutorial, you will learn how to plan a simple Rich UI page and create the page by using the EGL Rich UI editor.

Learning objectives

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

- Design a Web page for Rich UI
- Create an organization chart for the elements on the page
- Understand widgets
- Understand the Rich UI Handler
- Create an EGL Rich UI project
- Create a Rich UI Handler
- Structure a page by using box widgets
- Structure a page by using columns only
- Format the elements on the page by using EGL properties
- Format the elements on the page by using cascading style sheets (CSS)
- Add widgets to the page by writing EGL code rather than by using the graphical interface
- Trigger a function by pressing a button

Time required

60 minutes

Other EGL tutorials:

Introducing EGL

Create a hello world program with EGL

Create a hello world service with EGL

Build a JSF search page with EGL

Access relational databases with EGL

Lesson 1: Design your Web page

Before you begin to write code, carefully plan both the overall appearance of the page and the way you might divide the page into self-contained regions.

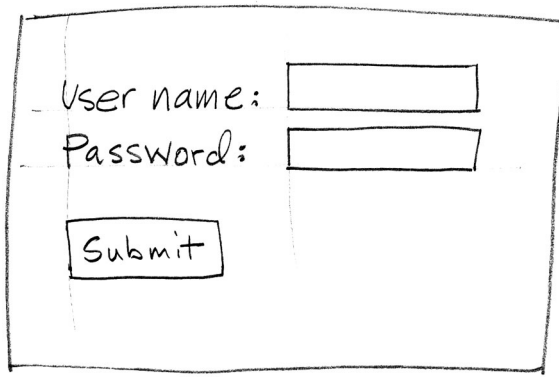
About this task

The first step in creating any Web page is to sketch the finished product. To create the sketch, you must answer a few basic questions:

- What does the page need to accomplish?
- What elements does the page need to do this work?
- How should you arrange those elements to make using the page as easy as possible?

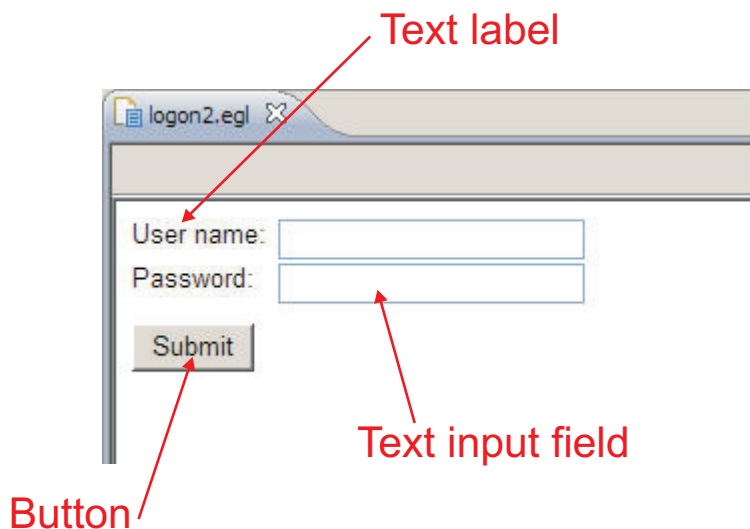
- Do you want to add graphics, such as logos, photos, decorative borders, or other purely visual elements, such as colored backgrounds, to make the page more appealing?

The target Web page that you will build in this tutorial is simple: a logon screen with ID and password fields and a Submit button.



When you use EGL Rich UI to design a page, each element in the page must be inside a container called a *box*. A box can contain multiple elements, including other boxes. On the target Web page, you can see the following page elements:

- Text labels
- Text input fields
- Buttons and other controls (such as a Submit button)



You use boxes to help control the layout of the page. There are few rules for how to design with boxes; most needs have multiple solutions, some of them equally "correct." You typically use boxes to align elements on the page. You need to use boxes because Rich UI has the following limitations:

- No tab characters. You cannot use tabs to align elements inside a box.
- No newline characters. You cannot artificially break a row of elements inside a box.
- No trailing spaces. You cannot pad text fields with spaces.

The following illustrations show several valid solutions to the problem of laying out the target page.

Illustration 1 shows a login form layout. The form is contained within a red rectangular box. It has a title bar at the top with a file icon and the text 'logon2.egl'. The form itself has a light gray background. Inside, there are three labels: 'User name:', 'Password:', and 'Submit'. The 'User name:' and 'Password:' labels are on the left, and the 'Submit' button is on the right. The input fields for the username and password are on the right, aligned with their respective labels.

Illustration 2 shows a login form layout. The form is contained within a red rectangular box. It has a title bar at the top with a file icon and the text 'logon2.egl'. The form itself has a light gray background. Inside, there are three labels: 'User name:', 'Password:', and 'Submit'. The 'User name:' and 'Password:' labels are on the left, and the 'Submit' button is on the right. The input fields for the username and password are on the right, aligned with their respective labels.

Illustration 3 shows a login form layout. The form is contained within a red rectangular box. It has a title bar at the top with a file icon and the text 'logon2.egl'. The form itself has a light gray background. Inside, there are three labels: 'User name:', 'Password:', and 'Submit'. The 'User name:' and 'Password:' labels are on the left, and the 'Submit' button is on the right. The input fields for the username and password are on the right, aligned with their respective labels.

Chart the page structure

About this task

A useful intermediate step between the sketch and the code is the construction of a sort of genealogical chart that shows all the boxes and elements for your page in hierarchical order. Again, this is something you can draw on a piece of scratch paper or a marker board.

In keeping with the idea of a family tree, two metaphors are typically used in describing such charts:

- The tree. In this case, the tree is upside down, with the root on top and leaves on the bottom.

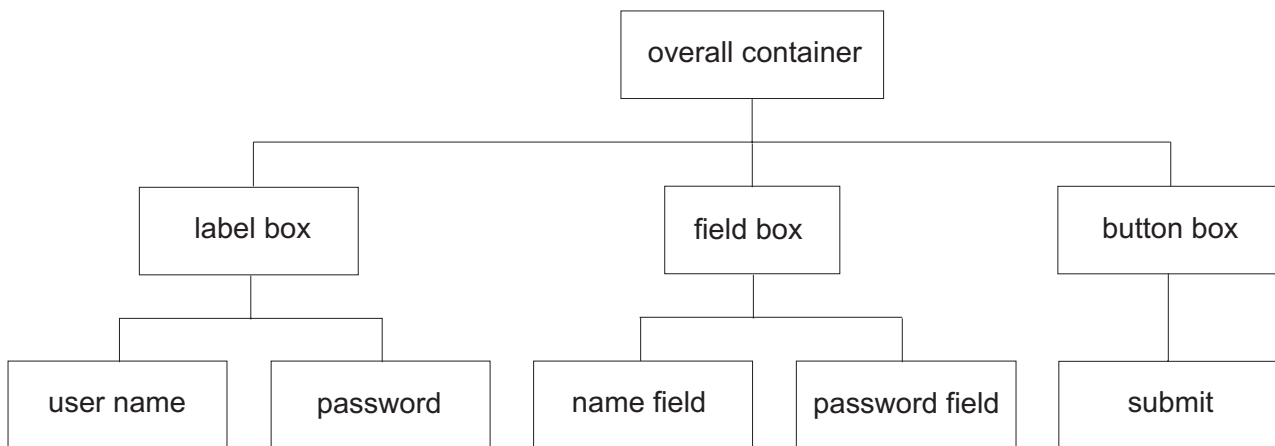
- The family. When an element on the screen is contained within another element, the container is the parent and the element inside the container is the child.

All such charts begin with a box that encloses all the other boxes and elements. This box is the root of the tree and represents the browser window.

Draw a box for each container and element on the page:

- Each element has exactly one *parent*, except for the root element, which has none.
- An element can have *siblings*, that is, other elements with the same parent.
- An element can have one or more *children*.
- An element with no children is called a *leaf* (going back to the tree metaphor).

For example, here is the second version of the target page layout in family tree form.



You may encounter the term Document Object Model (DOM) in connection with such a family tree. The DOM uses standard terms to name each element in an HTML or XML page, organizes the elements in the same kind of hierarchical chart, and provides an interface for changing such a page. The charts that you just created use the same organizational principles as the DOM.

In EGL Rich UI, each element on the page is called a *widget*. You will learn about widgets in the next lesson.

