



Tutorial: Build a JSF search page with EGL

Version 8.0.1.2



Tutorial: Build a JSF search page with EGL

Version 8.0.1.2

Note

Before using this information and the product it supports, read the information in "Notices," on page 41.

This edition applies to version 8.0.1.2 of Rational Business Developer and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2000, 2011.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Build a JSF search page with EGL . . . 1

Introduction	1
Lesson 1: Create a simple search page	3
Create the web page.	4
Create the records	4
Add the records to the page	4
Lesson checkpoint	7
Lesson 2: Add code for the search function	7
Create a library for the search functions	7
Add a SQL search function to the library	8
Use the search function in the JSF Handler	9
Bind the search function to the web page	10
Change the CSS file.	11
Lesson checkpoint	12
Lesson 3: Use the OR search condition	12
Add the OR search code to the library	13
Add a radio button group to the page	13
Add the OR search code to the page	14
Lesson checkpoint	16
Lesson 4: Populate a combo box dynamically	16
Add the code to the library	16
Add code to the JSF handler.	17
Add the combo box to the page	18
Lesson checkpoint	20

Lesson 5: Customize the search results	20
Add code to the library	21
Add code to the page code file	22
Create the customized data table	23
Lesson checkpoint	23
Lesson 6: Use type-ahead to prompt the user	24
Lesson checkpoint	30
Summary	30
Resources	30
Completed SearchLibrary.egl file after lesson 2	31
Completed customersearch.egl file after lesson 2	31
Completed SearchLibrary.egl file after lesson 3	32
Completed customersearch.egl file after lesson 3	33
Completed SearchLibrary.egl file after lesson 4	33
Completed customersearch.egl file after lesson 4	34
Completed SearchLibrary.egl file after lesson 5	35
Completed customersearch.egl file after lesson 5	36
Completed customersearchAJAX.egl file after lesson 6.	38

Appendix. Notices 41

Trademarks	43
----------------------	----

Build a JSF search page with EGL

This tutorial expands on the *Introducing EGL* tutorial by teaching you more advanced uses of EGL and JSF. In this tutorial, you create two pages that allow a user to search a database in different ways.

Learning objectives

In this tutorial, you learn how to do these tasks:

- Use SQL statements to filter results for a search page
- Create a customized EGL record part and display it on a page
- Populate a JSF combo box with dynamic data
- Apply type-ahead support to an input control

Time required

90 minutes

Introduction

This tutorial expands on the *Introducing EGL* tutorial by teaching you more advanced uses of EGL and JSF. In this tutorial, you create two pages that allow a user to search a database in different ways.

These pages accept input from the user, search the database for records that match the input, and display the results on the same page. This is not the only way or the best way to create a search page, but this way illustrates several important EGL and JSF concepts.

The search pages you create in this module are very different from web search engines or pages that search the Internet or a single web site. The search pages you create in this module search for database records, not for web pages or information on web pages.

Learning objectives

In this tutorial, you learn how to do these tasks:

- Use SQL statements to filter results for a search page
- Create a customized EGL record part and display it on a page
- Populate a JSF combo box with dynamic data
- Apply type-ahead support to an input control

Time required

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

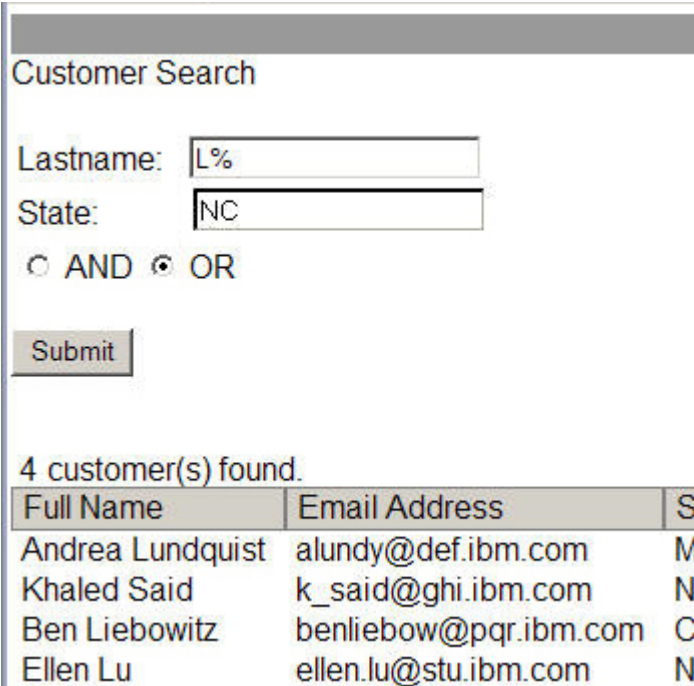
Prerequisites

Before you start this tutorial, you must complete the Introducing EGL tutorial. The current tutorial uses the database connection and the pages you set up in that earlier tutorial.

Tutorial application


When you are finished with the tutorial, you will have a search page that can take input from users, compare it against the data in a database, and return results to the user. You will learn how to create a search that uses two parameters simultaneously (an AND search); you will also learn to change that search to one that uses one or the other of the two parameters (an OR search) and you will place a radio button group on the page to allow the user to choose between types. You will also learn how to limit the user's choice of search parameters by placing a combo box on the page; this presents a list of selections to the user instead of allowing them to type in a string. Finally, you will learn how to customize the search results, combining fields and determining how the output appears on the page.

The first completed search page will look like this:




Full Name	Email Address	S
Andrea Lundquist	alundy@def.ibm.com	M
Khaled Said	k_said@ghi.ibm.com	N
Ben Liebowitz	benliebow@pqr.ibm.com	C
Ellen Lu	ellen.lu@stu.ibm.com	N

The second search page you will create demonstrates some simple uses of AJAX functionality with EGL. One common use of AJAX is to provide suggestions for user input as *type-ahead* support. In this way, the page searches the database for items similar to what the user has already typed in an input field:



Customer search with type-ahead

LastNameInput: 

Once the user has accepted one of the suggestions, the page will use another AJAX request to display the database information without reloading the page:



LastNameInput:

Customerid: 1
Firstname: Francisco
Lastname: Ramirez
Phone: (201)652-3456

Lesson 1: Create a simple search page

In this lesson, you will set up a simple search page. In the next lesson, you will add EGL code to make the search page work.

The basic steps for building this page are essentially the same as any other EGL-JSF web page:

1. Create the web page.
2. Create the EGL variables.
3. Bind the EGL data to JSF components.
4. Add EGL functions to manage the EGL variables.

This lesson covers the first three steps above, and the next lesson covers the fourth step. All lessons build on the files and knowledge from the Introducing EGL tutorial.

Create the web page

1. In the Project Explorer view, right-click the **WebContent** folder of the EGLWeb project, then click **New > Web page**.
2. In the **File Name** field, type this text as the name of the new file, including the extension:
`customersearch.jsp`
3. Make sure that the **Folder** field shows the `/EGLWeb/WebContent` folder.
4. In the **Template** list, click **My Templates**.
5. In the **Preview** box, click the **A_gray.html** template.
6. Click **Finish**. The new page is created and opens in the editor.
7. In place of the default text, type Customer Search.
8. Press **Enter** three times to insert three blank lines.

Create the records

In these steps, you create two EGL records. The `searchTerms` record represents the search input, or the terms of the search. In this case, the `searchTerms` record holds the name and state of the customers being searched for. The `searchResults[]` array of records represents the search results, or the records from the database that match the search input.

1. From the EGL drawer of the Palette view, drag a **New Variable** onto the page, underneath the text Customer Search. The Create a New EGL Data Variable window opens.
2. Under **Type Selection**, click **Record**.
3. Under **Record Type**, click **Customer**.
4. In the **Enter the name of the field** field, type the following text:
`searchTerms`
5. Clear the **Array** check box.
6. Clear the **Add controls to display the EGL element on the web page** check box.
7. Click **OK**.
8. Drag another **New Variable** from the Palette view onto the page, underneath Customer Search. The Create a New EGL Data Variable window opens again.
9. Under **Type Selection**, click **Record**.
10. Under **Record Type**, click **Customer**.
11. Under **Enter the name of the field**, type this name for the field:
`searchResults`
12. Select the **Array** check box.
13. Clear the **Add controls to display the EGL element on the web page** check box.
14. Click **OK**.

Now these two new variables are shown in the Page Data view.

