XPages Workshop: Building a Simple Application

by Michael McGarel and Roy Rumaner

Introduction

- You will be building a simple XPages application with a minimal number of new design components. Along the way, we'll introduce you to new concepts and get you familiar with XPages terminology.
- This workshop assumes knowledge of traditional Notes concepts such as documents, views and forms.
- Knowledge of standard Web technology (e.g. HTML, style sheets) is helpful but not necessary.

What are XPages?

- XPages is the collective name for the new design and coding elements from IBM/Lotus. The intention of XPages is to provide a modern Web interface for an application, whether it's on the Web or within the Notes client.
- Once you learn it, XPages is a faster, more efficient way to build good-looking, better-functioning interfaces.
- The final result of XPages coding is an XML (eXtensible Markup Language) file which conforms to the standards of JSF (Java Server Faces). This is important to know because you're going to be looking at the pre-compiled page source code in this workshop. Even if you're familiar with HTML, you may be wondering about all these unusual tags.

About OneUI

OneUI is the name for IBM's layout and formatting standards. Since it's used in many Lotus products, it is good to be familiar with it. You'll be using it as a base within this application (*see Illustration 1*).

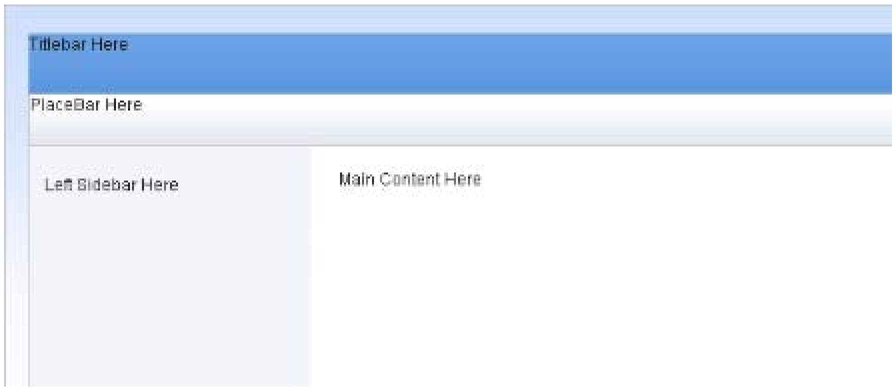


Illustration 1: The OneUI Framework

What You're Starting With

b We're giving you an existing database with a form, a view and some test data so you don't waste your time doing something you could do in your sleep.

b Providing the elements and data will also demonstrate how you can take an existing application and modify it for XPages use.

Pre-Planning is Key

With XPages, because so many design elements are dependent on each other, knowing where you going and how you're getting there is more important than ever.

Task 1: Modify the Designer Palette

Since Designer is now based on Eclipse we have the use of customizable palettes to drag and drop items into our editing window. We're going to modify your existing palettes and possibly select a new one. This way, all the design controls easily available.

1. Open the Domino Designer client (at least 8.5).
2. Open the File menu and select Preferences.
3. In the navigator on the left, under Domino Designer, select Palette.
4. Make sure all the boxes are checked.
5. Click Apply.
6. Click OK.

Task 2: Review the Classic Notes Design Elements

1. Open the database **SuggestionBox.nsf**.

The Form

The form is called **fm_Suggestion**. (*see Appendix: Illustration 2*)
Notice that:

b The form does not need formatting as it is just being used as a data container. No user will see it.

b The table structure is for easier maintenance and troubleshooting.

b As an option, and what we consider a best practice, the third column is for any type of comment you want to add, e.g., a field description or a design note.

The View

The view is called **By Dept** with an alias of vwByDept. Notice that:

b It is built for easy access and efficiency (e.g., no sortable columns).

b It does not need to be visually pretty. Again, no user will see it directly.

Task 3: Adding the OneUI Theme

Our first application design task is to implement the OneUI standards. The easiest way to accomplish this is making use of the Theme design element. (*See Appendix: Illustration 5.*) Themes are a container for style sheets, custom styles and/or JavaScript.