Lesson checkpoint

You have now sketched several different approaches to coding the target Web page.

In this lesson, you learned the following principles:

- A Rich UI Web page is composed of elements.
- Elements must be enclosed by containers.
- The containers must be in a hierarchical order, starting with the root.
- The root element is the browser window.

Lesson 2: Meet the Rich UI widgets and handlers

The elements in an EGL Rich UI page are called *widgets*. Widgets work with a type of program called an EGL Rich UI Handler.

About this task

Now that you have a family tree for the target Web page that you are designing, you can add some information about the way the individual elements, also known as *nodes*, should behave.

Widget properties

About this task

In this tutorial, *widget* means "window gadget." In EGL Rich UI, a widget is a reusable user interface component such as a button, a text input field, or a box. The elements in your target Web page are all widgets in the standard Rich UI palette.

In EGL, *properties* describe the behavior of elements in a program. A property is a name-value pair; in the code, the properties are enclosed in braces:

```
outerBox com.ibm.egl.rui.widgets.Box{ padding=8,  
    children = [ upperBox, lowerBox ],  
    columns = 1 };
```

In this example, **padding**, **children**, and **columns** are properties.

All widgets have properties, though the properties that are available depend on the specific widget. This tutorial uses the following properties that are associated with a box widget:

backgroundColor

Specifies a color to be applied to the background of a box.

borderColor

Specifies a color for a decorative border around the box.

borderStyle

Specifies the shape of the border.

borderWidth

Specifies the width of the decorative border, in pixels.

children

Lists all of the elements that appear inside the box. This might include buttons, text, entry fields, or other boxes.

class References a class in a cascading style sheet (CSS) that determines the formatting of elements within the box, such as the style of a font.

columns

Specifies the number of columns that divide the box, as an integer. You use columns to align elements.

padding

Specifies a distance, in pixels, between the contents and the border of a box.

Other properties specify further information about the box, such as:

- its location
- its ID
- its visibility

Essential widgets

About this task

In addition to the box widget, you will use the following widgets in this tutorial:

TextLabel

Defines a string that the user cannot change. You need the following TextLabel property for this tutorial:

text Specifies the string to be displayed.

TextField

Defines a text box in which a user can type a single line of input. (For lengthy input, Rich UI provides a TextArea widget.) You will not need any TextField properties for this tutorial.

PasswordTextField

Defines a text box where a user can type a single line of input. The value of the field is displayed as a string of bullet characters (•). You will not need any PasswordTextField properties for this tutorial.

Button

You can bind a function to a button; when the user clicks the button, the bound function is called. You need the following Button property for this tutorial:

text A string to be displayed on the button.

The Rich UI Handler

About this task

In EGL, a *handler* is a special kind of program with functions that are tied to specific events that occur when someone uses an interface. Technically, a Rich UI Handler is an EGL Handler part with the RUIHandler stereotype. In a Rich UI Handler, the widgets typically provide the events that are bound to the handler functions. The Rich UI Handler has the following properties:

initialUI

Specifies an array of widgets that makes up the initial display for the Web page.

onConstructionFunction

Specifies a function to call when the interface is first created.

cssFile

Specifies a cascading style sheet (CSS) to assign to the file. By default, EGL creates a CSS with the same name as the Rich UI Handler, and defines a few basic styles. To customize the appearance of the page, you can modify this file or assign a different one to the **cssFile** property.

The widget code that is included with EGL describes a set of abstract entities, which are like blueprints for controls on a page. To create the actual control, declare a variable that has the *type* of the widget you want to create. These

declarations go in the Rich UI Handler. For example, if you create a Box widget, you must assign a name to it, such as topBox. The declaration for this widget is included in the Rich UI Handler:

```
topBox com.ibm.egl.rui.widgets.Box;
```

You can create a Rich UI program by coding a Rich UI Handler one line at a time, or you can drag visual elements onto a representation of the Web page using the EGL Rich UI editor. That editor is the subject of the next lesson, in which you begin to build a working interface.

Lesson checkpoint

In this lesson, you learned about widgets and handlers. This is the last section of the tutorial that focuses on the concepts of Rich UI.

You learned the following concepts:

- What a widget is
- The important properties of each widget
- The types of widgets you will use in the tutorial and their important properties
- What a Rich UI Handler does

Lesson 3: Create an EGL framework

Use the graphical interface in the Rich UI editor to familiarize yourself with the components you need to create a Web page.

About this task

First, create an EGL Rich UI project where you can use the Rich UI editor. Next, create a Rich UI Handler for the project. When you open the Rich UI Handler in the Rich UI editor, you can drag widgets onto the virtual page and bind them to functions.

Create an EGL Rich UI project

About this task

You can use projects:

- To organize your EGL source files.
- To provide shortcuts and defaults tailored to a particular type of work.

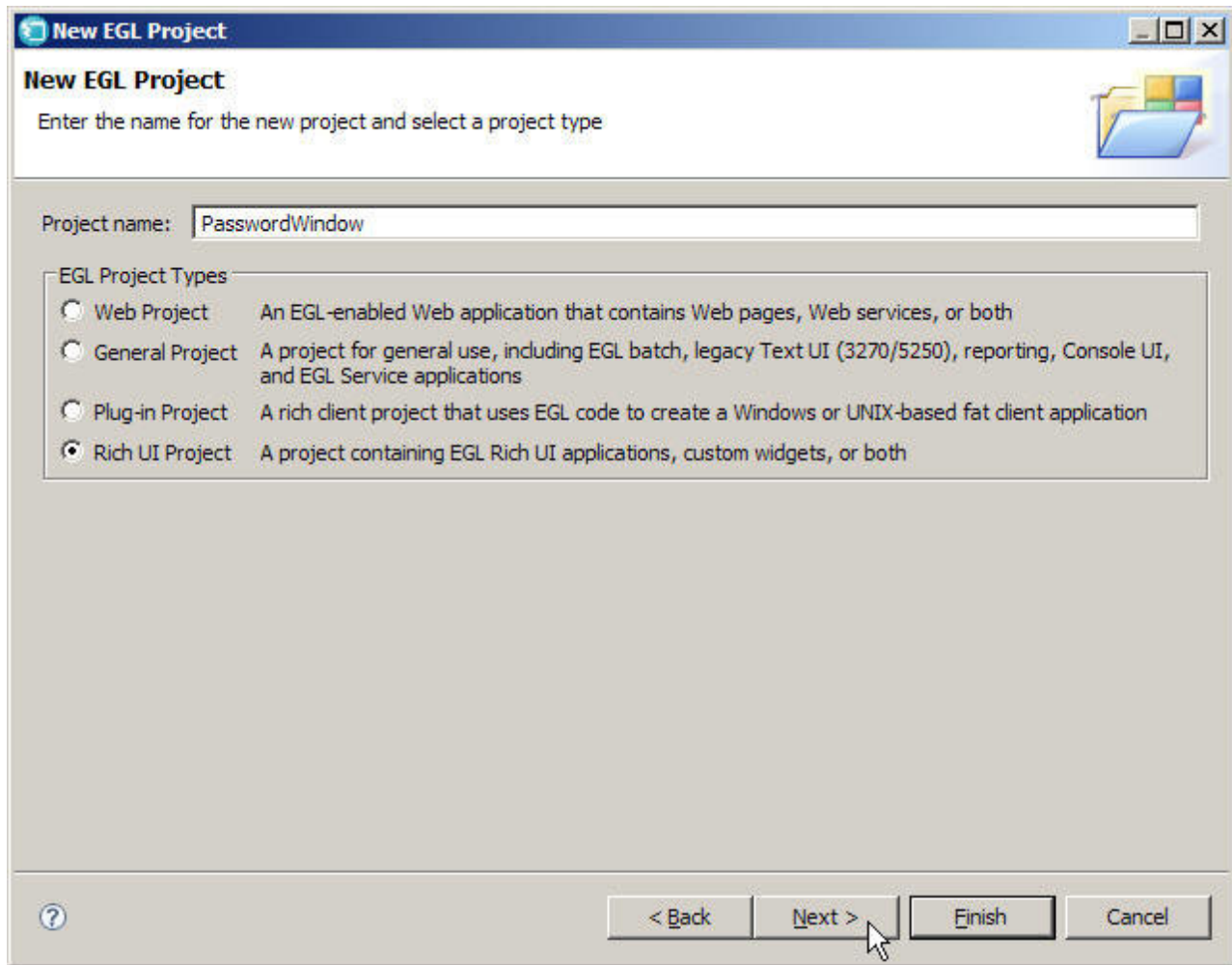
When you create a Rich UI project, EGL sets up only the folders and source files that are appropriate for this type of programming.

