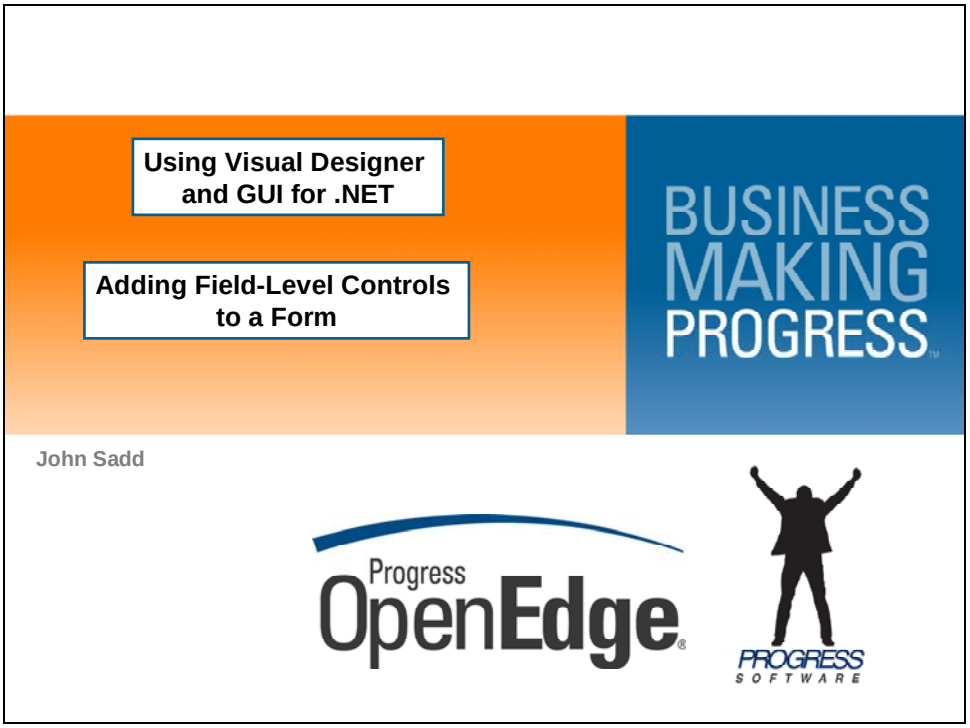
ADDING FIELD-LEVEL CONTROLS TO A FORM

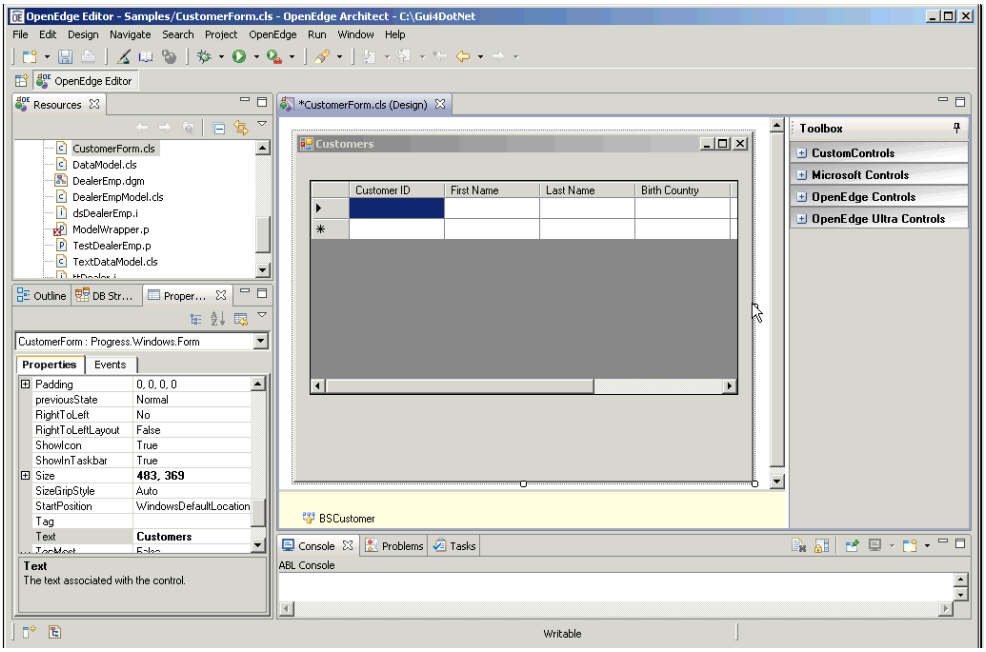
John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
November 2009



