

JavaScript & Base Forms

For some time I have wanted to experiment with JavaScript and OpenOffice.org. Finally I was able to start running some tests. The *OOO.js* file contains a brief library with functions and objects to help connect and manipulate Base databases.

The most fundamental function is *createUnoService(string service_name)*. This function takes a service name and returns a service object. I have also used this function to create another auxiliary function called *createRowSet(string DataSourceName, int CommandType, string Command)* which takes the data source name, command type, command string, and returns a rowset object.

HTML Forms & Base

While Base can export a form to HTML, it does not allow for that HTML form to communicate with the database. The reason is that only HTML is created, without any code to retain the link/communication to the database.

In addition to the previously noted functions, the *OOO.js* file also has a few objects that help bind an HTML form to a Base Database. Table 1 lists the objects that help provide some features of a Base Form to HTML forms.

Table 1: HTML/Base Form Objects

| Object Name | Description |
|--------------------|---|
| Container | This is a basic object that acts like a container. Elements can be accessed by name and index. |
| DataFormModel | Object to represent a basic form model—inherits from the <i>Container</i> object. This object has all information pertaining to the underlying table/data source. Also, it is responsible for fetching and data as well as committing changes. The method <i>addModel(oModel)</i> adds a model. |
| DataForm | Form controller which has a <i>DataFormModel</i> as a property (<i>Model</i>). This object gets the model information and displays it in the HTML form. Also, takes HTML element data and passes it to the model for storage. Also inherits from the <i>Container</i> object. The method <i>addControl(oControl)</i> adds a new control. |
| DataControlModel | This is the base object for a data aware control model; it contains bound column information which is used to fetch the column value from the data form model. Data control models are contained in the data form model. All control models inherit from this object. |
| DataControl | Base control (view) object which has a <i>DataControlModel</i> object as its <i>Model</i> property. Additionally, it is bound to a DOM element which interacts with the user. This object uses the model data to display the data in the DOM element, as well as passes data from the DOM element to the model, which is later committed to the database by the form model. All control views inherit from this object. Essentially this should be just an extension to the controller for an HTML form element. It only adds features to synchronize with the model. <i>DataControls</i> are contained in the form controller (<i>DataForm</i>). |
| DataTextFieldModel | Model for a text field—text form element. |

| Object Name | Description |
|-------------------|--|
| DataTextField | View for a text field. |
| DataListBoxModel | Model for a list box. Allows selection of a list source such as a table, query or SQL command. |
| DataListBox | View for list box—SELECTHTML element. This object has a property called <i>LinkedWith</i> which refers to another DataListBox. When the value changes in this list box, the linked list box is filtered according to the new value. |
| DataCheckBoxModel | Model for a check box—checkbox form element. |
| DataCheckBox | View for a check box. |
| LinkedControl | A simple object used to specify a linked list box. The <i>Control</i> property refers to the target DataListBox, the <i>Column</i> property refers to the column in the target's data source that will be used for filtering. For example, consider the source for a linked list box is <i>SELECT ID,NAME FROM PROPERTY_TYPES</i> . If the <i>Column</i> property is <i>ID</i> , the filter generated could be something along the lines of <i>ID=1</i> where 1 is the current value of the (master) list box. |

Consider the following HTML form. Note that some additional attributes have been added to the form controls which help determine how to create the form model and controller/view objects.

Listing 1: HTML form with special attributes

```

1 <FORM ID="FORM1">
2   Id: <INPUT ID="txtID" TYPE="TEXT" DataField="ID"/><BR>
3   Name: <INPUT ID="txtNAME" TYPE="TEXT" DataField="NAME"/><BR>
4   Group Id: <SELECT ID="lbGROUPID" SIZE=1 DataField="GROUPID"
5               SourceType=0 Source="APPGROUPS" BoundField=1
6               DisplayField=2> </SELECT><BR>
7   Decription: <INPUT ID="txtDESCRIPTION" TYPE="TEXT"
8               DataField="DESCRIPTION"/><BR>
9   Required?: <INPUT ID="cbREQUIRED" NAME="cbREQUIRED" VALUE=""
10              TYPE="checkbox" DataField="REQUIRED"/>