The following demonstration shows the steps involved in this task:

Show Me

To create an EGL Rich UI project:

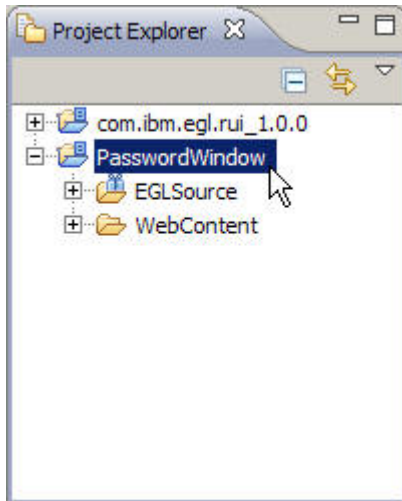
1. Click **File** → **New** → **Project**. Expand **EGL**, click **EGL Project**, and click **Next**.
2. In the New EGL Project wizard, enter the following information:
 - a. In the **Project name** field, type the following name:
PasswordWindow
 - b. Under **EGL Project Types**, select **Rich UI Project**.



- c. Click **Next**.
3. On the advanced settings page, clear the **Create an EGL service deployment descriptor** check box. In this tutorial, you do not use services. Click **Finish**.

Results

EGL creates a new project named PasswordWindow. There are two folders inside the directory: one for EGL source code and one for the Web content that you will create.



Create a Rich UI Handler

About this task

Now that you have a project that will hold your files, create a source file for your Rich UI application.

The following demonstration shows the steps involved in this task:

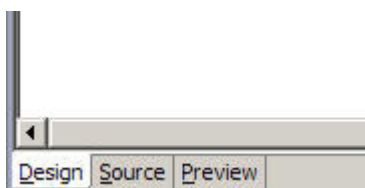
Show Me

To create a Rich UI Handler:

1. Right-click the **PasswordWindow** project, then click **New** → **Rich UI Handler**.
2. In the New Rich UI Handler part wizard, enter the following information:
 - a. Make sure the **Source folder** field specifies PasswordWindow\EGLSource.
 - b. For the package name, enter `ruihandlers`. EGL will create a folder that has this name in the EGL source folder.
 - c. For the file name, enter `logon1`. EGL will create a source file that has this name and the `.egl` extension.
 - d. Click **Finish**.

Results

EGL displays the new Rich UI Handler in the Design view of the Rich UI editor. In the Design view, you can use visual tools to edit the page. Note the tabs at the lower left of the display:



Later you will use the Source tab to create functions to bind to the widgets. Before that, you will create two different page layouts, one that uses columns and another that uses boxes.

Lesson checkpoint

You created a project that will hold your source files, and a source file for your logon screen.

You learned how to perform the following tasks:

- How to create an EGL project
- How to create an EGL Rich UI Handler; Rich UI Handlers are described in Lesson 2.

Lesson 4: Use box widgets to lay out a page

Use the EGL Rich UI editor to create a logon page using box widgets to control formatting.

About this task

EGL Rich UI follows the Visual Formatting Model of the World Wide Web Consortium (W3C). This recommendation discusses 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. This best practice allows the page to adapt more easily to different screen resolutions, browsers, fonts, and other factors that affect display.

In Rich UI, you typically use boxes to create rows, and then use columns to divide those rows.

Use box widgets for layout

About this task

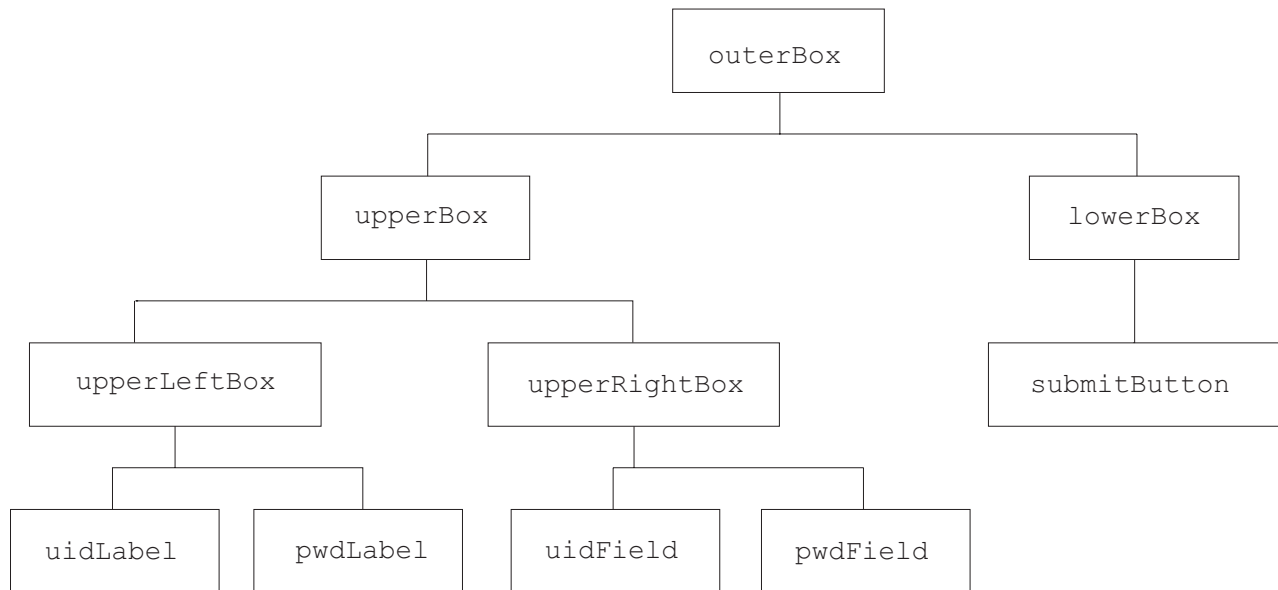
Refer to the sketches that you made in Lesson 1. Your first version of the logon screen will use boxes for formatting.

First you will create a single box to hold everything, called `outerBox`. Next, you will divide `outerBox` in half by creating an `upperBox` to contain the text fields and the labels associated with them, and a `lowerBox` to contain the Submit button. Next, you will create separate boxes within the `upperBox` for the text fields and the labels. Finally, you will add the label, text field, and button widgets to the appropriate boxes.

In this way, the hierarchy of the elements is clear:

- `outerBox` has two children:
 - `upperBox`
 - `lowerBox`
- `upperBox` has two children:
 - `upperLeftBox`
 - `upperRightBox`

And so on. The following chart shows the complete hierarchy:



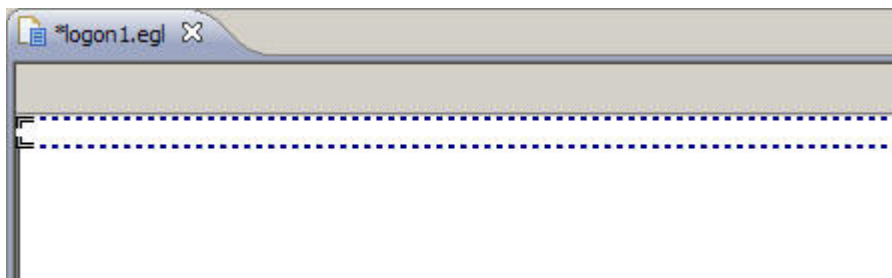
When you construct the layout that follows, you add boxes moving from left to right and top to bottom. By adding boxes in this order, you can be sure that you are not leaving areas of the outer box orphaned and inaccessible.