DISCLAIMER

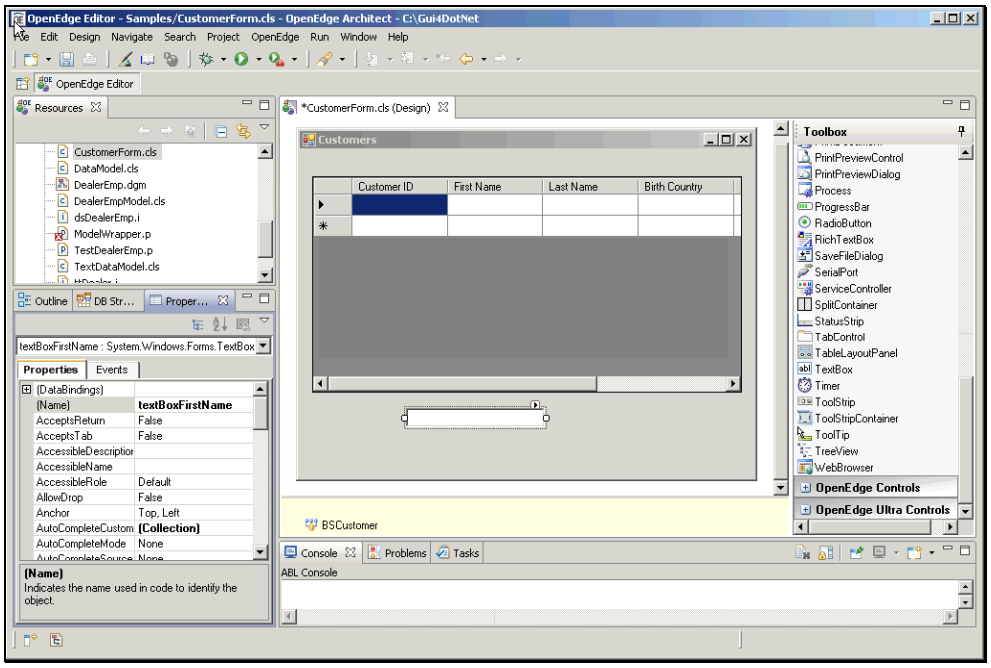
This presentation is part of a series on using Visual Designer and GUI for .NET in OpenEdge Architect. In this session I start with the Customer Form that was built in another session pair titled **Creating a Form and a ProBindingSource**, with its data grid and binding source, and add some field-level controls to it to show how to bind those controls to individual fields in the same binding source.

Opening the CustomerForm class, it contains a **ProBindingSource** control for the AutoEdge Customer table, and a Microsoft **DataGridView** control that displays all the data in the table. I first make room in the form and in the design view for the field controls that I want to add.