Add the records to the page

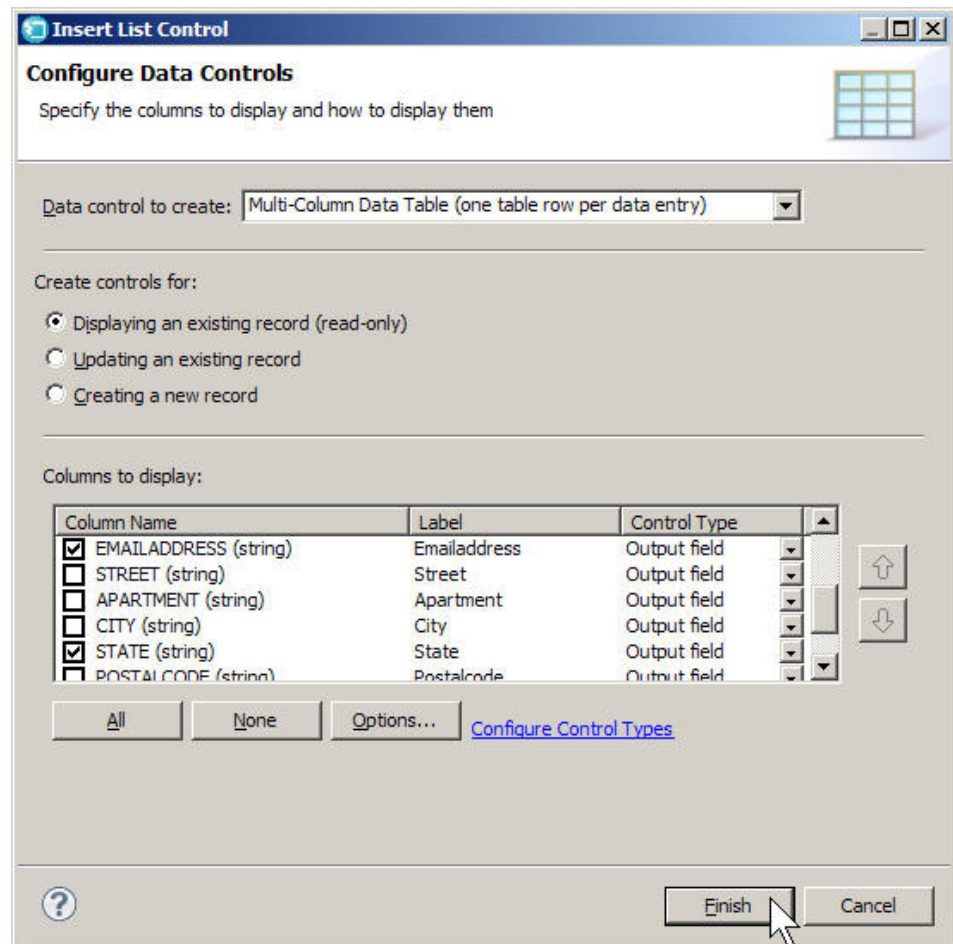
1. In the Page Data view, expand **JSF Handler** and **Data**.
2. From the Page Data view, drag the **searchTerms - Customer** record onto the page, below the Customer Search text. The Insert Control window opens.

3. In the Insert Control window, under **Create controls for**, click **Updating an existing record**.
4. Under **Fields to display**, click the **None** button under the list of fields. This clears all the check boxes.
5. Select the check boxes next to the **LastName** and **State** fields.
6. Click the **Options** button.
7. In the Options window, select the **Submit button** check box.
8. In the **Submit button Label** field, type Submit.
9. Clear the **Delete button** check box.
10. Click **OK**.
11. Click **Finish**. A form appears on the page, with fields for the customer's last name and state. You'll type your search terms into these input fields to search for a customer. The page looks like this:

The screenshot shows a web browser window with the title '*customersearch.jsp'. The address bar shows 'customersearch.jsp - A_gray *'. The main content area displays a form titled 'Customer Search'. The form contains two input fields: 'LastName' and 'State'. Both fields have placeholder text '{LastName}' and '{State}' respectively. Below these fields is a 'Submit' button and a red error message icon followed by '{Error Messages}'. The browser window has a standard Windows-style title bar and a scrollbar on the right.

12. Add a blank line below the Submit button by placing the cursor to the right of the {Error Messages} field for the Submit button and pressing **Enter**.
13. From the Palette, open the **Enhanced Faces Components** drawer.
14. Drag two **Output** components from the **Enhanced Faces Components** drawer onto the page, on the new line below the **Submit** button. These output fields will display the number of results returned and a message, such as 5 Customer(s) found. Search again?
15. Add another blank line after the output fields. From the Page Data view, drag **searchResults - Customer[]** onto the new blank line, below the **Submit** button and the two new output fields. The Insert List Control window opens.
16. In the Insert List Control window, click the radio button next to **Displaying an existing record (read-only)**.

17. Under **Columns to display**, click the **None** button under the list. This clears the check boxes.
18. Select the check boxes next to the **LastName**, **EmailAddress**, and **State** fields. The Insert List Control window looks like this:



19. Click **Finish**.
20. Save the page.

Now the search page has the input fields for the user to type the search terms, as well as a data table to display the search results. The search page looks like this:

The screenshot shows a web page with a search form at the top. The form includes two input fields: 'Lastname' with a placeholder '{LASTNAME}' and 'State' with a placeholder '{STATE}'. Below these is a 'Submit' button and a '{Error Messages}' label. There are also two 'outputText' labels. Below the form is a table with three columns: 'Lastname', 'Emailaddress', and 'State'. Each column has a header and a body containing a placeholder like '{LASTNAME}', '{EMAILADDRESS}', and '{STATE}' respectively.

Lesson checkpoint

You have now created a simple search page.

In this lesson, you have learned how to do the following:

- Create an EGL web page
- Create EGL records
- Add the EGL records to the web page

Now you are ready to begin Lesson 2: Add code for the search function.

Lesson 2: Add code for the search function

Now that you have created the web page, you must add the EGL code that receives the search terms from the web page, searches the database according to those terms, and displays the search results on the page.

Create a library for the search functions

Because these search functions can get complicated, you will want to create a library to hold them. Then, you can reuse those search functions and keep your JSF Handlers simple.

1. Right-click the **EGLSource** folder of your **EGLWeb** project and then click **New** > **Library**. The New EGL Library window opens.
2. In the **EGL source file name** field, type this name for the new library:
SearchLibrary
3. In the **Package** field, type `libraries`. EGL will create this new package if you don't have one by this name.
4. Under **EGL Library Type**, click **Basic**.
5. Click **Finish**. The new library is created and opens in the EGL editor.

6. Remove all the filler text from the new library so all that is left is the following code:

```
package libraries;

library SearchLibrary type BasicLibrary

end
```

Now you can add functions to this library and use them in your JSF Handlers.

Add a SQL search function to the library

1. At the bottom of the SearchLibrary file but before the final End statement, insert this code:

```
function NameAndStateSearch_And(lname STRING in,
    state CHAR(2) in, customer Customer[])
    get customer;
end
```

This function is similar to the functions in other libraries that retrieve every record from the database. The difference is that this function receives these three parameters:

- The string variable `lname`, which represents the last name of the customer being searched for.
- The character variable `state`, which represents the state of the customer being searched for.
- The array of customer records `customer`, which will hold the results of the search.

Right now, this function will retrieve every record in the database. In the next few steps, you edit the SQL statement generated by this function so that only the records matching the search terms `lname` and `state` are returned.

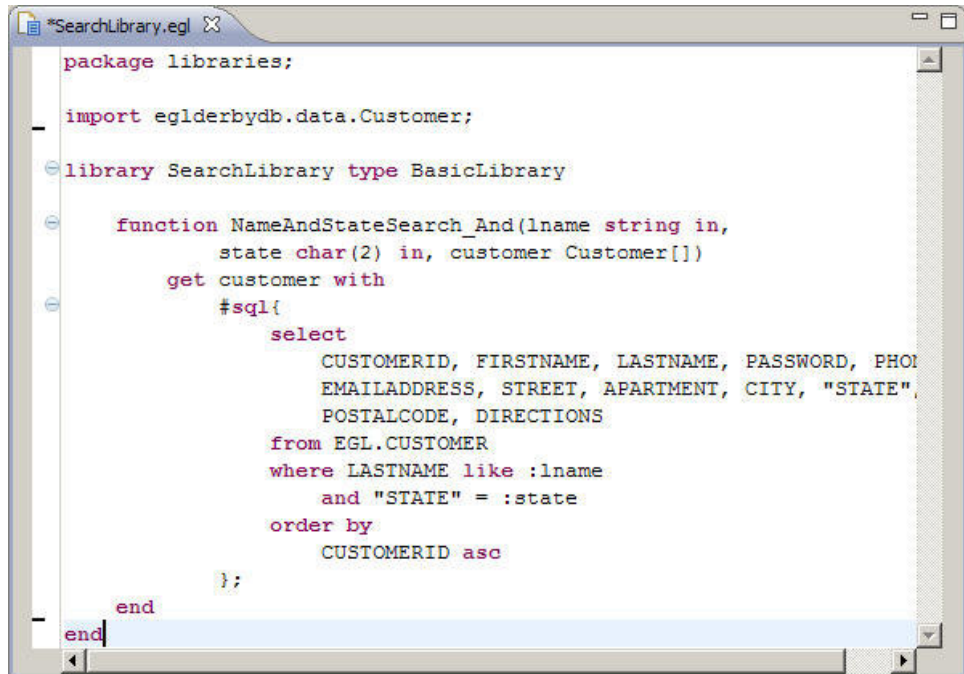
2. If you did not use code completion to insert this function, press **Ctrl+Shift+O** to organize your imports.
3. Right-click the word `customer` in the line `get customer` and click **SQL Statement > Add** from the popup menu. The explicit SQL statement is added to the `get customer` line of code.

Strictly speaking, nothing has changed in your code. EGL has merely exposed the default SQL code that it creates when it encounters the code `get customer`. Now that this SQL code is explicitly shown on the page, you can edit it to make it behave differently. In this case, you want to change the statement from retrieving every customer record to retrieving only the customer records with a matching last name and state.

4. Add a blank line after the line from `EGL.CUSTOMER` by placing the cursor at the end of the line and pressing **Enter**.
5. In the new blank line below from `EGL.CUSTOMER`, insert this code:

```
where LASTNAME like :lname
and "STATE" = :state
```

The code looks like this:



```
package libraries;

import eglderbydb.data.Customer;

library SearchLibrary type BasicLibrary

function NameAndStateSearch_And(lname string in,
    state char(2) in, customer Customer[])
get customer with
    #sql{
        select
            CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
            EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
            POSTALCODE, DIRECTIONS
        from EGL.CUSTOMER
        where LASTNAME like :lname
            and "STATE" = :state
        order by
            CUSTOMERID asc
    };
end
end
```

The code you've added is not EGL but SQL. LASTNAME is the complete name of a field in the sample database your project is using. If you look at the Record parts in the package eglderbydb.data, you will see that the records such as the Customer record also refer to these fields. The code :lname and :state are called *host variables*, which in this context are EGL variables that you use in SQL code. The STATE is enclosed in quotes to indicate that "state" is the name of the table, not the SQL reserved word.