The following demonstration shows the steps in this task:

Show Me

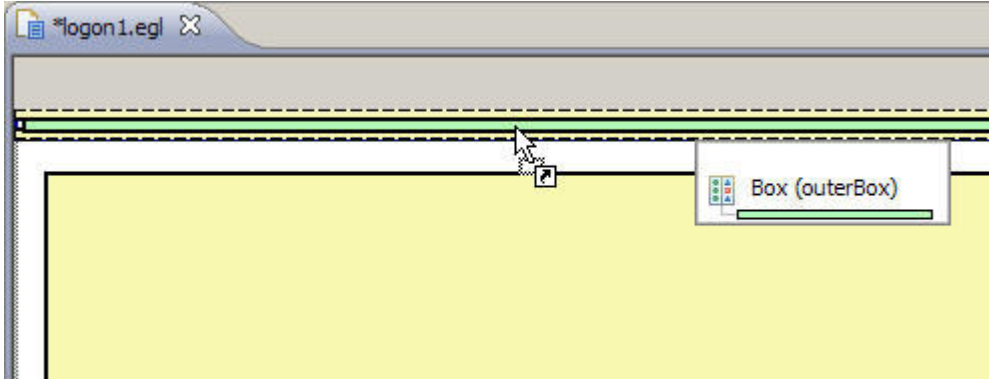
To create a layout by using boxes:

1. Make sure you are in the Design view for your Rich UI Handler (logon1.egl). Drag a Box widget from the **EGL Widgets** palette onto the screen. By default, the **Palettes** view is located to the left of the editor pane. The entire editor pane turns green, indicating that you can drop the widget anywhere on the surface. When you release your mouse button, the New Variable window appears.
2. In the New Variable window, in the **Variable name** field, enter the following string:
outerBox
3. Click **OK**. A thin, long box appears on the screen.

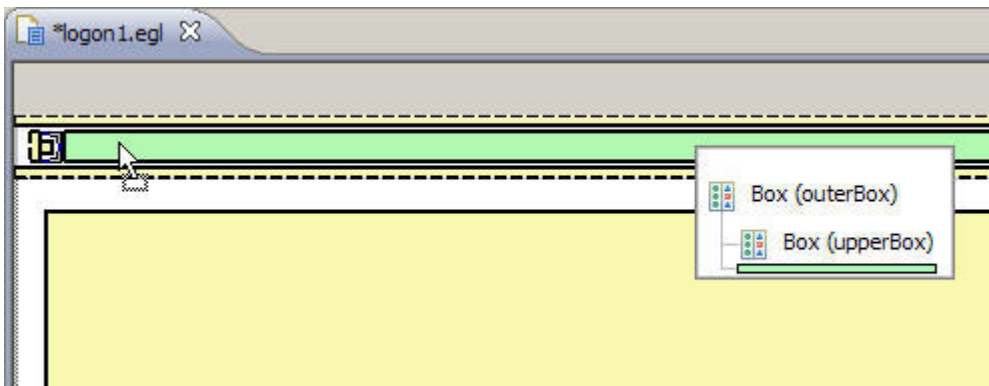


EGL creates a new variable in your Rich UI Handler named `outerBox` that is based on the `Box` widget type. Because this is the first container that you created for this program, EGL also sets `outerBox` as the value for the `initialUI` property.

4. Drag a second box widget onto the first. The `outerBox` surface turns green:



5. In the New Variable window, in the **Variable name** field, enter the following string:
`upperBox`
6. Click **OK**.
7. Drag a third box onto the `outerBox` widget, to the right of `upperBox`.



Study this picture for a moment. The yellow regions indicate locations where you can place a new box. The following locations are available:

- The thin yellow line at the top of the screen. This indicates a position that is outside of the `outerBox` widget and that precedes it in the visual design.
- The thin yellow line to the left of `upperBox`. This indicates a position that is inside the `outerBox` widget, but that precedes `upperBox` in the hierarchy.
- The larger green area to the right of `upperBox`. This position is also inside of the `outerBox` widget, but comes after `upperBox` in the hierarchy.
- The thin yellow line at the bottom of `upperBox`. This position is the same as that of the large yellow box at the bottom of the display. Both positions are outside of `outerBox` and after it in the hierarchy.

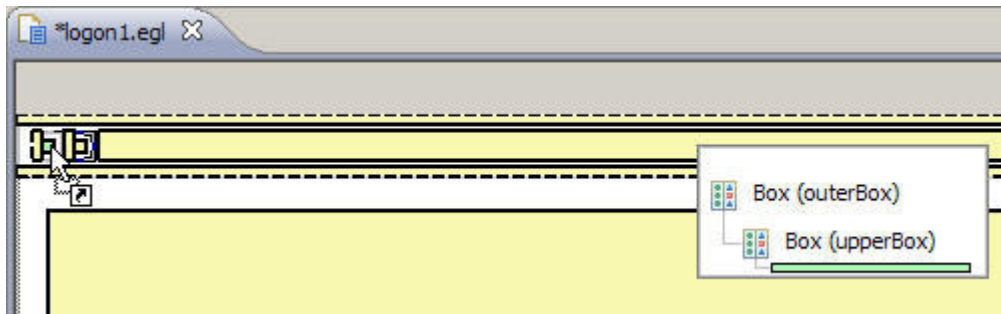
After you click the `Box` icon in the Palette, try dragging it to various locations in the layout without releasing the mouse button. Watch the pop-up box to see how the green line that represents the new box moves in relation to the other widgets.

You can also see the hierarchy of widgets in the **Outline** view, located in the lower left corner of the workbench by default. In this view, you can use your mouse to move the widgets; you can even drag widgets from the Palette onto the outline.

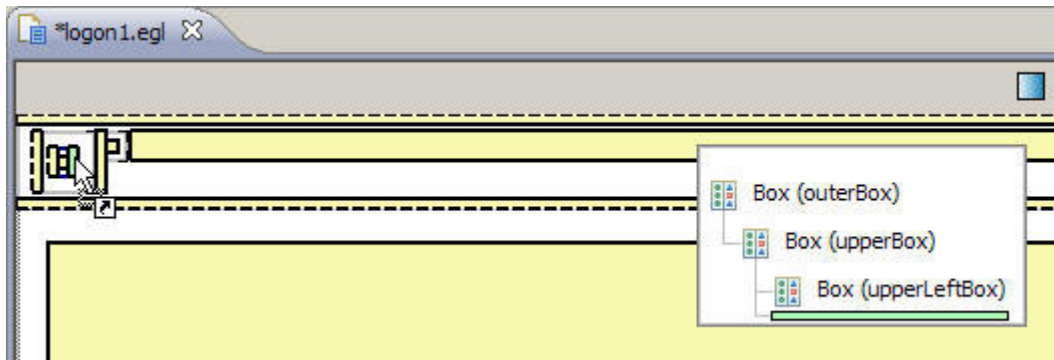
8. In the New Variable window, in the **Variable name** field, enter the following string:
lowerBox

Note that upperBox and lowerBox are displayed side by side. Later, you will use the **columns** property to adjust the positions of the boxes.

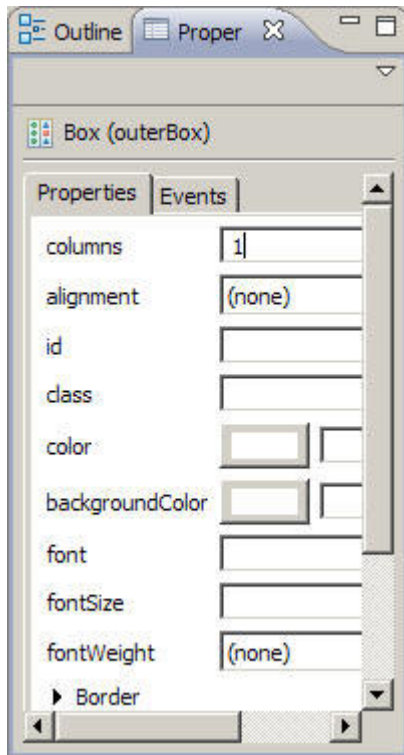
9. Click **OK**.
10. Drag a new box onto upperBox.



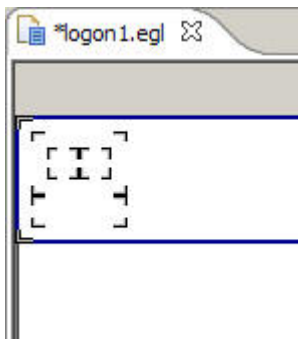
11. Enter the following string as the **Variable name**:
upperLeftBox
12. Click **OK**.
13. Drag a second box onto upperBox, to the right of upperLeftBox. Your target is a narrow line to the right of upperLeftBox. The pop-up box should show that your new box, which is represented by the green line, is a child of upperBox and a sibling of upperLeftBox.