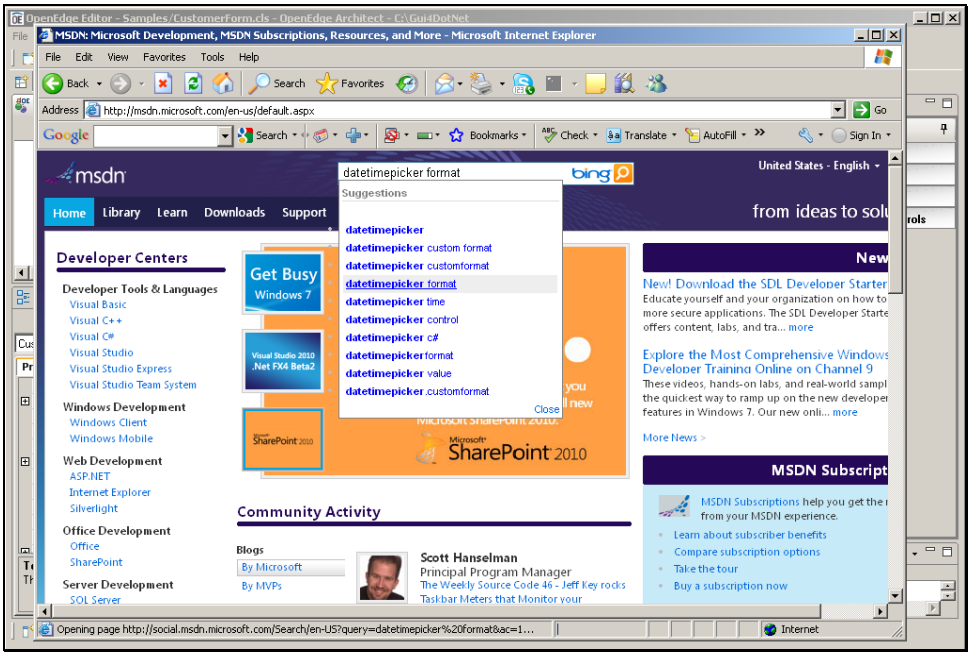
If I open the Microsoft Controls group you can see all the controls that are included with OpenEdge 10.2. The **MaskedTextBox** for instance is good for enforcing specific data entry formats; and here's a **RichTextBox** for displaying large amounts of data in an editor. I just select the standard **TextBox**, and drag one onto the form, and make an initial attempt at resizing it the way I want it.

Scrolling up through its properties in the **Properties** View to the left, I can give it a meaningful name; I'll display the Customer's First Name in this control; the **Name** property is the name of the variable in the generated code that will hold this control's object reference.

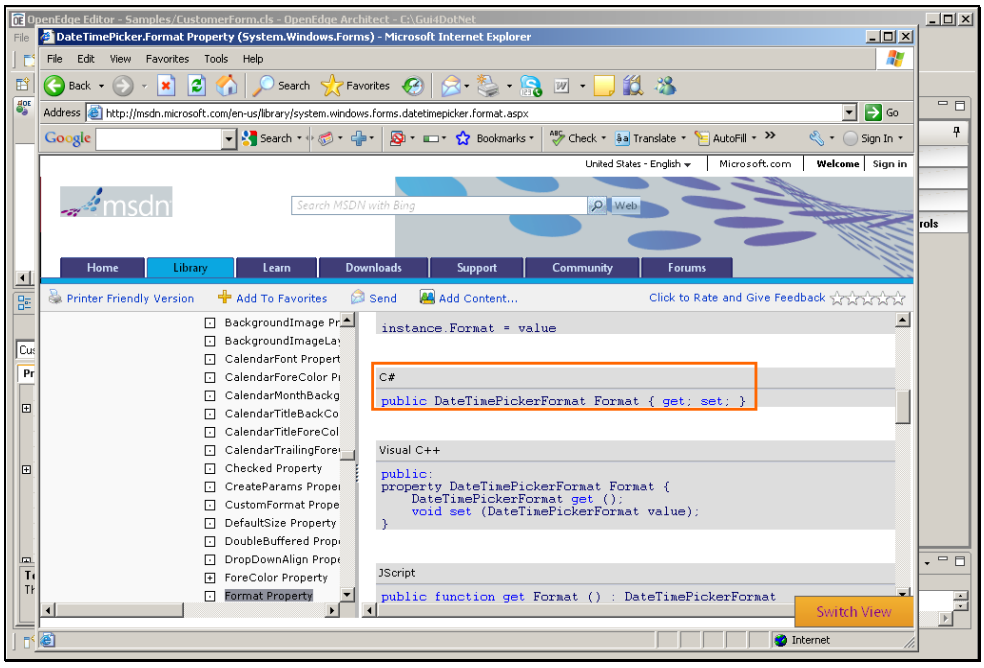


I then get another **TextBox** to display the Customer's Last Name, and once again give a meaningful name to the variable that will hold that reference. Next I use another control type, this one a calendar format that displays a date, called the **DateTimePicker**. I use this to display the Customer's Birthday. But I don't really care about displaying the day of the week, so I want to learn how to reformat the field into the format I want.