1. Under Resources, double-click Themes (which will be empty).
2. Click the New Theme button.
3. Name it `mwlugui`.
4. Click the plus sign to open the Theme properties.
5. Double click on "Extends"
6. You can edit the Theme document 2 different ways:

   a) Select the Properties tab
   b) Click on "value."
   c) Enter `oneui`.

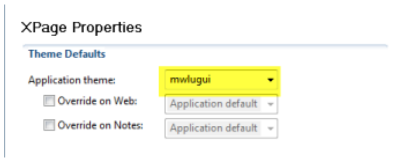
   OR

   a) Select the Source tab
   b) Replace "**webstandard**" with `oneui`.

```
<theme extends="oneui">
```

9. Save and close your Theme.

10. Double-click the Application Properties.
11. Select the XPages tab (along the bottom of the editor window).
12. Under Theme Defaults, change the "Application theme" to "mwlugui."



XPage Properties

Theme Defaults

Application theme: mwlugui

Override on Web: Application default

Override on Notes: Application default

13. Save and close the Theme..

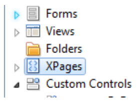
Task 4: Creating the XPages

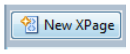
NOTE: XPages are containers for your other elements. As a suggested best practice, the fewer controls and resources referenced directly by the XPage, the more reusable your code is.
We know we will have three Xpages:

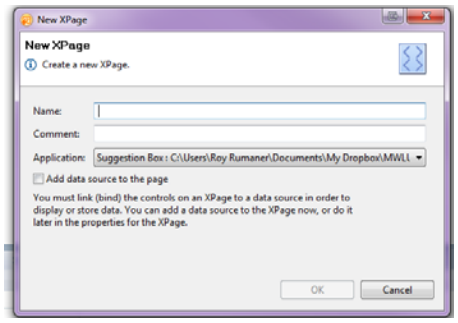
- b Home
- b ByDept
- b Suggestion

We will create these now so we have targets for links within future custom controls.

To create an Xpage

1. Double click the XPage option. 

1. Click the New XPage button. 



New XPage

Create a new XPage.

Name:

Comment:

Application: Suggestion Box : C:\Users\Roy Rumaner\Documents\My Dropbox\MWLL

Add data source to the page

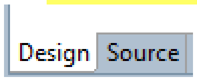
You must link (bind) the controls on an XPage to a data source in order to display or store data. You can add a data source to the XPage now, or do it later in the properties for the XPage.

3. Name it **`Home`** .
4. Don't add anything yet. We'll add custom controls after we create them in the following tasks.
5. Close the XPage.
6. Do the same for **ByDept** and **Suggestion.**

TASK 5: Building the layout_Header Custom Control

1. Double-click Custom Controls (which will be empty).
2. Click the New Custom Control button.
3. Name it **`layout_Header`**.

NOTE: As a suggested best practice, we prefix our element names with its design function.

4. Drag the **Block-level content** Container Control onto the editor window.
5. Click within the box in the editor window and the box disappears. Enter `Suggestion Box` where the cursor is.
6. Click at the end of the text to get to the Paragraph control.
7. Name the control **`header`** under Properties below.
8. In the Style Properties, enter **`lotusTitleBar`** in the Class box
9. Switch to the Source tab 

NOTE: In the Source code where it says <xp:view>, this does not refer to a Notes view, instead it refers to a JSF element.

10. Look for the line:

```
<xp:div id=”header” styleClass=”lotusTitleBar”>Suggestion Bar</xp:div>
```

11.Apply the <h1> HTML tag to the text.

```
<xp:div id=”header” styleClass=”lotusTitleBar”><h1>Suggestion Bar</h1></xp:div>
```

10.Switch back to the Design tab and verify your changes.

- Because this header is not using any XPage properties, we can remove the “xp:” from the div references and change “styleClass” to “class.”

```
<div id=”header” class=”lotusTitleBar”><h1>Suggestion Bar</h1></div>
```

NOTE: This uses just a little bit less memory when the browser displays the page.