14. Enter the following string as the **Variable name**:
upperRightBox
15. Click **OK**. You now have all the boxes you need to lay out the page. The next step is to arrange them.
16. Select the outerBox widget by clicking inside the box until the outline of the box is displayed as a dotted line. Locate the **Properties** view, which is located by default in the lower left corner of the workbench. Enter 1 in the **columns** field.



When the **columns** property is blank, EGL places all objects one after another, left to right. When you set the **columns** property to 1, you tell EGL to place elements in a single column, with each element below the previous.



Although the shape is small, it should look familiar; this is one of the layout patterns that you sketched in Lesson 1. Next, you will fill in the page elements.

17. Press Ctrl+S to save the file.

Add controls to the layout

About this task

Now that you have a layout defined by three separate boxes, you can add appropriate widgets to perform the work of the page.

The following demonstration shows the steps in this task:

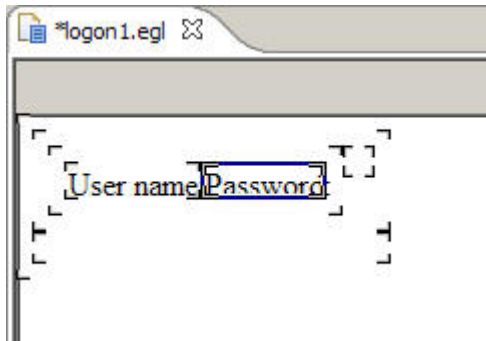
Show Me

To add widgets to the layout:

1. Drag a **TextLabel** widget onto the upperLeftBox widget.
2. In the New Variable window, in the **Variable name** field, enter the following string:
`uidLabel`
3. Click **OK**. The label is displayed on the page with the name **TextLabel**.
4. Make sure that `uidLabel` is selected and go to the **Properties** view. Change the **text** property to the following string:
`User name:`
5. Press Ctrl+S to save the file.
6. Drag a second **TextLabel** widget onto the upperLeftBox widget.
7. Enter the following string as the **Variable name**:
`pwdLabel`
8. Click **OK**. The label is displayed on the page with the name **TextLabel**.
9. With the `uidLabel` selected, go to the **Properties** view and change the **text** property to the following string:
`Password:`

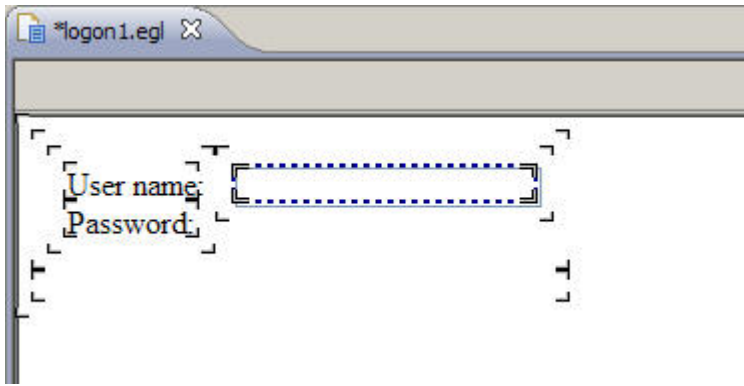
The label is displayed on the same line with the previous label.

This is not the behavior you want; however, you can control it with the

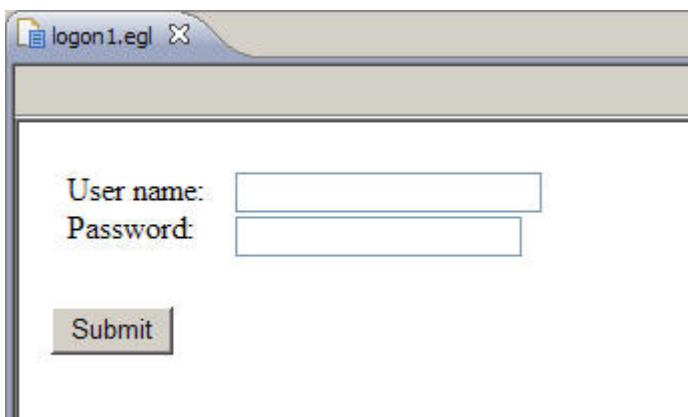


columns property of the containing box.

10. Click the white space surrounding the label to select the upperLeftBox widget. In the **Properties** view, set the **columns** property to 1.
11. Press Ctrl+S to save the file.
12. Drag a **TextField** widget onto the upperRightBox widget.
13. Enter the following string as the **Variable name**:
`uidField`
14. Click **OK**. A text entry field is displayed on the page as a small rectangle.



15. Drag a **PasswordTextField** widget onto the upperRightBox widget and enter the following name as the **Variable name**:
pwdField
16. Click **OK**. As with the text label, the second widget appears next to the first.
17. Click the white space surrounding the label to select the upperRightBox widget. In the **Properties** view, set the **columns** property to 1.
18. Press Ctrl+S to save the file.
19. Drag a **Button** widget to the lowerBox widget.
20. In the New Variable dialog, in the **Variable name** field, enter the following string; then click **OK**:
submitButton
21. Click **OK**.
22. Make sure that the new button is selected and go to the **Properties** view view. Change the **text** property to the following string:
Submit
23. Press Ctrl+S to save the file.
24. Click the **Preview** tab at the bottom of the editor window. The completed UI is displayed.



25. Compare this version of the logon window with the original sketch. Note the following issues:
 - The labels do not line up horizontally with their related fields.
 - The Submit button does not line up vertically with the labels.
 - The label font does not match the font on the button.

You can resolve the font issue by using widget properties, but the other issues are more difficult. The next lesson offers a different approach to laying out the page, one that addresses the first two issues.

26. Close the file by clicking the X icon on the tab that shows the file name (logon1.egl).

Results

Lesson checkpoint

You created a layout by using boxes, and then added widgets to those boxes.

You learned how to perform the following tasks:

- How to create boxes in a hierarchy
- How to select objects
- How to use the Properties view

Lesson 5: Use columns to lay out a page

Use the EGL Rich UI editor to create a logon page that uses the **columns** property to control formatting.

About this task

In the previous lesson, you forced elements to line up on the page by confining them in containers. This ultimately proved inefficient, as it took too much effort to control every aspect of the page. In this lesson, you will use the built-in behavior of the widgets to more efficiently accomplish the same task.

Use columns for layout

About this task

In a new file, you will create a version of the page that uses only a single box widget, which is required to hold the other elements. You will use less code to end up with a better result.

The following demonstration shows the steps in this task:

Show Me

To create a layout by using columns:

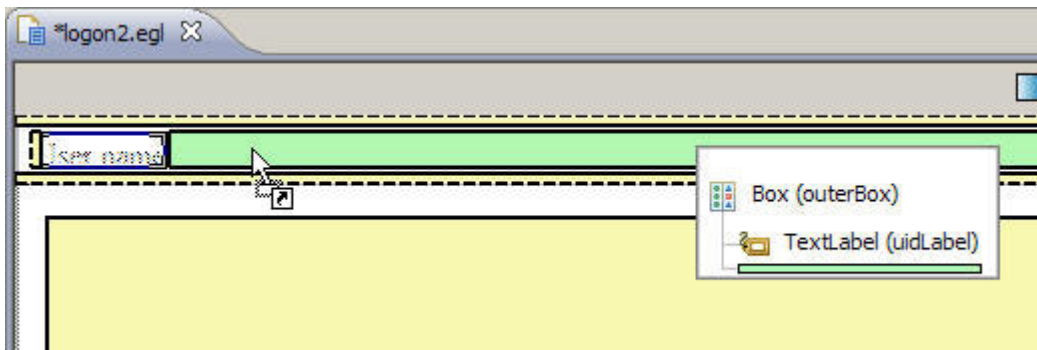
1. Create a second Rich UI Handler named logon2.egl:
 - a. Right-click the **PasswordWindow** project, then click **New** → **Rich UI Handler**.
 - b. For the package name, specify `ruihandlers`. For the file name, type `logon2`. Click **Finish**.

The new file eventually opens in the Rich UI editor.

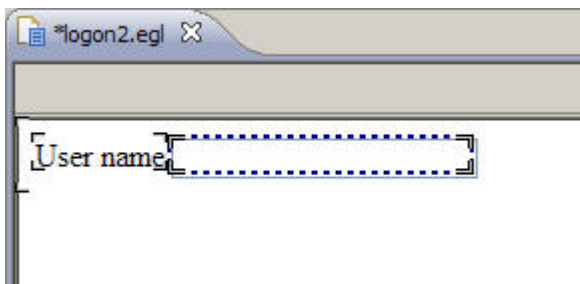
2. Make sure that you are in the Design view for your Rich UI Handler (logon2.egl). Drag a Box widget from the **EGL Widgets** palette onto the screen.
3. In the New Variable window, in the **Variable name** field, enter the following string:
`outerBox`

Note: Names must be unique within a handler only.