EGL provides different ways to create and generate SQL statements. In an earlier EGL tutorial, you retrieved a specific database record by specifying a particular customer ID number. This clause created a SQL where statement similar to the where statement you just added. You could also use a defaultSelectCondition to perform the same task.

6. Save the file. EGL generates the file automatically.

Here is the complete code of the library file. If you see any errors marked by red X symbols in the editor, make sure your code matches the code in this file: "Completed SearchLibrary.egl file after lesson 2" on page 31

Use the search function in the JSF Handler

1. Open the customersearch.jsp page.
2. Right click on the content area of the page in the editor and click **Edit Page Code** from the popup menu. The page code file opens.
3. In the JSF handler for the customersearch.jsp page, find the variables you created to represent the search terms and search result:

```
searchTerms Customer;
searchResults Customer[0];
```
4. Immediately after your previously created variables, add the following code to create two additional variables:

```
resultMessage CHAR(80);
numberOfResults INT;
```

Next, you need to create a function to be called from the web page. This function will pass the `searchResults` variable and the necessary fields from the `searchTerms` variable to the function in the library.

5. Add the following function to the JSF Handler just before the final **end** statement:

```
function searchFunction()  
    searchTerms.LastName = searchTerms.LastName::"%";  
    SearchLibrary.NameAndStateSearch_And(  
        searchTerms.LastName,  
        searchTerms.State, searchResults);  
    resultMessage = " customer(s) found.";   
    numberOfResults = searchResults.getSize();  
end
```

Ignore any red Xs for now.

- 6.
7. Add code to the `onPreRender` function to reset the page after a failed search:

```
function onPrerender()  
    if (searchResults.getSize() == 0)  
        resultMessage = "No customers found or no search criteria entered.";   
    end  
end
```

Here is some information about the page code you just added:

- The record `searchTerms` and the array of records `searchResults` are both instances of the Customer record. You can create multiple instances of a record or `DataItem` part.
- This code includes the function `searchFunction`, which will be bound to the page's Submit button. This function calls the `NameAndStateSearch_And` function you added to the library earlier in this lesson.
- The search function adds a wildcard character to the end of the last name that the user enters. For example, if the user enters `Sm`, the search string becomes `Sm%` and returns results like `Smith` and `Smiley`.
- The search function is case sensitive.

8. Organize imports (Ctrl+Shift+O) and save the file.

Here is the complete code of the `customersearch.egl` file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed `customersearch.egl` file after lesson 2" on page 31

Bind the search function to the web page

Now that you have set up the data and function in the JSF handler, you can use them on the page.

1. Open the `customersearch.jsp` page.
2. In the Page Data view, expand **JSF Handler**.
3. From the **Data** group, drag the `numberOfResults` variable directly onto the left output text field that is below the Submit button.
4. From the **Data** group, drag the `resultMessage` variable directly onto the right output text field.
5. From the **Actions** group, drag `searchFunction()` directly onto the **Submit** button on the page. The appearance of the page does not change, but the function is now bound to the button.
6. Save the page.

7. Test the page by running it on the server and entering search terms into it:
 - a. In the Project Explorer view, right-click **customersearch.jsp** and then click **Run > Run on Server**. In previous tutorials, you should have set up a default server for the project. If you did not do this or have changed the settings, you may have to select which server to use again.
 - b. When the page opens in the web browser, enter a letter in the LastName field and a state in the State field and then press the **Submit** button. Note that this search page is case sensitive.

This search page is difficult to use because the user must know both the customer's state and the first letter of the customer's last name. It would be better if the user were able to choose between an AND search and an OR search. In the next lesson, you will add this option to the page. In a later lesson, you will change the State input field to a combo box that lists all of the valid states used in the database.

In addition, there is a problem with "customer(s) found" display; on output, the space before "customer(s)" was lost. In the next exercise, you will change the cascading style sheet (CSS) for the page to fix this problem.

Change the CSS file

Cascading style sheets work by associating elements in an HTML page with a set of styles that determine how those elements are displayed. EGL provides two style sheets for the page template you selected:

stylesheet.css

A general style sheet for all the templates

gray.css

A style sheet specific to the **A_gray.html** template.

In this exercise you will change the more general template.

1. Without closing the search results page in the browser, expand **WebContent** and **theme**. Double-click **stylesheet.css** to open it in the editor.
2. In the right pane of the editor, locate the **.outputText** element. Add the following line between the braces:

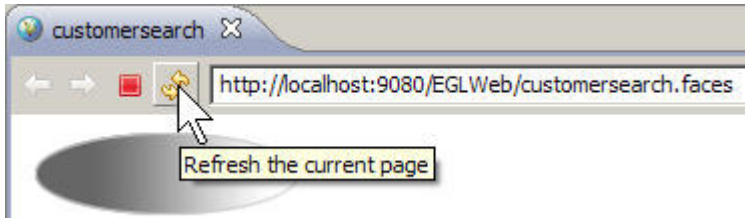
```
padding-right:5px;
```

This requires a browser to add 5 pixels of space to the right of any text tagged as `<h:outputText>` when rendering it for display. The element now looks like the following example; note the colors of the fonts:

```
.inputHidden {  
}  
  
.outputText {  
    padding-right:5px;  
}  
  
.outputFormat {  
}
```

3. Save and close the CSS file.

- Go back to the search results page (**customersearch**) in the browser window.
Click the page refresh icon next to the address:
The refreshed page now shows a space between the number and the text in the



"customer(s) found" message.

Lesson checkpoint

You have created the EGL code that will power the search page you created in the last lesson.

In this lesson, you have learned how to do the following:

- Create an EGL library to contain functions
- Edit and add code to the EGL JSF Handler to call functions in the library
- Bind the functions in the JSF Handler to the controls on the web page

Now you are ready to begin Lesson 3: Use the OR search condition.

Lesson 3: Use the OR search condition

In this lesson, you add a radio button group to the page that allows the user to choose between an AND search condition and an OR search condition.

When you run this improved page, it will look like this:

Full Name	Email Address	
Andrea Lundquist	alundy@def.ibm.com	M
Khaled Said	k_said@ghi.ibm.com	N
Ben Liebowitz	benliebow@pqr.ibm.com	C
Ellen Lu	ellen.lu@stu.ibm.com	N

Add the OR search code to the library

In the previous lesson, you added a function that searches with the AND condition. In the following steps, you add a function that searches with the OR condition. In this way, the user will be able to search for records that match either a last name or a state.

1. Open the SearchLibrary.egl library file.
2. Add the following code to the file, just before the final end statement:

```
function NameAndStateSearch_Or(lname STRING in,
    state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname
or "STATE" = :state
order by
CUSTOMERID asc
};
end
```

This function is identical to the NameAndStateSearch_And function you added in the previous lesson, except that it uses OR instead of AND in the where statement.

3. Save the file. EGL generates the library automatically.
4. Close the file.

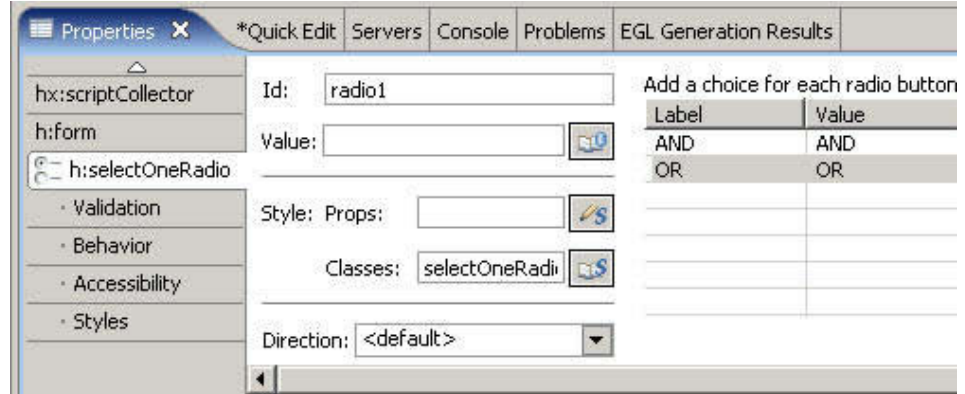
Here is the complete code of the SearchLibrary.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed SearchLibrary.egl file after lesson 3" on page 32.

Add a radio button group to the page

Now that you have two different search functions, you need to add a radio button to the page so the user can choose which type of search to use.