12. Save and close this Control.

Congratulations! You have created your first Custom Control!

Task 6: Building the layout_Menu Custom Control

12.Double-click Custom Controls (if it's not open).
13.Click the New Custom Control button.
14.Name it **`layout_Menu`**.
15.Drag the **Block-level content** Container Control onto the editor window.
16.Type `Views` into the box.
17.Name the control **`lotusMenu`** in the Properties box.
18.In the Style Properties, in the Class box enter **`lotusMenu`** again.
19.Switch to the Source tab.
20.Add the HTML tag <h3> before the word “**Views**” and </h3> after the word “**Views.**”

```
<h3>Views</h3>
```

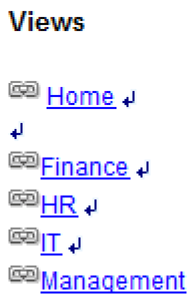
10.Drag the **Link** Custom Control onto the editor window under the title.
11.The name will be auto-generated. You can leave this alone or make up a name yourself.
12.Name the label `Home.`
13.There are 3 types of links that you can choose from at this point. We want the Open page type. Select **Home** from the dropdown.

➢ *Now you know why we created these three pages up front.*

14.Add two carriage returns after the word “**Home.**”
15.Drag another **Link** Custom Control onto the editor window under Home.
16.Change the **label** to `Finance.`
17.There are 3 types of links that you can choose from at this point. We want the URL type. In the edit box, type `/ByDept.xsp?dept=Finance.`

18. Add a carriage return after the word "**Finance.**"
19. Repeat steps 15 – 18 for `HR, IT` and `Management` replacing the word Finance in the URL box with the appropriate option.
20. Save and close this Control.

Congratulations! You have created your first XPage Menu!


Views

Home ↵
↵
Finance ↵
HR ↵
IT ↵
Management

Task 7: Create the content_Home Custom Control

1. Open Custom Controls.
2. Click the New Custom Control button
3. Drag the **Block-level Content** Container Control onto the editor window.
4. Name the control **`imgHolder`** in the Properties box.
5. There is no need to add a Style name at this time.
6. Drag the Image Core Control into the **imgHolder** container. A Select Image box will come up and display all image resources in the database.
7. Select box.jpg and click OK.
8. Save the control.

Create the Suggestion Link

1. Under the image, drag in another **Block-level Content** Container Control.
2. Name the Control **`actionButtons.`**
3. In the Style Properties, in the Class box enter **`lotusBtnContainer.`**
4. Drag a **Span** Core Control into the **Block-level** Content Container Control.
5. Next select Style and enter `lotusBtn lotusBtnSpecial` in the Class box. This applies two different classes to the button.
6. Drag a **Link** Core Control over the top of the **Span** Control.
7. Name the Link **`btnSuggest`** and label it as `Make Suggestion.`
8. Select Open page as the Link type and choose Suggestion.

Task 8: Create the content_Suggestion Custom Control

1. Create a new Custom Control and call it `content_Suggestion`.
2. Click on the Data property and select **Add** Data source and select Domino Document.
3. In the Data Source: Domino Document section, create a name for the variable. This is the reference you will use to access data.
4. Type `Suggestion` at the top of the page.
5. In the Source tab, add <h2> and </h2> around the title.

```
<h2>Suggestion</h2>
```

6. In the Form option, select **fm_Suggestion.**
7. Leave **Create Document** as the Default action.
8. Click on Data tab in the upper right corner of the Designer.
9. You are now presented with a list of the fields that are on the form you selected.
10. Select the fields you want to appear on the XPage. You can select multiple fields if you wish.
11. Drag the selected fields under the word Suggestion.

NOTE: The Designer will automatically create a table with the fields and a label for each. All the fields become Text fields.

12. Remove the "1" from the names of the fields so that the field names match their counterpart on the form.

Labels

1. Remove the "1" from the label name and change the label text.
2. Change the Target Control under Options to the new name of the field. For example sgAuthor1 became sgAuthor so the new target control is sgAuthor.
3. Repeat this for the remaining field labels.