4. Click **OK**.
5. Drag a **TextLabel** widget onto the **outerBox** widget.
6. In the New Variable window, in the **Variable name** field, enter the following string:
uidLabel
7. Click **OK**. The label is displayed on the page with the name **TextLabel**.
8. Make sure that uidLabel is selected and go to the **Properties** view. Change the **text** property to the following string:
User name:
9. Drag a **TextField** widget onto the **outerBox** widget, after the uidLabel.

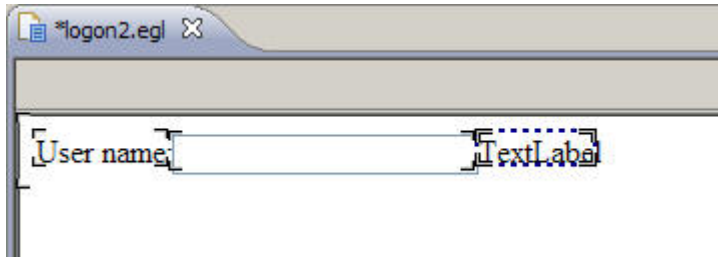


10. Enter the following string as the **Variable name**:
uidField
11. Click **OK**. A text entry field is displayed on the page as a small rectangle immediately after the label.



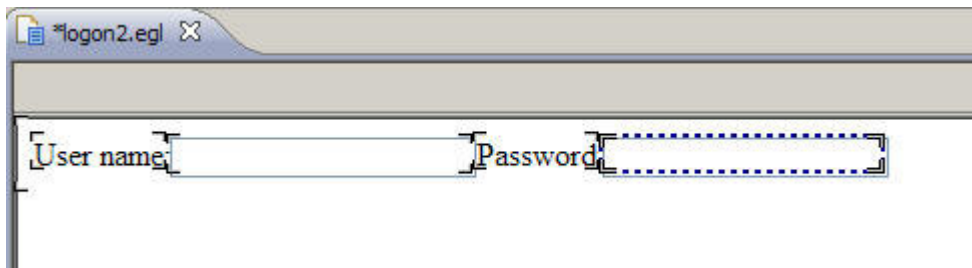
12. Press **Ctrl+S** to save the file.
13. Drag a second **TextLabel** widget onto the **outerBox** widget.
14. Enter the following string as the **Variable name**:
pwdLabel

The label is displayed on the page with the name **TextLabel**. The label appears on the same line with the previous two widgets.

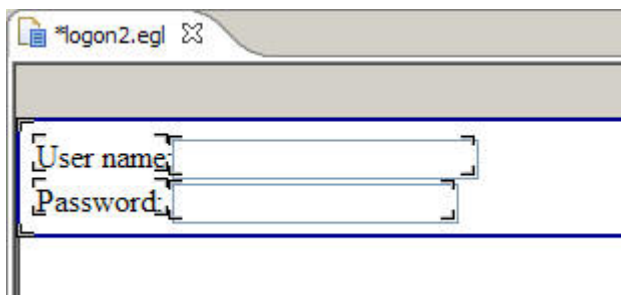


Before you correct this problem, update the label text and add the other input field.

15. In the **Properties** view, change the **text** property to the following string:
Password:
16. Click **OK**.
17. Drag a **PasswordTextField** widget onto the upperRightBox widget in the upper right quadrant of the page.
18. Enter the following string as the **Variable name**:
pwdField
19. Click **OK**. All four widgets are displayed on the same line.



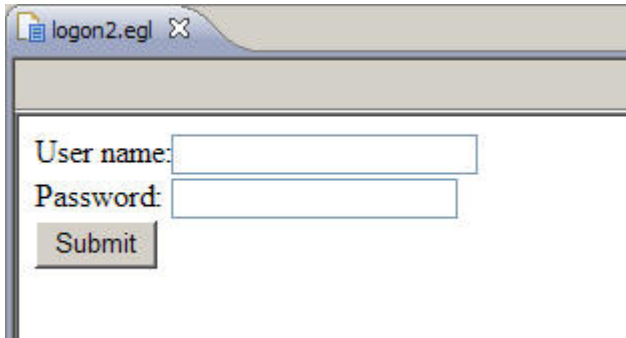
20. Click the white space surrounding the four new elements to select the original outerBox widget. In the **Properties** view, set the **columns** property to 2. The text entry fields (the empty rectangles) are now left aligned, though the entry field for the user name is uncomfortably close to the label.



21. Press Ctrl+S to save the file.
22. Drag a **Button** widget to the outerBox widget.
23. Enter the following string as the **Variable name**:
submitButton
24. Click **OK**.
25. Make sure that the new button is selected and go to the Properties view. Change the **text** property to the following string:

Submit

26. Press Ctrl+S to save the file.
27. Click the **Preview** tab at the bottom of the editor window. The completed UI is displayed.



Notice that the labels are better aligned with the corresponding input fields. The left margin of the second column, which holds the input fields, begins at the right edge of longest element in the first column.

The page can still be improved. For example, the `uidLabel` is too close to the `uidField` and the **Submit** button is too close to the field labels. Also, the font on the labels does not match the font on the button. You can fix these and other issues by using various widget formatting properties, which you will learn about in the next lesson.

28. Close the file by clicking the X icon on the tab that shows the file name (logon1.egl).

Lesson checkpoint

You just recreated the logon page, using columns to format the elements on the page.

You learned more about the way EGL uses columns to format a page:

- In a two-column format, EGL places elements alternately in each column.
- The default width of a column is determined by the widest element within it.
- You can use columns to solve many basic layout problems, giving you smaller programs than you would have if you used boxes for formatting.

In this case, using columns alone proved to be a better solution than using boxes. However, you might find other situations where boxes are necessary to format the page properly.

Lesson 6: Format the page

You can use the EGL Rich UI editor **Properties** view to format the elements in your Rich UI Handler.

About this task

Most of these properties have CSS equivalents. After you experiment with page formatting by using the Rich UI editor properties, try to make changes to the PasswordWindow project CSS file to accomplish the same effects.

This lesson uses the following properties:

- backgroundColor
- border
- fontSize
- fontStyle
- padding

Use properties to format boxes

About this task

When you use properties, be aware of which widget you have selected in the editor. For example, if you change the font for a box, you change all text elements within the box to the new font. If you meant to change the font for a label only, you might end up with a surprise.

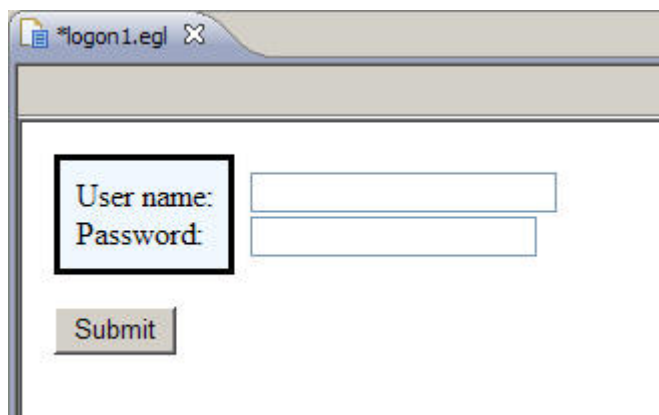
The first exercise use the logon1 program. The multiple boxes in this program make some aspects of the formatting clearer.