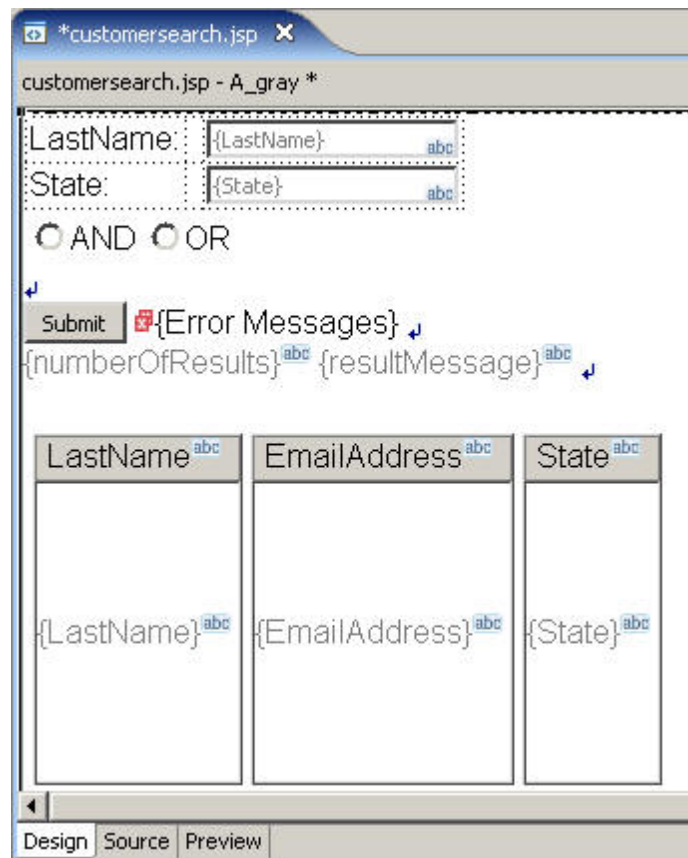
1. Return to the customersearch.jsp file.
2. Add a new line above the Submit button by placing the cursor to the left of the Submit button and pressing **Enter**.
3. From the Palette, open the **Enhanced Faces Components** drawer.
4. Drag a **Radio Button Group** onto the new line.
5. Click the radio button group to select it.
6. If you do not have the Properties view open, open it by clicking **Window > Show View > Properties**.
7. In the Properties view, click the **Add Choice** button. The **Add Choice** button is on the far right of the Properties view. A new choice for the radio button group is listed in the table to the left of the buttons.
8. In the **Label** field of the new choice, type this text:
AND
9. In the **Value** field of this choice, type this text:
AND
10. Click **Add Choice** again.

11. Type this text for the **Label** and **Value** of the second choice:
OR The properties view looks like this when you have finished:



12. Save the page.

The page looks like this when you finish adding the radio button group:



Add the OR search code to the page

Next, you must configure the JSF handler to use the input from the radio button to decide which search function to use.

1. Right-click a blank area of the page and click **Edit Page Code** from the menu.
The file customersearch.egl opens in the editor.

2. With the variable declarations at the top of the handler, add this line of code:
`andOr CHAR(3);`

Later, you will bind this variable to the radio buttons. It holds the value "AND" or "OR," depending on which radio button you select on the page.

3. Replace the function call to `NameAndStateSearch_And` with the following code:

```
if (andOr == "AND")
    SearchLibrary.NameAndStateSearch_And(
        searchTerms.LastName,
        searchTerms.State, searchResults);
else
    SearchLibrary.NameAndStateSearch_Or(
        searchTerms.LastName,
        searchTerms.State, searchResults);
end
```

The entire function now looks like the following code:

```
function searchFunction()
    searchTerms.LastName = searchTerms.LastName+"%";

    if (andOr == "AND")
        SearchLibrary.NameAndStateSearch_And(
            searchTerms.LastName,
            searchTerms.State, searchResults);
    else
        SearchLibrary.NameAndStateSearch_Or(
            searchTerms.LastName,
            searchTerms.State, searchResults);
    end

    resultMessage = " customer(s) found.";
    numberOfResults = searchResults.getSize();

end
```

This function now calls different functions depending on the value of the `andOr` variable.

4. Save and close the file.
5. Return to the `customersearch.jsp` page.
6. From the Page Data view, bind the **andOr - char(3)** variable to the radio button group by dragging it onto the radio button group on the page.
7. Bind the **searchFunction()** function to the Submit button on the page.
8. Save the page.
9. Test the page.

When you test the page, try using the new radio button functions. You must select one of the radio buttons for the search page to work properly.

This search page is still difficult to use because there are not many records in the sample database and many states to guess from. In the next lesson, you will change the State input field to a combo box that lists all of the states used in the database.

Here is the complete code of the `customersearch.egl` file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed customersearch.egl file after lesson 3" on page 33.

Lesson checkpoint

You now have a search page that can search based on two parameters simultaneously, or can find results that match one, but not the other.

In this lesson, you have learned how to do the following:

- Add an OR search to the search function in your EGL library
- Add a radio button group to your search page
- Add the OR search code to the JSF Handler
- Bind the new search function to the radio button group

Now you are ready to begin Lesson 4: Populate a combo box dynamically.

Lesson 4: Populate a combo box dynamically

In this lesson, you enhance the search page by listing the possible choices for the customer's state in a combo box.

To make searching as easy as possible for the users, you should prevent user error and simplify the decisions the user must make wherever possible. In this lesson, you learn how to make the search page easier to use by replacing the State input field with a combo box. This combo box lists only the states that are represented by at least one customer record in the database, preventing the user from having to guess which state to use.

Add the code to the library

First, you need to add a function to your library that retrieves every state represented in the database. This function is simpler than the other functions in the library because you need to retrieve only one column from the database and there are no input parameters for the search, only an output array holding the list of states. You could use an array of customer records for the state information, but an array of strings is simpler to work with because you don't need the rest of the fields in the record.

1. Open SearchLibrary.egl.
2. Add the following function to the library:

```
function getAllCustomerStates(listOfStates STRING[])
    customers Customer[0];
    counter INT;

    get customers with
    #sql{
        select "STATE"
        from EGL.CUSTOMER
        order by "STATE" asc
    };

    listOfStates.removeAll();
    for (counter from 1 to customers.getSize() by 1)
        listOfStates.appendElement(customers[counter].State);
    end

end
```
3. Save the file.
4. Generate the library.

Here are some technical notes about the `getAllCustomerStates` function you just added:

- This function accesses the customer records from the database in the same way as the `getAllCustomers()` function. The major difference is that the `getAllCustomerStates()` function selects only the STATE fields instead of every field in the Customer table.
- The "group by" SQL command groups the results by state so each state is listed only once in the results.
- The "order by" SQL command puts the results in alphabetical order; the `asc` keyword indicates ascending order.
- The `for` loop moves only the state field from the records into an array of strings.

Here is the complete code of the `SearchLibrary.egl` file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed `SearchLibrary.egl` file after lesson 4" on page 33.

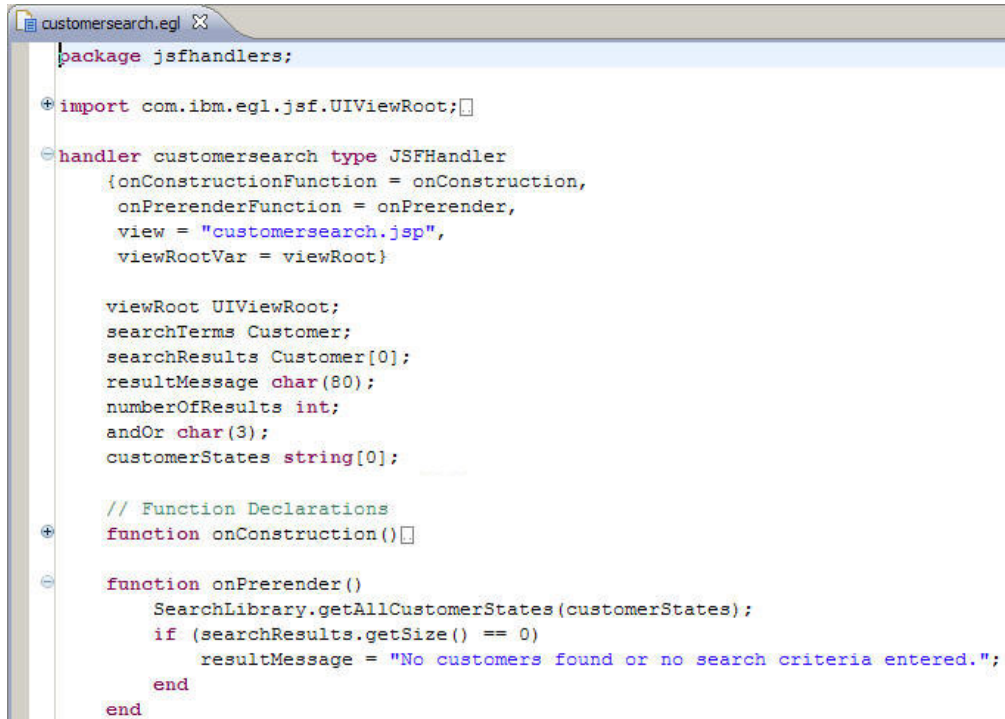
Add code to the JSF handler

1. Return to the `customersearch.jsp` page.
2. Right-click on the `customersearch.jsp` page and click **Edit Page Code** from the popup menu.
3. After the line of code `andOr CHAR(3);`, add this line of code:
`customerStates STRING[0];`

This variable holds the list of states returned by the function in the library.

4. Add a blank line after function `onPreRender()`. Then, add this line of code on the blank line:
`SearchLibrary.getAllCustomerStates(customerStates);`
5. Save the file.

The `customersearch.egl` file looks like this when you are done (some functions are compressed, indicated by a plus sign in the left margin of the page):



```
customersearch.egl
package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
 onPreRenderFunction = onPreRender,
 view = "customersearch.jsp",
 viewRootVar = viewRoot}

viewRoot UIViewRoot;
searchTerms Customer;
searchResults Customer[0];
resultMessage char(80);
numberOfResults int;
andOr char(3);
customerStates string[0];

// Function Declarations
function onConstruction()

function onPreRender()
    SearchLibrary.getAllCustomerStates(customerStates);
    if (searchResults.getSize() == 0)
        resultMessage = "No customers found or no search criteria entered.";
    end
end
```

Here are some technical notes on the code you just added:

- The customerStates array holds the list of states represented by at least one customer in the database.
- The line you added to the onPreRender function sends the customerStates array to the getAllCustomerStates function in the library, populating the array with the list of states.