```

The text and check boxes have only one special attribute—DataField. This attribute indicates the column name in the source to which this control is bound. The list box (SELECT DOM Element), however, has several more special attributes—SourceType, Source, BoundField, DisplaField, in additiona tothe DataField attribute. The source type indicates the type for the list source (TABLE/0, QUERY/1, COMMAND/2). The source is the name of the source (e.g. A table/query name, or valid SQL command). The BoundField property indicates which column from the source is to be stored in the database—corresponds to the DataField property. The DisplayField property indicates which field/column is to be displayed. This allows the list box to display one field, and store another. Code listing 2 shows the JavaScript code that binds the HTML form to the underlying database.

Listing 2: Binding HTML form to database

```

1 var FormModel= new DataFormModel("FORM1","OOFORUMTESTS",0,"APPCATEGORIES");
2 /*
3 NOW CREATE FORM CONTROLLER. TAKES A FORM MODEL AS PARAMETER
4 */
5 var Form= new DataForm(FormModel);
6

```

```

7 //ARRAY WITH HTML FORM CONTROL IDS--FOR EASY ITERATION/CREATION
8 //--TEXT FIELDS ONLY
9 var FormControlNames= new Array("txtID","txtNAME","txtDESCRIPTION");
10 for(var i=0;i<FormControlNames.length;i++)
11 {
12 //CREATE REFERENCE TO DOM ELEMENT
13 var Elt=document.getElementById(FormControlNames[i])
14 //NOW CREATE THE MODEL WHICH TAKES DOM ELEMENT AS PARAMETER.
15 //DOM ELEMENT IS ONLY NEEDED TO GET DATA BINDING INFORMATION
16 var model=new DataTextFieldModel(Elt);
17 //ADD CONTROL MODEL TO FORM MODEL
18 FormModel.addModel(model);
19 //NOW CREATE FORM CONTROLLER, TAKES MODEL AND DOM ELEMENT AS PARAMETER
20 //THE CONTROLLER WILL CREATE A LINK BETWEEN THE CONTROLLER
21 // AND THE DOM ELEMENT FOR DATA
22 //SYNCHRONIZATION--TAKES MODEL DATA AND DISPLAYS IN DOM ELEMENT, AS WELL AS
23 //DOM ELEMENT DATA
24 //TO THE MODEL WHICH IN TURN UPDATES THE DATABASE
25 var control=new DataTextField(model,Elt);
26 //FINALLY, ADD TO FORM CONTROLLER
27 Form.addControl(control);
28 }
29 //CREATE OTHER ELEMENTS--LIST BOX IN THIS CASE
30 var elt=document.getElementById("lbGROUPID");
31 var model= new DataListBoxModel(elt);
32 var lb2= new DataListBox(model,elt)
33 FormModel.addModel(model);
34 Form.addControl(lb2);
35
36 //CREATE A CHECK BOX
37 var elt=document.getElementById("cbREQUIRED")
38 var model=new DataCheckBoxModel(elt);
39 FormModel.addModel(model);
40 var control= new DataCheckBox(model,elt);
41 control.Enabled=false;
42 Form.addControl(control);
43 /**/
44
45 Form.refresh();

```

The form model, controller, as well as form controls have a method called *refresh()* which refreshes the data. In the case of the views, this entails writing the new data in the model to the DOM elements. The *refresh()* method for the form model and controller cycles through the contained elements and calls their refresh method. In this way, line 45 of code listing 2 refreshes the entire form.

The form model and controller also have navigation methods such as *first()*, *next()*, *previous*, and *last()*. In the model, these methods perform the respective action on the underlying row set. In the controller, these methods call the model's corresponding method, and calls the *refresh()* method to update the control views (DOM elements).