The following demonstration shows the steps involved in this task:

Show Me

To format boxes using properties:

1. Open logon1.egl in the EGL Rich UI editor.
2. In the Design view, select the upperLeftBox widget.
3. In the Properties view, click the button for **backgroundColor**. In the Color selection window, click **Name format** to choose a color by name from the center section of the window. Select a pastel color such as **AliceBlue** and click **OK**. To remove the color, you must click **Custom** and not enter a value.
4. Expand the **Border** group and make the following changes:
 - a. Click the **borderColor** button. In the Color selection window, click **Name format** to choose a color by name. Select **Black** and click **OK**.
 - b. For **borderWidth**, enter the number 3. This represents 3 pixels; unlike other properties that take a unit of measure, such as **fontSize**, **borderWidth** accepts numerals only.
 - c. For **borderStyle**, select **Solid**.
 - d. Switch to **Preview** mode, where you should see the following layout:



5. Save and close the file.

Use properties to format content

About this task

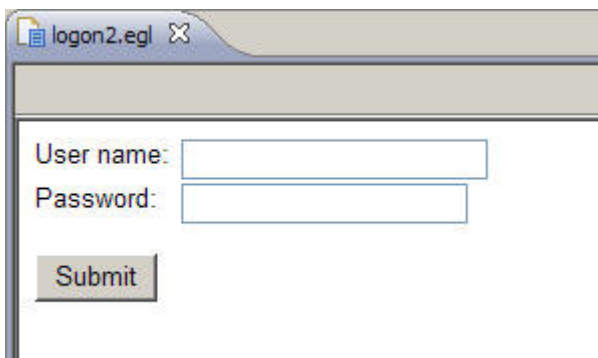
Be aware of your selection border. Your selection determines whether you change the format of a single element on the page or all elements within a container.

The following demonstration shows the steps in this task:

Show Me

To format content by using properties:

1. Open logon2.egl in the EGL Rich UI editor if it is not already open.
2. Select the uidLabel widget ("User name:"), then specify an easily recognizable typeface such as monospace for the **font** property. To see what fonts are available on your system, open a word processor such as Lotus Symphony and look at the Font menu. The EGL Rich UI editor accepts font family designations like "serif", "sans-serif", and "monospace" as well as specific font names like "Courier New." Before you get too creative, remember that if the user's system does not have the specified font installed, the browser will use its default, which is typically Times New Roman.
3. Select the outerBox widget and specify sans-serif for the **font** property. The font for the pwdLabel widget changes to the default sans serif font for the browser, which is typically Arial. The font for the uidLabel widget does not change because, as you would expect, more specific formatting overrides more general formatting.
4. Select the uidLabel widget and clear the font name. The label changes to the default sans serif font for the browser.
5. Select the outerBox widget and specify a **fontSize** of 10 pt. The **fontSize** property defaults to pixels (px), so to use point sizes, you must specify that unit of measure (pt). You can also specify relative sizes, as you can in CSS, from xx-large to xx-small. The font in the labels now matches the font on the **Submit** button.
6. Select the uidLabel widget, expand the **Spacing** group, and set the **paddingRight** field to 6. There is now a reasonable space between the labels and the input fields.
7. Select the pwdLabel widget, expand the **Spacing** group, and set the **paddingBottom** field to 20. This moves the **Submit** button to a more comfortable distance from the labels.
8. Save the file and click **Preview**. The completed page layout is easy to read and use:



Results

You have finalized the layout for the logon page. In the next lesson, you will learn how to bind a function to the **Submit** button that validates the name and password you enter. But first, here is a brief introduction to using style sheets with Rich UI.

Use CSS to format content

About this task

This tutorial is not intended as a course in cascading style sheets (CSS). However, one example should give you an idea of how to use CSS with the EGL Rich UI editor.

The following demonstration shows the steps in this task:

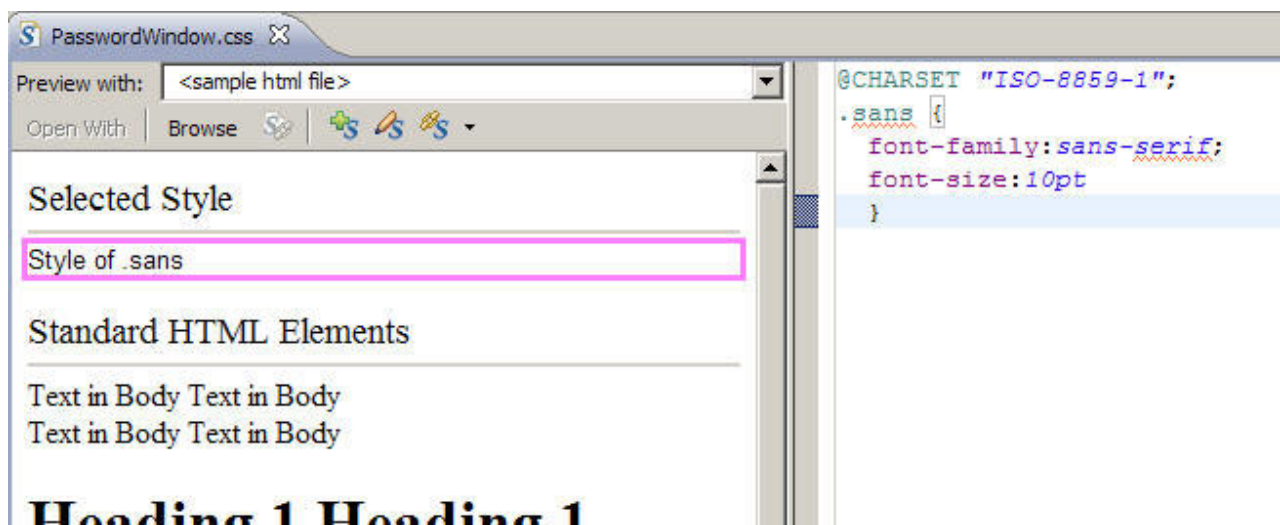
Show Me

To format content by using CSS:

1. In your PasswordWindow project, open the file WebContent/css/PasswordWindow.css.
2. In the right pane of the CSS editor, add the following lines to the end of the file:

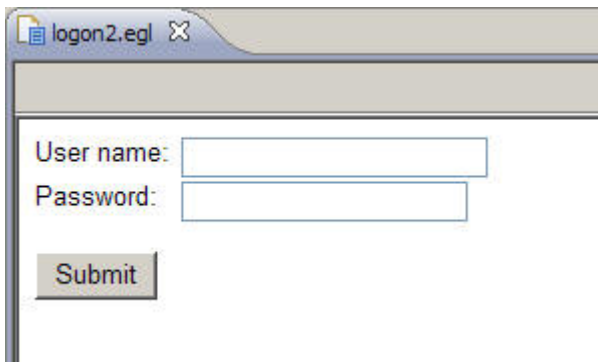
```
.sans {  
    font-family:sans-serif;  
    font-size:10pt  
}
```

This defines a class named "sans" that specifies the same font characteristics that you had previously specified by using properties. Save the file; the left pane should now show the new style:



3. Close PasswordWindow.css and open logon2.egl again.
4. Select the outerBox widget and clear the font and fontSize properties. Both labels revert to the default font (typically Times New Roman) and size.
5. For the **class** property, specify sans.

6. Save the file and click **Preview**. The labels display in 10-point Arial again.



Lesson checkpoint

You learned several different ways to format EGL Rich UI widgets.

Working with boxes, you changed the following properties:

- Background color
- Borders

Working with elements within a container, you changed the following properties:

- Font style and size
- Padding

You also saw how you can use cascading style sheets instead of properties to format content.

Lesson 7: Work with EGL source code

In this lesson you will bind a simple function, written in EGL source code, to the Submit button.

About this task

The EGL Rich UI Handler is entirely composed of EGL code. The Rich UI editor provides shortcuts that you can use to create code in the program, but you can easily write the equivalent code yourself. In this lesson, you will learn how to create a text field widget and write a function in the Source mode of the editor. Then you will bind the function to the **Submit** button to test it.

Create a text field in source code

About this task

The EGL Rich UI widgets that you see on a Web page are specific examples of general widget types. Those types are found in the `com.ibm.egl.rui` project that was automatically placed in your workspace when you created the Rich UI project. To create a new example of one of these types, write a statement with the following syntax:

```
msgField com.ibm.egl.rui.widgets.TextField;
```

`msgField` is the name of the field that you are creating, and `com.ibm.egl.rui.widgets.TextField` specifies the model that the new field is based

on. When you drag a **TextField** widget from the palette and assign a name to it, EGL writes code that is similar to this declaration.

You can shorten a type name by using an EGL **import** statement. The **import** statement tells EGL to look in the specified location to resolve unqualified references. For example, you can add the following statement to the beginning of the source file, after the **package** statement, but before the **handler** declaration:

```
import com.ibm.egl.rui.widgets.*;
```

With this statement in place, you can use the types without qualifying them:

```
uidField TextField{};
```

If you have the **import** statement in your code, EGL also omits the qualifiers for the widget declarations that it generates.