Here is the complete code of the customersearch.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed customersearch.egl file after lesson 4" on page 34

Add the combo box to the page

Adding a combo box that is populated dynamically is more complicated than adding a JSF control that has predefined values, such as the radio button group you added in the previous lesson. This combo box must be bound to two pieces of EGL data:

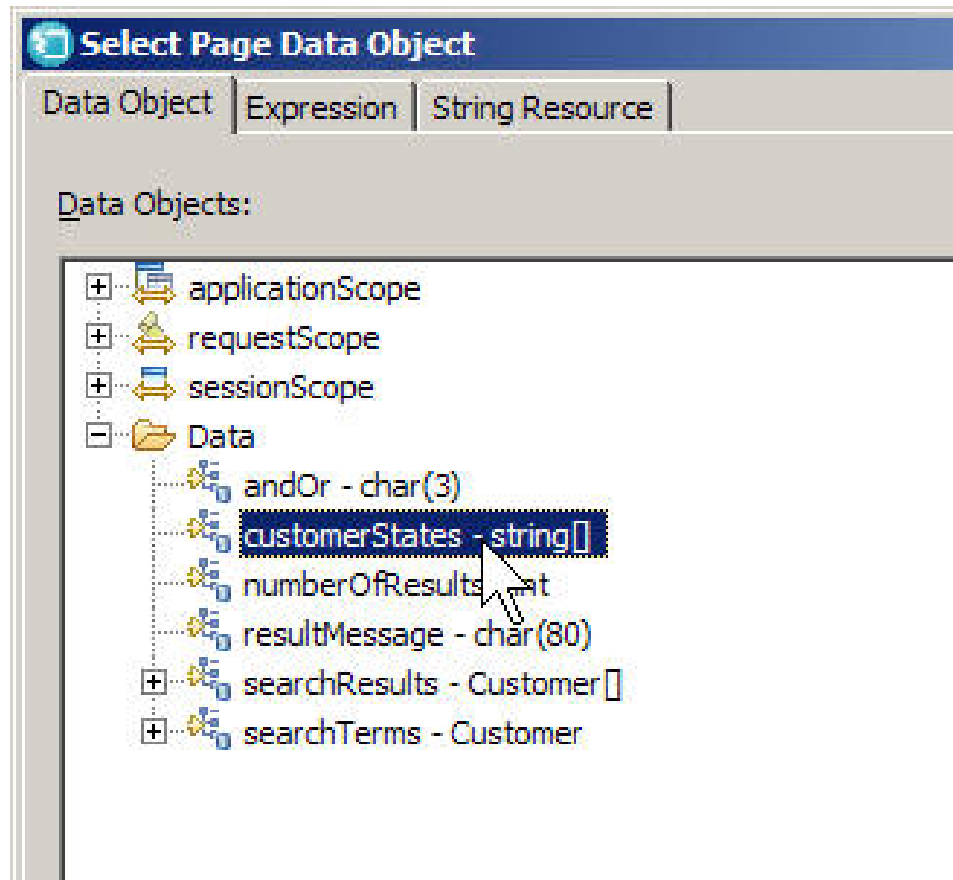
- The customerStates array, which provides the list of options for the combo box.
 - The searchTerms.State variable, which holds the user's selection from the combo box.
1. Return to the customersearch.jsp page.
 2. Click the STATE input field to select it and press **Delete**. The input field is removed from the page.
 3. From the Palette, open the **Enhanced Faces Components** drawer.
 4. Drag a **Combo Box** item to the page, and place it where the STATE input field was.
 5. In the Page Data view, expand **JSF Handler > Data > searchTerms - Customer**.

- Under **searchTerms - Customer** in the Page Data view, drag **State - State** onto the combo box.



- Click the combo box to select it.
- Open the Properties view. In the Properties view, note that the **Value** field is set to `#{customersearch.searchTerms.State}`, indicating that the value of the selection in the combo box is placed into the State field of the searchTerms record.
- In the Properties view, click **Add Set of Choices**, which is at the far right of the view, near the table of choices. A new item is added to the list of choices with the label `<selectitems>` and a default value. The `<selectitems>` label is a JSF tag that represents multiple options, rather than a single static label. In other words, the combo box will use the values you specify in the **Value** column for both the labels in the combo box and the values the labels represent.
- Next to the `<selectitems>` label, click the **Select Page Data Object** button in the **Value** field. The Select Page Data Object window opens.
- In the Select Page Data Object window, expand **Data** and click **customerStates - string[]**.

The Select Page Data Object window looks like this:



12. Click **OK**. Now the choices in the combo box come from the `customerStates` variable, while the selected state in the combo box is put into the `searchTerms` variable.
13. Save the page.
14. Test the page.

Lesson checkpoint

You have created a combo box on your web page that creates a list of search parameters.

In this lesson, you have learned how to do the following things:

- Add the code for a limited search to the library.
- Add code to the JSF Handler to call the revised search function.
- Add a combo box to the web page.
- Bind the revised search function to the combo box.

Now you are ready to begin Lesson 5: Customize the search results.

Lesson 5: Customize the search results

In this lesson, you learn to make a more complex data table to display your search results.

Until now, each JSF component you have added to Web pages has been bound to data from a single database table. If you are using a complex relational database, you may want to work with data from more than one table at a time.

In this lesson, you customize the results by displaying data from both the Customer table and the State table. In this way, the results show both the customer's name (from the Customer table) and the full name of the customer's state instead of the two-letter abbreviation (from the State table). You also manipulate the results by combining the customer's first name and last name into a full name field. The resulting data table looks like this:

4 customer(s) found.

Full Name	Email Address	State
Andrea Lundquist	alundy@def.ibm.com	MICHIGAN
Khaled Said	k_said@ghi.ibm.com	NORTH CAROLINA
Ben Liebowitz	benliebow@pqr.ibm.com	CONNECTICUT
Ellen Lu	ellen.lu@stu.ibm.com	NEW YORK

The easiest way to create a customized data table like this is to create a customized EGL record that represents a single record in this data table. Then, you create an array of these records that is bound to the data table. The customized EGL record you create in this lesson has the following three fields:

- The email field, which holds the customer's email address from the Customer table.
- The fullName field, which holds the customer's combined first and last name from the Customer table.
- The State field, which holds the full name of the customer's state. To get the full name, the search function cross-references the customer's state abbreviation from the Customer table with the list of abbreviations and state names in the State table.

Add code to the library

1. Open SearchLibrary.egl.
2. Add the following function to the library:

```
function getOneState(state Statetable)
  get state;
end
```

The stateTable record is defined in the Statetable.egl file, so if you did not use code completion to enter that function, you must organize your imports (Ctrl+Shift+O).

3. Add the following code to the file, just *after* the final **end** statement in the library:

```
Record customizedResult type basicRecord
  fullName STRING {displayName = "Full Name"};
  email STRING {displayName = "Email Address"};
  stateName STRING {displayName = "State"};
end
```

Note: Because the library itself can not contain record definitions, you must add the customizedResult record definition after the end statement that closes the library.

4. Save the file.
5. Generate the file.

Here is the complete code of the SearchLibrary.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: “Completed SearchLibrary.egl file after lesson 5” on page 35.

Add code to the page code file

1. Return to the customersearch.jsp page.
2. Right-click on the customersearch.jsp page and click **Edit Page Code** from the popup menu.
3. Add the following variable definition to the JSF Handler, after the customerStates STRING[0]; line:
allRecords customizedResult[0];

This variable represents the new search results, based on the customized record you just created.

4. If you did not use code completion to insert this line of code, be sure to organize imports.
5. Add the following function to the JSF Handler:

```
function generateCustomResults(passedResults Customer[])
    allRecords.removeAll();
    oneRecord customizedResult;
    counter INT = 1;
    state Statetable;

    //loop once for each search result returned
    while (counter <= (passedResults.getSize()))
        oneRecord.fullName = passedResults[counter].FirstName ::
            " " :: passedResults[counter].LastName;
        oneRecord.email = passedResults[counter].EmailAddress;
        state.STATE_ABBREV = passedResults[counter].state;
        SearchLibrary.getOneState(state);
        oneRecord.stateName = state.STATE_NAME;
        allRecords.appendElement(oneRecord);
        counter = counter + 1;
    end
end
```

This function assembles the customized search results. You must call this function at the end of the searchFunction() function.

6. Add this line of code immediately before the **end** statement that closes the searchFunction() function in the JSF Handler:
generateCustomResults (searchResults);
7. Save the file.

The new function you added to the JSF handler assembles the customized search results by following these general steps:

1. Assemble the customer's full name from the first and last name.
2. Get the customer's email address.
3. Get the abbreviation of the customer's state.
4. Look up the state name that matches the abbreviation.
5. Add the full name, email address, and state name to the allRecords array.

Here is the complete code of the customersearch.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: “Completed customersearch.egl file after lesson 5” on page 36.

Create the customized data table

1. Return to the customersearch.jsp page.
2. If you want to remove the old search results, you can delete the old data table. These steps are optional:
 - a. Click anywhere in the old search results table to set the cursor focus there.
 - b. Press the **Down** arrow key. The entire data table is now selected.
 - c. Press **Delete**. The data table is removed from the page.
3. From the Page Data view, drag **allRecords - customizedResult[]** onto the page, just below the old data table location. The Insert List Control window opens.
4. Click the radio button next to **Displaying an existing record (read-only)**.
5. Click **Finish**. The new data table is created on the page.
6. Save the page.
7. Test the page.

Now when you search for a customer, you see the customer's full name, email address, and full state name in the data table. The page looks like this:

The screenshot shows a web page titled "Customer Search". It has two input fields: "Lastname:" with the value "L%" and "State:" with the value "NC". Below these fields are two radio buttons: "AND" (unselected) and "OR" (selected). A "Submit" button is located below the radio buttons. Below the search form, it says "4 customer(s) found." followed by a table with three columns: "Full Name", "Email Address", and a third column with single letters. The table contains four rows of data.