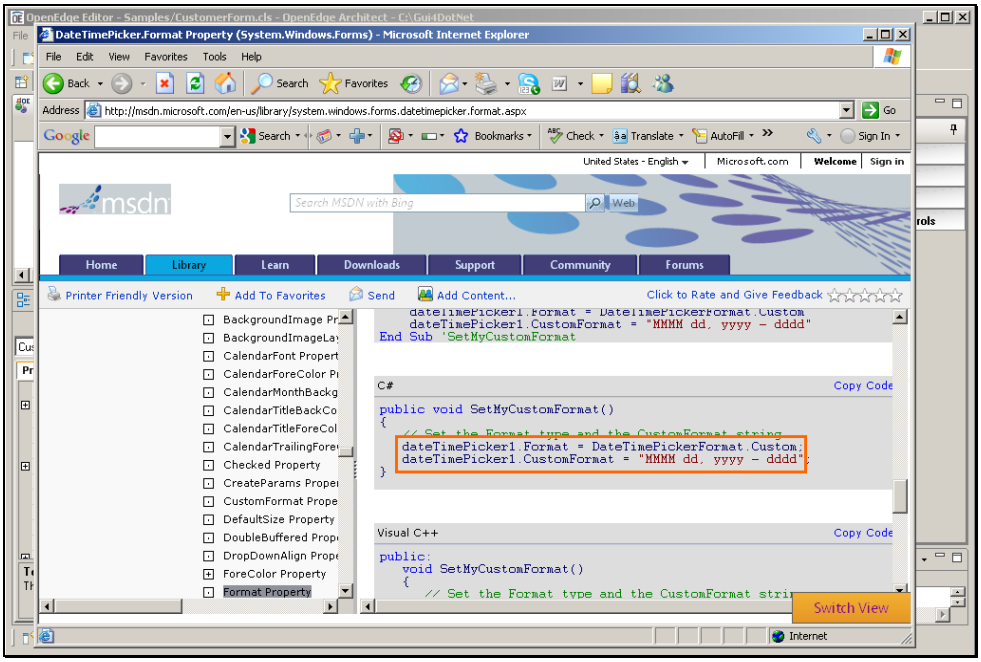
I can go back to MSDN to learn all about this control just like any other. Under **DateTimePicker**, I select *format* to direct me to a description of the control's format.



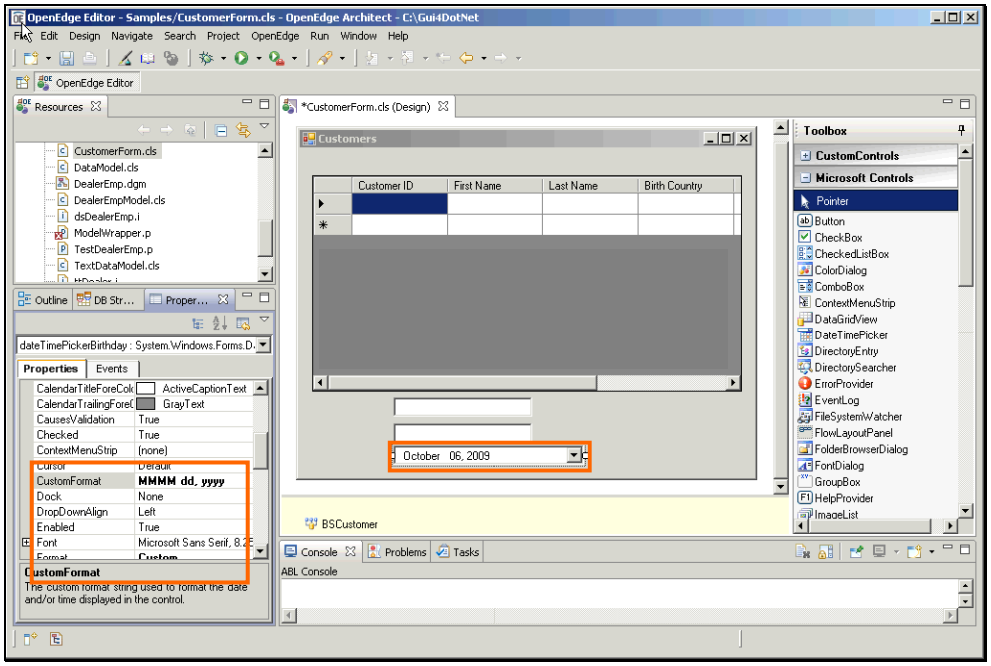
The first link in the list of options I'm shown leads me to the control's Format description. I see that its **Format** property is an object reference to a special **DateTimePickerFormat** object:



Looking further for a code example, I can see where the code sets the **Format** to the value **Custom**, and then the **CustomFormat** property to a string. This tells me what I need to know to set the **Format** in my own control.