The following code fragment creates navigation buttons. Note that *onClick* event handler simply calls the respective navigation method of the form controller—on that particular instance.

```

<BUTTON onClick="Form.first();" Value="|<"> |<</BUTTON>
<BUTTON onClick="Form.previous();"> <</BUTTON>
<BUTTON onClick="Form.next();"> >>/BUTTON>
<BUTTON onClick="Form.last();">>|</BUTTON>
<BUTTON onClick="FormModel.commit();">Save Form</BUTTON>

```

This code can be placed at the bottom of the form tag.

Linking List Boxes

The previous example is relatively simple. Consider now that two lists are linked. In other words, the contents of list box 2 are dependent upon the selected value of list box 1. This is a simple matter of assigning list box 2 as the to the *LinkedWith* property of list box 1. See listing 3 for the HTML form.

Listing 3: Linked list boxes

```

1 <FORM ID="FORM2">
2 Applications Group:
3     <SELECT ID="lbAPPGROUPS" SIZE=1 DataField="GROUPID" Sourcetype=0
4         Source="APPGROUPS" BoundField=1 DisplayField=2></SELECT><BR>
5 Applications Category:
6     <SELECT ID="lbAPPLICATIONS" SIZE=1 DataField="GROUPID" SourceType=0
7         Source="APPCATEGORIES" BoundField=1 DisplayField=4>
8     </SELECT><BR>
9 </FORM>

```

Note that the form looks the same as before—the `SELECT` elements anyways. Ideally, the linked control would be specified in the HTML markup. However, this cause a reference to a control that has not been created. For example, if the first list box is linked to the second one (which happens to be the case in this example), the constructor for the list box controller would try to reference the second list box, which is not yet created. An alternative is to perform this task after all the controls have been added—by some initializing method of the form controller. Listing 4 shows the corresponding JavaScript code.

Listing 4: Linking the list boxes

```

1 //A SIMPLE FORM WITH TWO LINKED SELECT (LIST) BOXES)
2 var FormModel2= new DataFormModel("FORM2","OOFORUMTESTS",0,"APPLICATIONS");
3 var Form2= new DataForm(FormModel2);
4
5 var elt=document.getElementById("lbAPPLICATIONS");
6 var model= new DataListBoxModel(elt);
7 var lb= new DataListBox(model, elt)
8 FormModel2.addModel(model);
9 lb.ReadOnly=true;
10 Form2.addControl(lb);
11
12 var elt=document.getElementById("lbAPPGROUPS");
13 var model= new DataListBoxModel(elt);
14 var lb2= new DataListBox(model,elt)
15 FormModel2.addModel(model);
16 Form2.addControl(lb2);
17

```

```
18 Form2.getByName("lbAPPGROUPS").LinkedWith = new LinkedControl(lb,"GROUPID");
19
20 Form2.refresh();
```

The code to link the list boxes is in line 18. The rest of the code is exactly as the previous example. The original command for the second list box is:

```
SELECT *FROM APPCATEGORIES
```

consider that at some arbitrary point the value of the first list box is (1). When the *OnChange* event triggers on the first list box, it will generate the following command for the second list box, as well as refresh it.

```
SELECT *FROM APPCATEGORIES WHERE GROUPID=1
```

All this work is done by the list box controller; the only overhead required is that of line 18 in code listing 4.

This may all seem like a lot of work, so I have created a function that attempts to detect the HTML form, and generate the appropriate control models and views/controllers—this is the *detectForm(DataFormModel oFormModel,DataForm oFormController)* function. Code listing 2 can be replaced with the following:

```
1 var FormModel= new DataFormModel("FORM1","OOFORUMTESTS",0,"APPCATEGORIES");
2
3 var Form= new DataForm(FormModel);
4 try{ detectForm(FormModel,Form); }
5 catch(e){alert(e);}
6
7 Form.refresh();
```

Conclusion

This is just a preliminary experiment. However, it does show promise for linking HTML forms to a Base database. Furthermore, the HTML form can be generated by reading a base form, as it may be easier to create a nice looking for with the Base form builder.

A better, or more formal, approach may be to use an existing framework such as Dojo (<http://www.dojotoolkit.com/>). While I have not looked under the hood, it does look like a very robust toolkit.