Full Name	Email Address	
Andrea Lundquist	alundy@def.ibm.com	M
Khaled Said	k_said@ghi.ibm.com	N
Ben Liebowitz	benliebow@pqr.ibm.com	C
Ellen Lu	ellen.lu@stu.ibm.com	N

Lesson checkpoint

You have customized your search results by combining the first name and last name fields into a full name field, and by translating the two character code from the state field into the full name of the state.

In this lesson, you have learned how to do the following:

- Add a function to the library that customizes results
- Add a record to the library file, outside the library itself

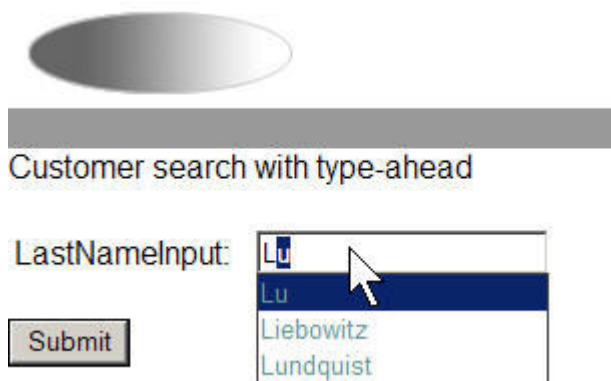
- Add the new customized results function to the JSF Handler
- Place a new results table on the Web page
- Bind the new results function to the table

Lesson 6: Use type-ahead to prompt the user

Controls with the *type-ahead* feature try to anticipate what the user might be typing into an input field, speeding up the process of entering information. In this lesson, you learn to create a separate search page that uses type-ahead functionality to search in a different way than in the previous page.

Type-ahead functionality uses *AJAX*, or Asynchronous Javascript and XML, which is a web technology that allows web applications to change portions of a web page without reloading the entire page. In this case, the AJAX functionality allows your EGL JSF Handler to add and remove suggestions to the input control without resubmitting the page. The EGL type-ahead functionality provides a shortcut to this common use of AJAX functionality, preventing you from having to put together an AJAX request by yourself. In the next lesson, you will learn to create custom AJAX requests.

By convention, input controls with type-ahead provide options based on the first few characters the user types into a field:



Depending on how you write the EGL code for the type-ahead, you can provide suggestions based on the first few characters of input or from an arbitrary function that determines the suggestions dynamically.

To provide suggestions for type-ahead based on the first few characters of input, you must specify those suggestions at design time, either in the **validValues** property or in a Data Table part. This example uses the **validValues** approach:

```
state STRING {typeahead = YES,
    validValues = ["AK","AL","AR","AZ",
        "NC","NY","NH","NJ",
        "NM","NE","NV","ND"]};
```

In this variable, the valid values are the abbreviations of U.S. states beginning with the letters "A" and "N." When the user types the letter "A" into the input control, type-ahead will provide the abbreviations beginning with "A", and likewise with the letter "N" and the abbreviations beginning with "N."

In this lesson, you use type-ahead functionality to suggest the last names of customers in the database for the user to search on. In this case, you must determine the type-ahead suggestions at run time, because the suggestions will depend on the values in the database. Therefore, you will create a function in a new JSF Handler that compares what the user has typed into an input field with the last names of customers in the database. The function then provides the matching names as type-ahead suggestions.

1. Create a new web page in the EGLWeb project named `customersearchAJAX.jsp`:
 - a. In the Project Explorer view, right-click the **WebContent** folder of the EGLWeb project and then click **New > Web page**.
 - b. In the **File Name** field, type this text as the name of the new file, including the extension:
`customersearchAJAX.jsp`
 - c. Make sure that the **Folder** field shows the `/EGLWeb/WebContent` folder.
 - d. In the **Template** list, click **My Templates**.
 - e. In the **Preview** box, click the **A_gray.html** template.
 - f. Click **Finish**. The new page is created and opens in the editor.
 - g. Remove the default text from the page and replace it with the following text:

Customer search with type-ahead

- h. Press Enter three times to create blank space below the page title.
2. Open the JSF Handler for the page by right-clicking on the content area of the page and then clicking **Edit Page Code**.
3. Add these three variables to the JSF Handler:

```
lastNameInput STRING {TypeaheadFunction = suggestLastNames};
allLastNames Customer[0];
customerToDisplay Customer;
```

The first variable will be bound to an input control on the page. The **TypeaheadFunction** property indicates that this variable will have a function that provides type-ahead suggestions. You will create this function later in this lesson.

The second variable will hold the list of last names in the database. The function providing suggestions will need this list.

The third variable is a single record that will show the results of the search.

4. Remove the **onConstructionFunction** property from the JSF Handler, but leave the **onPreRenderFunction**:

```
handler customersearchAJAX type JSFHandler
{onPreRenderFunction = onPreRender,
view = "customersearchAJAX.jsp",
viewRootVar = viewRoot}
```

5. Remove the stub `onConstruction()` function.
6. Complete the stub `onPreRender()` function:

```
function onPreRender()
get allLastNames with
#sql{
select
  LASTNAME
from EGL.CUSTOMER
group by
  EGL.CUSTOMER.LASTNAME
};
end
```

This function will run each time the page is refreshed, in order to retrieve a list of customer last names from the database to compare with the user's input.

7. Add the following function to the JSF Handler:

```
function suggestLastNames(typedCharacters STRING in) returns (STRING[])
matchingLastNames STRING[0];
oneCustomer Customer;
oneCustomerName STRING;

//This variable represents the characters the user has typed.
typedCharacters = StrLib.upperCase(typedCharacters);

//Compare the user input to the values in the database.
for (counter INT from 1 to allLastNames.getSize())
oneCustomer = allLastNames[counter];
oneCustomerName = StrLib.upperCase(oneCustomer.LastName);

if (StrLib.indexOf(oneCustomerName, typedCharacters) == 1)
//This value starts with the same characters.
//Add this value to the type-ahead suggestions.
matchingLastNames.appendElement(oneCustomer.LastName);
end
end
return (matchingLastNames);
end
```

This function is the function referred to in the **TypeaheadFunction** property of the variable you created earlier. As its name implies, this function provides the suggestions for type-ahead. This function must receive a single parameter: a STRING representing the characters that the user has typed into the input control. Also, it must return an array of STRINGS, representing the suggestions. With this function, you can determine the type-ahead suggestions with any arbitrary EGL logic.

In this case, the function follows the convention that the suggestions should start with the same characters as the user has typed into the input control. The function compares the characters that the user has typed in to each customer's last name, in each case, setting both values to upper case to eliminate any differences in capitalization. Each time the function finds a match, it adds the customer's last name to the array of results representing the type-ahead suggestions.

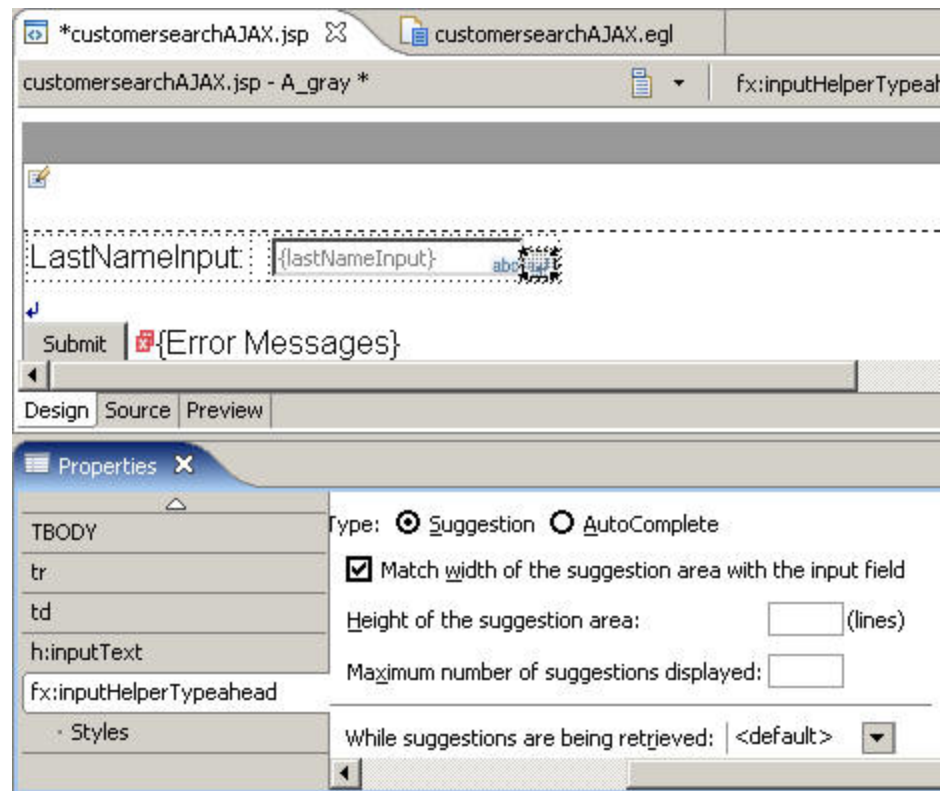
8. Add the following function to the JSF Handler:

```
function displayCustomer()
get customerToDisplay with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where
LASTNAME = :lastNameInput
};
end
```