Task 9: Changing Field Types and Setting Default Values

Because all the fields we dragged over were converted to text fields, we have to replace them with their proper field control types.

Adding a default value to the sgAuthor field

1. Click on the sgAuthor field.
2. Open the Data section under the Properties tab.
3. Click the blue diamond next to the Default value field and select Compute Value.
4. In the editor window, type the following:

```
var userName:NotesName = session.createName(@UserName());
return (userName.getCommon());
```

5. Click OK.

Replacing the sgSuggestion Field

1. Delete the sgSuggestionText field and drag a **Multiline Edit Box** Core Control into its place.
2. Name the edit box using the same name as the field you just deleted: **`sgSuggestionText`**.
3. Set the Height to 10 Ems and the Width to 20 Ems.
4. Under the data tab, select "Simple data binding."
5. The **data source** should be sgDoc.
6. Bind to will be sgSuggestionText.
7. This is our only required field. Open the Validation section and click the box for Required field. Then enter a message such as `Please Add A Suggestion`.

Replace the Department Field

1. Remove the sgDepartment field and drag a Combo Box Core Control into its place
2. Name the Combo Box using the same name as the field you just deleted **`sgDepartment`**
3. Under the data tab, select "**Simple data binding**".
4. The **data source** should be sgDoc.
5. **Bind to** will be sgDepartment.
6. Next we need to add the choices for the dropdown.
7. Under the Values tab, select Add item and add `-Please select-` to the Label and leave the Value blank. The label is what you see on the screen and the value is how it is stored.
8. Click the Add Item button again. This time enter `Finance` under both Label and Value.
9. Repeat the step above for the remainder of the labels/values: `HR, IT, Management`.

Replace the Importance Field

1. Repeat the steps for the Department field for the sgImportance field.
2. For the default value, enter the number 2.
3. For the labels, enter `Very important, Important` and `Nice to have` and their respective values will be `1, 2,` and `3`.

Add the Button Container and the Submit Button

1. Under the last field, drag in another **Block-level Content** Container Control.
2. Name the Control **`actionButtons`**.
3. In the Style Properties, in the Class box enter **`lotusBtnContainer`**.
4. Drag a **Span** Core Control into the Block-level Content Container Control.
5. Next select Style and enter `lotusBtn lotusBtnSpecial`. This applies two different classes to the button.
6. Drag a **Link** Core Control within the **Span** Control.
7. Name the Link **`btnSubmit`** and label it as `Submit.`

After all this work, you are finally about to write some actual JavaScript.

Add Code to the Submit Button

1. In the Events tab, select Mouse and then onclick.
2. Because we are going to be using SSJS, we need to write the code under the Server tab. Select the Server tab and then the script Editor radio button.

3. Go into the Resources section of Designer and open the file “**code.txt**.” Copy the text into the script editor window,; otherwise, enter the code below.

```
var baseDoc:NotesDocument = sgDoc.getDocument(true);
baseDoc.replaceItemValue("Form","fm_Suggestion");
var curDate = session.createDateTime("Today");
curDate.setNow();
var userName:NotesName = session.createName(@UserName());
baseDoc.replaceItemValue("sgSubmittedDT",curDate);
baseDoc.save();
context.redirectToHome();
```

4. Under the Server Options, select “**Full Update.**”

Add the Cancel Button

1. Under the last field, drag in another **Block-level Content** Container Control.
2. Name the Control **`actionButtons`**.
3. In the Style Properties, in the Class box enter **`lotusBtnContainer`**.
4. Drag a **Span** Core Control into the Block-level Content Container Control.
5. Next select Style and enter `lotusBtn lotusBtnSpecial` This applies two different classes to the button.
6. Drag a **Link** Core Control within of the **Span** Control.
7. Name the Link **`btnCancel`** and label it as `Cancel.`
8. In the Events tab, select Mouse and then select onclick.
9. Select the Server tab and then the Simple Action radio button.
10. Click Add Action which opens a dialog box.
11. Change the Category to **All.**
12. Change the Action to **Open Page.**
13. Change the page to **Home.**
14. Under Server Options, select Full Update and click the radio button for “Do not validate or update data.”