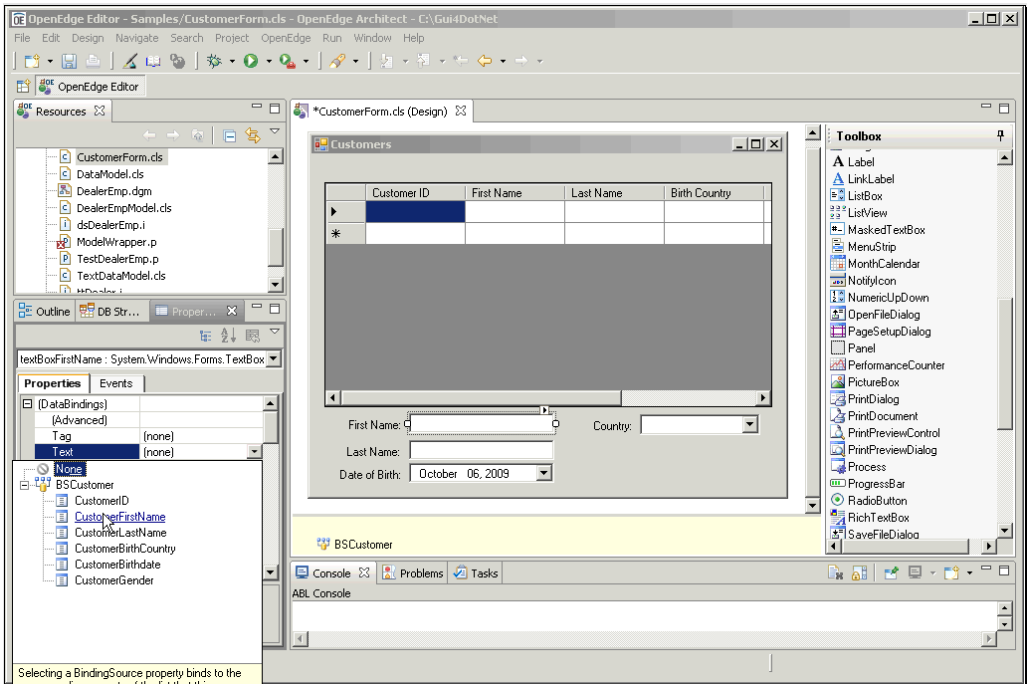


Back in Visual Designer, I find the control's **Format** property in the Properties View, and set that to the enumeration value **Custom**, and then find its **CustomFormat** property and indicate that I want to see the month name, day of the month, and four-digit year. Visual Designer redisplays today's date as a default value in the format I asked for, so I can resize the field to match.