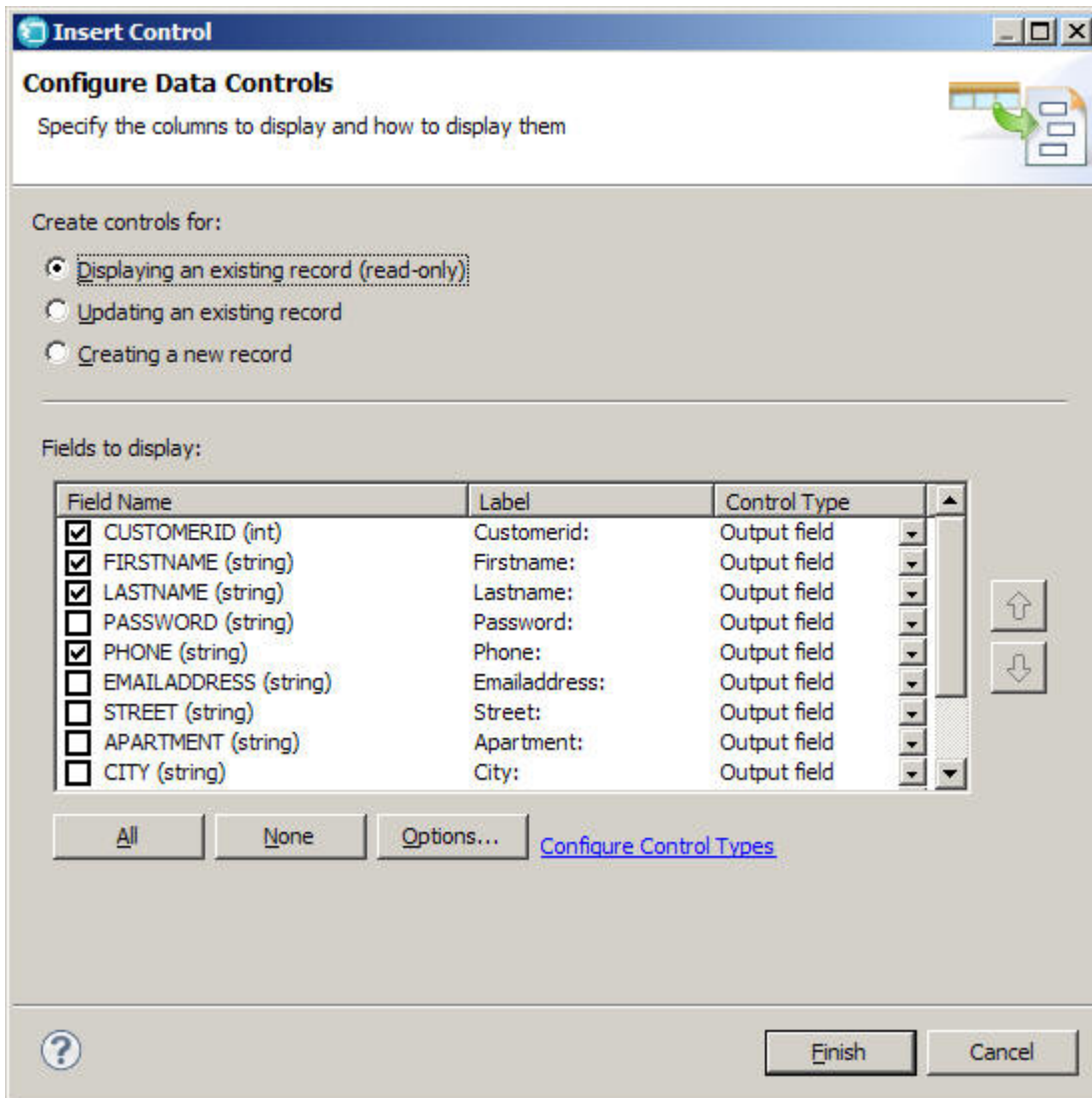
This function will be bound to a button on the page. When the type-ahead control has helped the user find the name of a unique customer, the user will click the button and run this function to retrieve the customer's details.

9. Optimize imports and save the JSF Handler file.
- 10.

11. Expand **JSF Handler** and **Data** in the Page Data view. Drag the `lastNameInput` variable onto the `customersearchAJAX.jsp` page. The Insert Record window opens.
12. Click **Updating an existing record** and make sure that the **Control type** for the input control is set to **Input Field**. You can use type-ahead only on JSF input controls.
13. Click **Options**.
14. In the Options window, clear the **Delete button** check box and select the **Submit button** check box. Leave the default label of Submit for the button.
15. Click **OK**.
16. Click **Finish**. The input control and a button are created on the page. Because of the **TypeaheadFunction** property on the variable, the new input control is automatically configured for type-ahead. You can view the options for the type-ahead functionality, such as the length of the delay before offering suggestions, by clicking the type-ahead icon to the right of the input control and opening the Properties view:



17. Drag the `CustomerToDisplay` variable from the Page Data view onto the bottom of the page. The Insert Control window opens.
18. In the Insert Control window, click **Displaying an existing record (read-only)**.
19. Under **Fields to display**, select the check boxes next to the fields in the customer record that you want to display in your search results. For example, you might select the `CustomerId`, `FirstName`, `LastName`, and `Phone` fields:

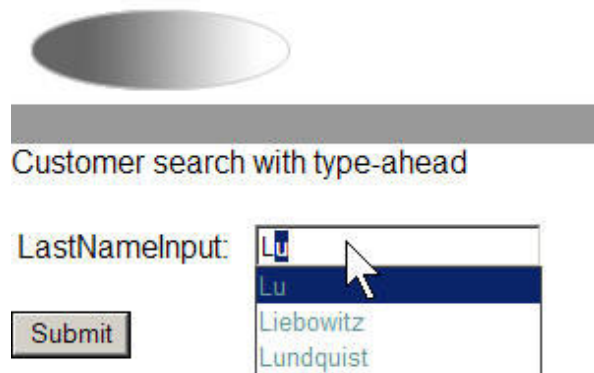


20. Click **Finish**. The controls representing the search results are added to the page.
21. Bind the `displayCustomer()` function to the Submit button on the page by dragging the function from the Page Data view onto the button.

You now have configured the page to use type-ahead. The page looks like this:



You can test the page by running it on the server and typing the first character or two of a customer's name into the input field:



When you choose a customer's name, you can then click **Submit** and see the customer's information:



Here is the complete code of the customersearchAJAX.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed customersearchAJAX.egl file after lesson 6" on page 38.

Lesson checkpoint

In this lesson, you learned to apply type-ahead support to an input field. Type-ahead can help the user enter valid information, but you must beware of the strain on the system providing the suggestions. Each time the value of the input control changes, the function associated with the type-ahead control runs. This function should be as simple as possible to prevent the system from becoming overloaded.

For more information on type-ahead, see *Providing type-ahead support for input controls*.

This concludes the tutorial.

Summary

This is the end of the *Build a JSF search page with EGL* tutorial.

Lessons learned

By completing this tutorial, you learned how to do the following tasks:

- Create a simple search page
- Create an EGL library containing functions called by the search page
- Create a customized EGL record part and display it on a page
- Use SQL statements in EGL code to filter results for a search page
- Create Enhanced Java Server Faces components on the web page and bind functions and variables to them
- Modify your search functions to provide users with a choice of search types
- Customize your search results
- Use type-ahead support on an input control

You may want to continue learning by working with the tutorial application. Try adding this functionality on your own:

- Add a link from the search page to the updatecustomer.jsp page, which you created in a previous tutorial. You will need to make the link pass a parameter as you did on the allcustomers.jsp page.
- Try customizing the search results to include data other than the name, email address, and state of the customer.
- Try applying type-ahead support or AJAX requests to other pages in the project.

Resources

This tutorial used the following resources:

- "Completed SearchLibrary.egl file after lesson 2" on page 31
- "Completed customersearch.egl file after lesson 2" on page 31
- "Completed SearchLibrary.egl file after lesson 3" on page 32
- "Completed customersearch.egl file after lesson 3" on page 33
- "Completed SearchLibrary.egl file after lesson 4" on page 33

- “Completed customersearch.egl file after lesson 4” on page 34
- “Completed SearchLibrary.egl file after lesson 5” on page 35
- “Completed customersearch.egl file after lesson 5” on page 36
- “Completed customersearchAJAX.egl file after lesson 6” on page 38

Following are some links to the help system on topics covered in this tutorial:

- Providing type-ahead support for input controls
- Updating portions of a web page with AJAX requests

Completed SearchLibrary.egl file after lesson 2

This code is the version of the SearchLibrary.egl file completed after lesson 2. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package libraries;
import eglderbydb.data.Customer;

library SearchLibrary type BasicLibrary

function NameAndStateSearch_And(lname STRING in,
state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname
and "STATE" = :state
order by
CUSTOMERID asc
};
end
end
```

Return to “Lesson 2: Add code for the search function” on page 7.

Completed customersearch.egl file after lesson 2

This code is the version of the customersearch.egl file completed after lesson 2. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;
import eglderbydb.data.*;
import libraries.SearchLibrary;

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
onPrerenderFunction = onPrerender,
view = "updatecustomer.jsp",
viewRootVar = viewRoot}

viewRoot UIViewRoot;
searchTerms Customer;
searchResults Customer[0];
resultMessage CHAR(80);
numberOfResults INT;
```

```

// Function Declarations
function onConstruction()
end

function onPrerender()
  if (searchResults.getSize() == 0)
    resultMessage = "No customers found or no search criteria entered.";
  end
end

function searchFunction()
  searchTerms.LastName = searchTerms.LastName::"%";
  SearchLibrary.NameAndStateSearch_And(
    searchTerms.LastName,
    searchTerms.State, searchResults);
  resultMessage = " customer(s) found.";
  numberOfResults = searchResults.getSize();
end
end

```

Return to “Lesson 2: Add code for the search function” on page 7.