Task 10: Create the layout_Container Custom Control

This custom control stores other custom controls and provides the framework for the Web pages.

1. Open Custom Controls.
2. Select New Custom Control.
3. Drag the **Block-level Content** Container Control onto the editor window.
4. Name the element **`lotusFrame`**.

5. We won't be using XPage properties on this element, so you can remove the xp: before the div.
6. From the Custom Controls palette (by default on your lower right of the screen), drag the layout_Header element into the lotusFrame div.
7. Check the Source tab to make sure the header is within the div. If not, move the text where necessary.

   It should something look like this:

```
<div id="lotusFrame">
<xc:layout_Header></xc:layout_Header>
</div>
```

4. Select the design tab. Drag the **Block-level Content** Container Control onto the editor window under the header.
5. Name the element **`lotusMain`**.
6. Go to the Source tab and make sure the page looks something like this:

```
<div id="lotusFrame">
<xc:layout_Header></xc:layout_Header>
<div id=”lotusMain”></div>
</div>
```

NOTE: If you add some temporary text, it gives you a target to drag things to, for example:

```
<div id="lotusFrame">
<xc:layout_Header></xc:layout_Header>
<div id=”lotusMain”>here</div>
</div>
```

7. Drag a Block-level Content Container Control into the lotusMain div.
8. Name the element **`lotusColLeft`**
9. Drag a Block-level Content Container Control under the lotusColLeft div.
10. Name the element **`lotusContent`**
11. Now we'll add a menu within the left column. Drag the layout_Menu Custom Control within the lotusColLeft div.
12. Now we'll add a area to add custom controls for the individual XPages. Drag the Editable Area Core Control within the lotusContent div. Keep the default name.

    So far, your XPages source should look something like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core"
        xmlns:xc="http://www.ibm.com/xsp/custom">
<div id="lotusFrame">
        <xc:layout_Header></xc:layout_Header>
        <div id="lotusMain">
                <div id="lotusColLeft">
                        <xc:layout_Menu></xc:layout_Menu>
                </div>
                <div id="lotusContent">
                        <xp:callback facetName="facet_1" id="callback1"></xp:callback>
                </div>
         </div>
</div>
</xp:view>
```

13. Now let's add the footer. Drag the **Block-level Content** Container Control onto the editor window after the lotusMain div.
14. Name this element **`footer`**.
15. Select Style and make the Class **`lotusFooter`**.
16. Within the div, type in `MWLUG 2010 Xpages Workshop.`
    The bottom of the source should look something like this.

```
                <div id="lotusContent">
                        <xp:callback facetName="facet_1" id="callback1"></xp:callback>
                </div>
         </div>
        <xp:div id="footer" styleClass="lotusFooter">
                MWLUG 2010 XPages Workshop
        </xp:div>
</div>
</xp:view>
```

17. The final thing we'll add to this custom control is a style sheet. Since this control will be used on all XPages, this give us a single source for future style changes.
18. After selecting the custom control itself, select Resources from the Properties window.
19. Click the Add Style Sheet button.
20. Select the **suggestion.css** file then click OK.
21. Save and close.

Task 11: Create the content_ByDept Custom Control

This custom control will display the submissions.

1. Open Custom Controls.
2. Click New Custom Control.
3. Name this control **`content_ByDept`**.
4. Click the Data section in the Properties box.
5. Under Data Source, click the Add button then select Domino View.
6. In the Data Source section on the right, keep the application as the current one.
7. For the view name, select "By Dept" from the drop-down.
8. For Data source name, replace the default with **`byDept`**. This will be your view reference in the source code.
9. Open the All Properties section, open data then data again, find dominoView[0] to get to the **categoryFilter** property.
10. Click on the diamond to access the Computed Value dialog box.
11. Enter the following line of code:

```
context.getUrlParameter('dept')
```

12. Click OK to close the dialog box
13. On the page, type in the text `By Department` which we'll use as the title.
14. In the Source tab, type the `<h2>` tag before and `</h2>` after the title. Example:

```
<h2>By Department</h2>
```

15. Open the Design tab again, then drag the **Table** Container Control onto the editor window.
16. The table needs 2 rows and 3 columns.
17. In the first cell of the top row, enter `Submitted`. In the second cell in the row, enter `Author`. In the third cell, enter `Suggestion`.
18. Open the Source tab so we can make the row a properly formatted HTML header row. Edit the source code:
19. Add `<thead>` before and `</thead>` after the first table row.
20. Replace the the <xp:td></xp:td> tags for each cell within that row with `<th>` and `</th>`. When done, it should something look like this:

```
<xp:table>
<thead>
        <tr>
        <th>Submitted</th>
        <th>Author</th>
        <th>Suggestion</th>
        </tr>
</thead>
```

21. Open the Design tab. Drag a **Repeat** Container Control onto the editor window.

22. In the Repeat Properties, leave the default name as is.
23. Under Data Binding, the source will be the view reference byDept.
24. In the Options section on the right, enter rowData for the Collection name. This is a reference to the view entry.
25. Enter `rowIndex` for the Index name. This indicates the position of the view entry within the repeat.
26. We want the Repeat Control to create a new row for each view entry, so it must start before and end after the second row in our table. Open the Source tab and edit the code until you get the following:

```
</thead>
<xp:repeat id="repeat1" rows="5" value="#{byDept}" var="rowData"
        indexVar="rowIndex">
        <xp:tr>
                <xp:td>
                </xp:td>
                <xp:td>
                <xp:td>
                </xp:td>
                <xp:td>
                </xp:td>
        </xp:tr>
</xp:repeat>
```

NOTE: In order to behave properly, a Repeat Control needs to have its header and footer set up manually.

27. Go into the Resources section of Designer and open the file “**facets.txt**.” Copy the text in the file.
28. Come back to this custom control and open the Source tab. Add the facets text below the start of the repeat and above the <xp:tr>. It should look something like this:

```
<xp:repeat id="repeat1" rows="5" value="#{byDept}" var="rowData"
indexVar="rowIndex">
<xp:this.facets>
        <xp:text disableTheme="true" xp:key="header" escape="false">
                <xp:this.value><![CDATA[<tbody>]]></xp:this.value>
        </xp:text>
        <xp:text disableTheme="true" xp:key="footer" escape="false">
                <xp:this.value><![CDATA[</tbody>]]></xp:this.value>
        </xp:text>
</xp:this.facets>
<xp:tr>
```

29. Now we'll add the data. Open the Design tab and drag a Computed Field Core Control into the first empty cell under the Submitted heading.
30. Change the name to **`cmpSubmitted`**.
31. Under Value, change the Data Binding to JavaScript.
32. In the script box, enter the following formula:

```
rowData.getColumnValue('$4')
```

33. Drag a Computed Field Core Control into the empty cell under the Author heading.
34. Change the name to **`cmpAuthor`**.
35. Under Value, change the Data Binding to JavaScript.
36. In the script box, enter the following formula:

```
rowData.getColumnValue('sgAuthor')
```

37. Drag a Computed Field Core Control into the empty cell under the Suggestion heading.
38. Change the name to **`cmpSuggestion`**.
39. Under Value, change the Data Binding to JavaScript.
40. In the script box, enter the following formula:

```
rowData.getColumnValue('sgSuggestionText')
```

41. Save and close.

Task 12: Add Content to the XPages

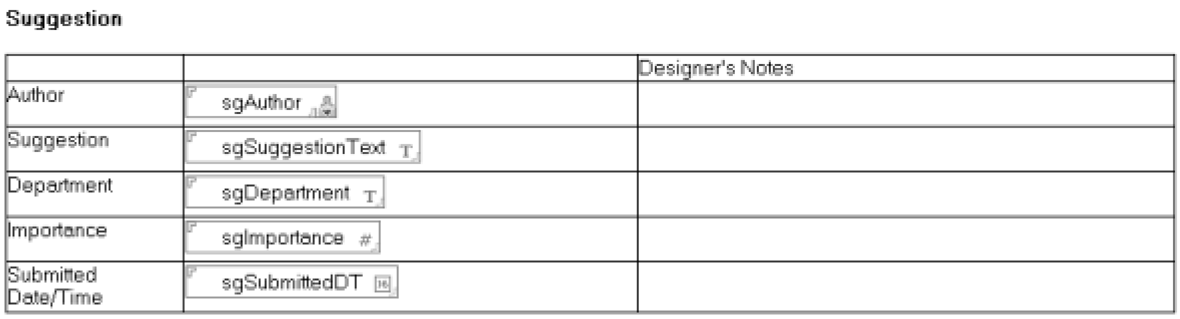
Here's where the planning pays off.

1. Open the Home XPage.
2. Drag the layout_Container custom control onto the page.
3. Drag the content_Home custom control into the facet_1 element.
4. Save and close it.
5. Preview in the Web browser.
6. Repeat the process for the ByDept and Suggestion XPages with their respective custom controls.

Congratulations! Your application is done!

(For now, anyway. Until the inevitable user modifications, new requests, etc.)

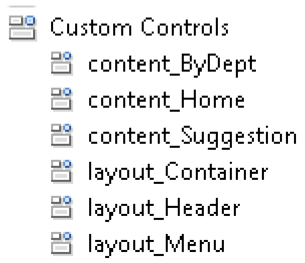
Appendix 1: Additional Illustrations



Suggestion

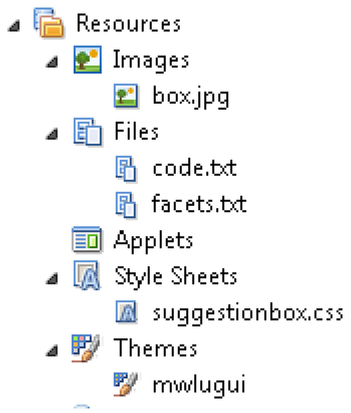
		Designer's Notes
Author	sgAuthor	
Suggestion	sgSuggestionText	
Department	sgDepartment	
Importance	sgImportance	
Submitted Date/Time	sgSubmittedDT	

Illustration 2: fm_Suggestion



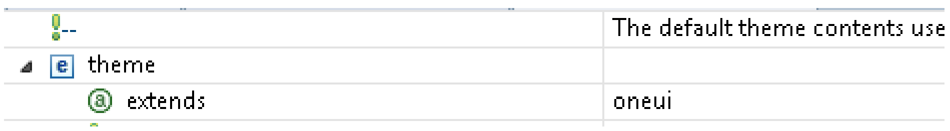
- Custom Controls
  - content_ByDept
  - content_Home
  - content_Suggestion
  - layout_Container
  - layout_Header
  - layout_Menu

Illustration 3: Custom Controls needed for the application



- Resources
  - Images
    - box.jpg
  - Files
    - code.txt
    - facets.txt
  - Applets
  - Style Sheets
    - suggestionbox.css
  - Themes
    - mwlugui

Illustration 4: Resources needed for the application



!--	The default theme contents use
theme	
extends	oneui

Illustration 5: Setting the oneui theme

Appendix 2: suggestionbox.css

```
/* Customizes oneui for the Suggestion Box */

th {
      font-weight: bold;
}

#lotusFrame {
      width: 500px;
}

#lotusTitleBar h1, .lotusTitleBar h1 {
      font-size: 2.8em;
      text-align: center;
}

.lotusTitleBar {
      //height: 12em;
}

.lotusForm h2 {
      margin: 0 0 15px 0;
}

.lotusFooter {
      text-align: right;
}

.lotusMenu {
            background-image: none;
            border: 1px solid #629be0;
            padding-bottom: 10px;
}

.lotusMenu h3 {
      color: #629be0;
      padding-bottom: 6px;
}

.lotusMenu a {
      padding: 2px 0 2px 20px;
}

.lotusBtnContainer {
      margin-top: 24px;
}

.lotusBtn {
      float: left;
}
```