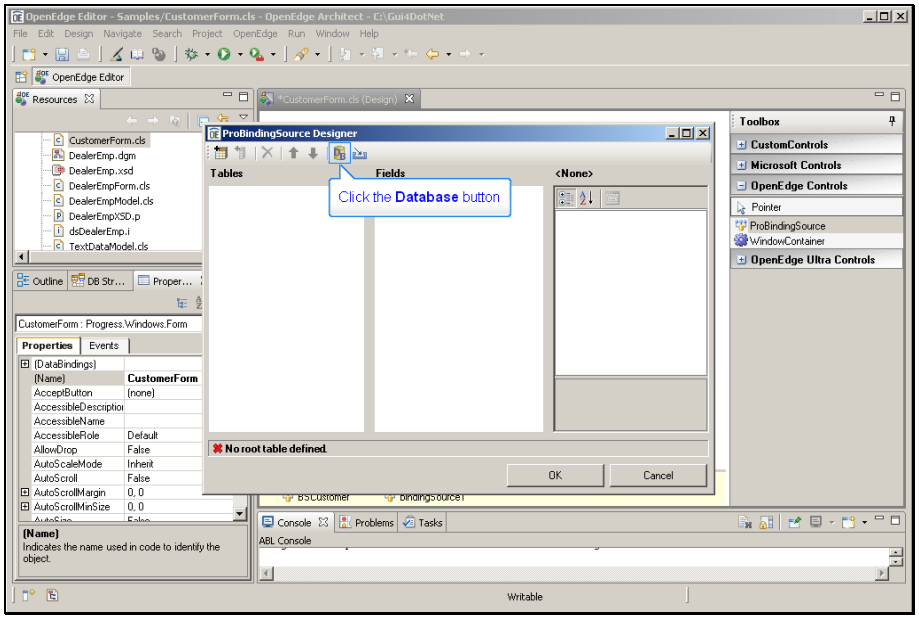
Next I need to get a **Label** control for each of the fields. Unlike working in AppBuilder, you don't get a label associated with a field automatically, so I drag a **Label** control onto the form and set its **Name** property. The **Text** property is the text that's displayed as the field's label, so I set that to **First Name:** in this case. And the **TextAlign** property determines exactly how the label is aligned with the field; I pick the enumeration value of **Middle Right**, so I can get the label right where I want it. After adding labels for the others fields, I add a **ComboBox** control to the form. I display the Customer's BirthCountry in this drop-down list control, and give it a label just as I did the other fields in the form. Once this is complete, I can get to the heart of this session, and associate each of my four fields with one of the fields in the Customer **ProBindingSource**, the same binding source that the grid uses, so that they display fields from the currently-selected row in the Customer table.

At the top of the property list for each control I find the **DataBindings** property. Under that the **Text** value lets me select the Customer binding source, and then the **CustomerFirstName** field as the one to bind to this control.

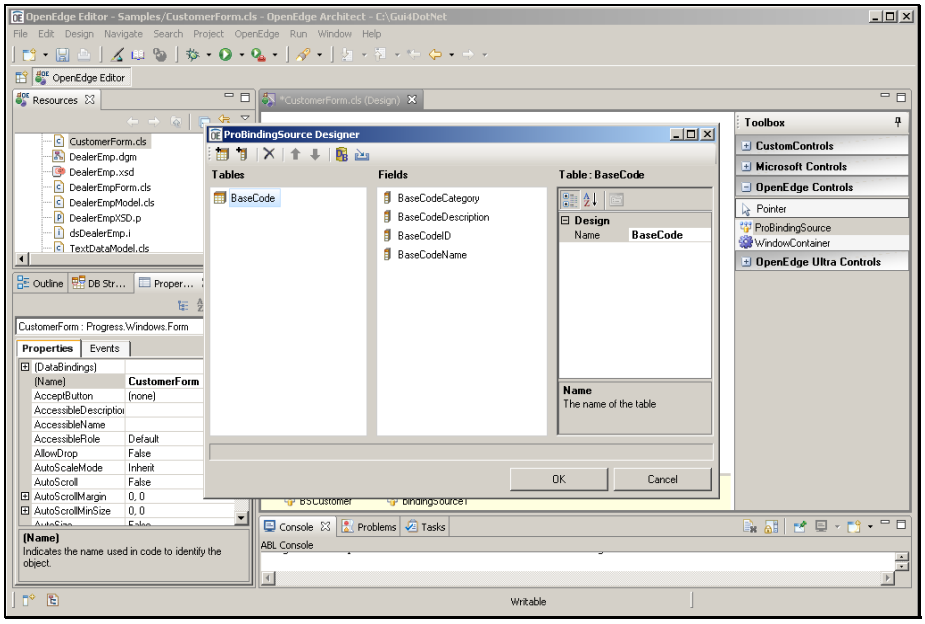


I do the same for the **CustomerLastName** field, selecting it from the binding source, and the Birthday field, selecting the CustomerBirthdate as the value to display. I do the same for the **Country ComboBox**. When I run the form you'll be able to see that because the field-level controls are bound to individual fields in the same binding source as the grid, the field values are refreshed automatically to show the fields in the current row.