Completed SearchLibrary.egl file after lesson 3

This code is the version of the SearchLibrary.egl file completed after lesson 3. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```

package libraries;
import egl DerbyDB.data.Customer;

library SearchLibrary type BasicLibrary

function NameAndStateSearch_And(lname STRING in,
state CHAR(2) in, customer Customer[])
  get customer with
    #sql{
      select
        CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
        EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
        POSTALCODE, DIRECTIONS
      from EGL.CUSTOMER
      where LASTNAME like :lname
        and "STATE" = :state
      order by
        CUSTOMERID asc
    };
end

function NameAndStateSearch_Or(lname STRING in,
state CHAR(2) in, customer Customer[])
  get customer with
    #sql{
      select
        CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
        EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
        POSTALCODE, DIRECTIONS
      from EGL.CUSTOMER
      where LASTNAME like :lname
        or "STATE" = :state
      order by
        CUSTOMERID asc
    };
end
end

```

Return to “Lesson 3: Use the OR search condition” on page 12.

Completed customersearch.egl file after lesson 3

This code is the version of the customersearch.egl file completed after lesson 3. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;
import egl DerbyDB.data.*;
import libraries.SearchLibrary;

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
  view = "updatecustomer.jsp",
  viewRootVar = viewRoot}

viewRoot UIViewRoot;
searchTerms Customer;
searchResults Customer[0];
resultMessage CHAR(80);
numberOfResults INT;
andOr CHAR(3);

// Function Declarations
function onConstruction()
end

function onPrerender()
  if (searchResults.getSize() == 0)
    resultMessage = "No customers found or no search criteria entered.";
  end
end

function searchFunction()
  searchTerms.LastName = searchTerms.LastName + "%";

  if (andOr == "AND")
    SearchLibrary.NameAndStateSearch_And(
      searchTerms.LastName,
      searchTerms.State, searchResults);
  else
    SearchLibrary.NameAndStateSearch_Or(
      searchTerms.LastName,
      searchTerms.State, searchResults);
  end

  resultMessage = " customer(s) found.";
  numberOfResults = searchResults.getSize();
end
end
```

Return to “Lesson 3: Use the OR search condition” on page 12.

Completed SearchLibrary.egl file after lesson 4

This code is the version of the SearchLibrary.egl file completed after lesson 4. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package libraries;
import egl DerbyDB.data.Customer;
```

```

library SearchLibrary type BasicLibrary

function NameAndStateSearch_And(lname STRING in,
state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname
and "STATE" = :state
order by
CUSTOMERID asc
};
end

function NameAndStateSearch_Or(lname STRING in,
state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname
or "STATE" = :state
order by
CUSTOMERID asc
};
end

function getAllCustomerStates(listOfStates STRING[])
customers Customer[0];
counter INT;

get customers with
#sql{
select "STATE"
from EGL.CUSTOMER
order by "STATE" asc
};

listOfStates.removeAll();
for (counter from 1 to customers.getSize() by 1)
listOfStates.appendElement(customers[counter].State);
end

end
end

```

Return to “Lesson 4: Populate a combo box dynamically” on page 16.

Completed customersearch.egl file after lesson 4

This code is the version of the customersearch.egl file completed after lesson 4. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```

package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;
import eglderbydb.data.*;
import libraries.SearchLibrary;

```

```

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
 view = "updatecustomer.jsp",
 viewRootVar = viewRoot}

viewRoot UIViewRoot;
searchTerms Customer;
searchResults Customer[0];
resultMessage CHAR(80);
numberOfResults INT;
andOr CHAR(3);
customerStates STRING[0];

// Function Declarations
function onConstruction()
end

function onPrerender()
SearchLibrary.getAllCustomerStates(customerStates);
if (searchResults.getSize() == 0)
resultMessage = "No customers found or no search criteria entered.";
end
end

function searchFunction()
searchTerms.LastName = searchTerms.LastName + "%";

if (andOr == "AND")
SearchLibrary.NameAndStateSearch_And(
searchTerms.LastName,
searchTerms.State, searchResults);
else
SearchLibrary.NameAndStateSearch_Or(
searchTerms.LastName,
searchTerms.State, searchResults);
end

resultMessage = " customer(s) found.";
numberOfResults = searchResults.getSize();
end
end

```

Return to “Lesson 4: Populate a combo box dynamically” on page 16.

Completed SearchLibrary.egl file after lesson 5

This code is the version of the SearchLibrary.egl file completed after lesson 5. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```

package libraries;
import egl DerbyDB.data.Customer;

library SearchLibrary type BasicLibrary

function NameAndStateSearch_And(lname STRING in,
state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname

```

```

        and "STATE" = :state
    order by
        CUSTOMERID asc
    };
end

function NameAndStateSearch_Or(lname STRING in,
state CHAR(2) in, customer Customer[])
get customer with
#sql{
select
CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,
EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",
POSTALCODE, DIRECTIONS
from EGL.CUSTOMER
where LASTNAME like :lname
or "STATE" = :state
order by
CUSTOMERID asc
};
end

function getAllCustomerStates(listOfStates STRING[])
customers Customer[0];
counter INT;

get customers with
#sql{
select "STATE"
from EGL.CUSTOMER
order by "STATE" asc
};

listOfStates.removeAll();
for (counter from 1 to customers.getSize() by 1)
listOfStates.appendElement(customers[counter].State);
end

end

function getOneState(state Statetable)
get state;
end
end

record customizedResult type basicRecord
fullName STRING {displayName = "Full Name"};
email STRING {displayName = "Email Address"};
stateName STRING {displayName = "State"};
end

```

Return to “Lesson 5: Customize the search results” on page 20.

Completed customersearch.egl file after lesson 5

This code is the version of the customersearch.egl file completed after lesson 5. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```

package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;
import eglderbydb.data.Customer;
import eglderbydb.data.Statetable;
import eglderbydb.primitivetypes.data.STATE;
import libraries.SearchLibrary;
import libraries.customizedResult;

```

```

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
  view = "customersearch.jsp",
  viewRootVar = viewRoot}

viewRoot UIViewRoot;
searchTerms Customer;
searchResults Customer[0];
resultMessage CHAR(80);
numberOfResults INT;
andOr CHAR(3);
customerStates STRING[0];
allRecords customizedResult[0];

// Function Declarations
function onConstruction()
end

function onPrerender()
  SearchLibrary.getAllCustomerStates(customerStates);
  if (searchResults.getSize() == 0)
    resultMessage = "No customers found or no search criteria entered.";
  end
end

function searchFunction()
  searchTerms.LastName = searchTerms.LastName::"%";

  if (andOr == "AND")
    SearchLibrary.NameAndStateSearch_And(
      searchTerms.LastName,
      searchTerms.State, searchResults);
  else
    SearchLibrary.NameAndStateSearch_Or(
      searchTerms.LastName,
      searchTerms.State, searchResults);
  end

  resultMessage = " customer(s) found.";
  numberOfResults = searchResults.getSize();
  if(numberOfResults == NULL)
    numberOfResults = 0;
  end

  generateCustomResults (searchResults);
end

function generateCustomResults(passedResults Customer[])
  allRecords.removeAll();
  oneRecord customizedResult;
  counter INT = 1;
  state Statetable;

  //loop once for each search result returned
  while (counter <= (passedResults.getSize()))
    oneRecord.fullName = passedResults[counter].FirstName ::
      " " :: passedResults[counter].LastName;
    oneRecord.email = passedResults[counter].EmailAddress;
    state.STATE_ABBREV = passedResults[counter].state;
    SearchLibrary.getOneState(state);
    oneRecord.stateName = state.STATE_NAME;
    allRecords.appendElement(oneRecord);
  end
end

```

```

        counter = counter + 1;
    end
end
end

```

Return to “Lesson 5: Customize the search results” on page 20.

Completed customersearchAJAX.egl file after lesson 6

This code is the version of the customersearchAJAX.egl file completed after lesson 6. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```

package jsfhandlers;

import egl DerbyDB.data.Customer;

handler customersearch type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
  view = "customersearch.jsp",
  viewRootVar = viewRoot}

viewRoot UIViewRoot;
lastNameInput STRING {TypeaheadFunction = suggestLastNames};
allLastNames Customer[0];
customerToDisplay Customer;

function onPreRender()
get allLastNames with
#sql{
  select
    LASTNAME
  from EGL.CUSTOMER
  group by
    LASTNAME
};
end

function suggestLastNames(typedCharacters STRING in) returns (STRING[])
matchingLastNames STRING[0];
oneCustomer Customer;
oneCustomerName STRING;

//This variable represents the characters the user has typed.
typedCharacters = StrLib.upperCase(typedCharacters);

//Compare the user input to the values in the database.
for (counter INT from 1 to allLastNames.getSize())
oneCustomer = allLastNames[counter];
oneCustomerName = StrLib.upperCase(oneCustomer.LastName);

if (StrLib.indexOf(oneCustomerName, typedCharacters) == 1)
//This value starts with the same characters.
//Add this value to the type-ahead suggestions.
matchingLastNames.appendElement(oneCustomer.LastName);
end

end
return (matchingLastNames);
end

function displayCustomer()
get customerToDisplay with
#sql{
  select
    CUSTOMERID, FIRSTNAME, LASTNAME, PASSWORD, PHONE,

```

```
        EMAILADDRESS, STREET, APARTMENT, CITY, "STATE",  
        POSTALCODE, DIRECTIONS  
    from EGL.CUSTOMER  
    where  
        LASTNAME = :lastNameInput  
    };  
end  
  
end
```

Return to “Lesson 6: Use type-ahead to prompt the user” on page 24.

Appendix. Notices

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:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, 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.

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
IBM Corporation
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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.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 <http://www.ibm.com/legal/copytrade.html>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.



Printed in USA