1. Open `logon2.egl` in the Rich UI text editor, if it is not already open.
2. Click **Source**. Note the various sections of the code:
 - The first line identifies the package that contains the program.
 - The first line of the handler declaration specifies a Rich UI Handler and lists the properties of the handler, in braces.
 - The next lines list the variables for the program and their properties, in braces.
 - After the list of variables there is a list of function declarations. Unlike a main EGL program, a handler does not require a `main()` function.
3. On a new line at the end of the list of variables, declare a new variable:
`msgField com.ibm.egl.rui.widgets.TextField{ color = "red" };`

The following screen shot shows that section of code:

```
submitButton com.ibm.egl.rui.widgets.Button{ text = "Submit", onClick ::= logon };
msgField com.ibm.egl.rui.widgets.TextField{ color = "red" };

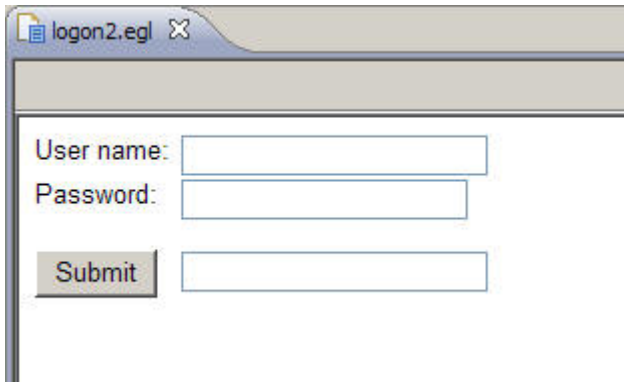
function initialization()
end
```

Although you have created the variable, you have not placed it on the screen. To do that, you must list the variable as a child of an existing container.

4. Locate the `outerBox` declaration in the source code and find the **children** property. Add `msgField` to the end of the list, after `submitButton`. Be sure to separate `msgField` from `submitButton` with a comma. The following screen shot shows that section of code:

```
handler logon2 type RUIhandler {initialUI = [ outerBox ],onConstructionFunction = initial:
  outerBox com.ibm.egl.rui.widgets.Box{ padding=8,
  children = [ uidLabel, uidField, pwdLabel, pwdField, submitButton, msgField ],
  columns = 2,
  class = "sans" };
  uidLabel com.ibm.egl.rui.widgets.TextLabel{ text = "User name:",
```

5. Save the file.
6. Switch to Preview mode. You can now see the new text field.



Bind a simple function to the Submit button

About this task

Functions that *bind* to elements of the interface respond to events that take place in that interface. EGL uses standard terms for these events, such as "onClick" or "onKeyDown", all of which are specific examples of the EGL Event type. When you create a function to handle one of these events, you must include the event as a parameter for the function:

```
function logon(e Event in)
```

In this example, *e* is an arbitrary name for an Event type variable, and *in* indicates that the function reads the parameter for input only.

The **onClick** property is actually an array; you can use it to call multiple functions, in order. You will only call one function, but you must still use the proper syntax for an array:

```
onClick ::= functionName
```

This adds `functionName()` to the end of the list of arrays to call. In a Rich UI Handler, you can add to this array, but you cannot replace it. The following code causes EGL to throw an exception:

```
onClick = [functionName] // does not work in RUIHandler
```

In this task, you will use the EGL Rich UI editor to automatically create a stub function, which contains no executable code, and then bind that function to a widget.

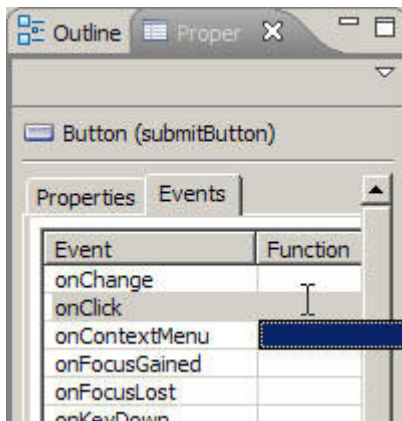
The following demonstration shows the steps in this task:

Show Me

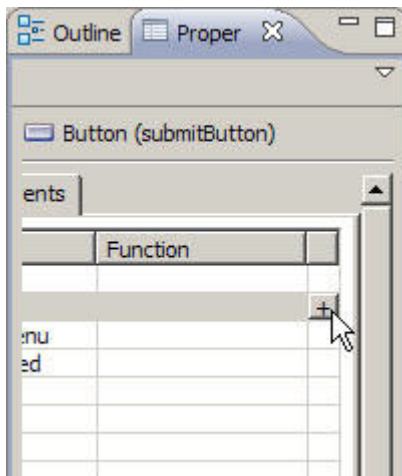
To bind a function to a widget:

1. With the `logon2.egl` file still open in the Rich UI text editor, switch to **Design** mode.
2. Click the **Submit** button to select it. The button is surrounded by a dotted line.
3. In the **Properties** view, click the **Events** tab, which is next to the **Properties** tab. Locate the **onClick** event in the list. Double-click the corresponding table cell in the **Function** column to release the dropdown list. The list shows all of the available event-handling functions. In this case the list is blank because you

have not created any functions to handle events.



4. Locate the plus sign in the far right column for the **onClick** event. You might have to scroll to the right to see it. Click the plus sign to add a stub function.



5. In the New Event Handler window, type the following name for the new function:
logon

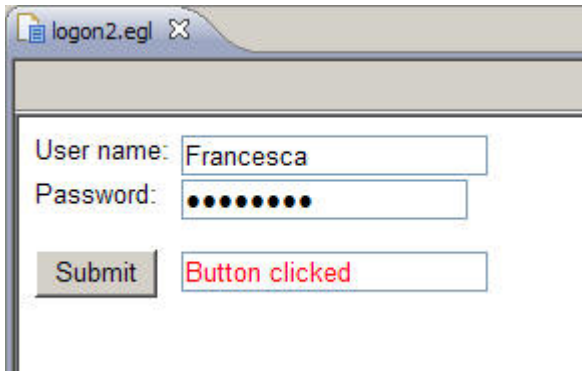


6. Click **OK**. The Rich UI editor switches automatically to **Source** mode and positions the cursor in the stub `logon()` function.
7. Type the following text to complete the function:
`msgField.text = "Button clicked";`
Note that the editor automatically added the **onClick** property to `submitButton`:

```
submitButton com.ibm.egl.rui.widgets.Button{ text = "Submit", onClick ::= logon };
```

On the **Events** page of the Properties view, logon is now available in the drop-down function list.

8. Save the file.
9. Switch to **Preview** mode. Type a name and password; note that the password is displayed as bullet characters. Click **Submit**. The new text field displays the notification message:



Lesson checkpoint

You saw how changes in the Rich UI editor are reflected in the EGL code, and you bound a simple function to the Submit button.

You learned how to perform the following tasks:

- How to create a widget in the EGL code
- How to write a function to handle an event
- How to bind the function to a widget

Summary

This is the end of the *Format a Rich UI logon page* tutorial.

Beyond giving you some basic practice at creating and formatting widgets in the EGL Rich UI editor, this tutorial was intended to demonstrate the following principles:

- Designing your work on paper is a useful preparation for coding the pages.
- There may be multiple design solutions that are equally valid. However, using columns in preference to boxes can reduce application overhead.
- While the Rich UI editor can simplify your coding, understanding the underlying EGL gives you additional programming options.

Lessons learned

- Design a Web page for Rich UI
- Create an organization chart for the elements on the page
- Understand widgets
- Understand the Rich UI Handler
- Create an EGL Rich UI project
- Create a Rich UI Handler
- Structure a page by using box widgets

- Structure a page by using columns only
- Format the elements on the page by using EGL properties
- Format the elements on the page by using cascading style sheets (CSS)
- Add widgets to the page by writing EGL code rather than by using the graphical interface
- Trigger a function by pressing a button

You can continue learning by working with the tutorial application. Try adding more complex functionality to the **Submit** button. For example, if you have taken the *Introducing EGL* tutorial, you could combine that database with the tutorial application and validate user IDs based on the information in the database.

Additional resources

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

Other EGL tutorials:

Introducing EGL

Create a hello world program with EGL

Create a hello world service with EGL

Build a JSF search page with EGL

Access relational databases with EGL