To populate the values you see when the combo box is expanded, I can give it its own binding source. AutoEdge has a **BaseCode** table to hold code values of all kinds. The Country combo's values should come from the values in the AutoEdge BaseCode table where the code category is Country. I create that second binding source, dragging it onto the form. Now in the ProBindingSource Designer I select the option again to create binding source schema directly from a database table.

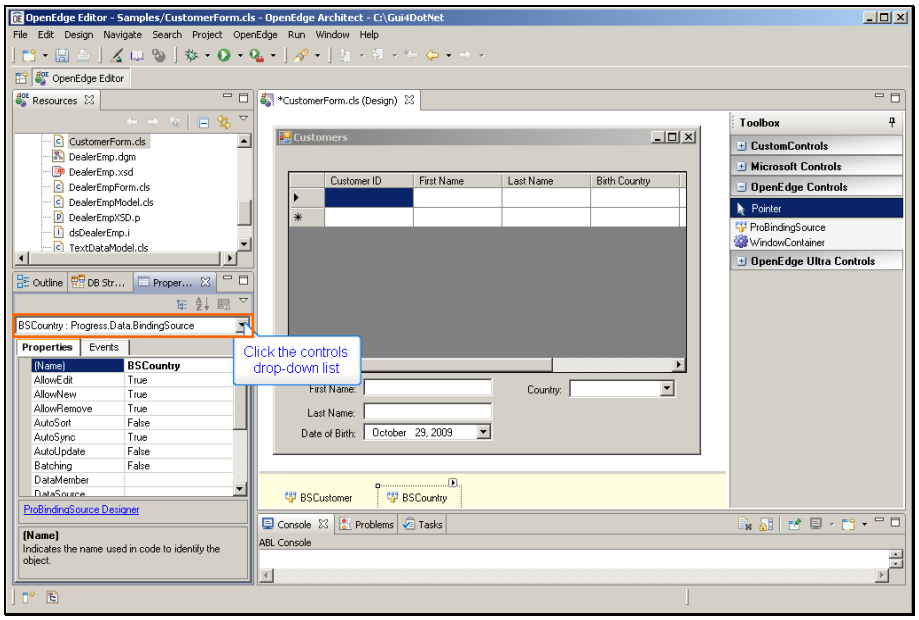


I select the BaseCode table as the data source. I don't need to make any changes to the field list for my purposes, so I'm all done.

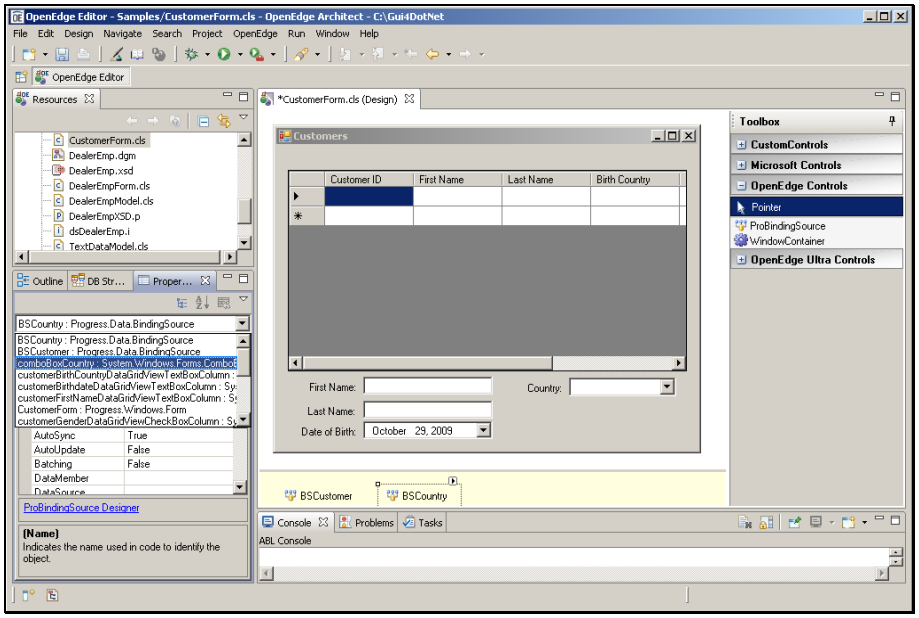


I give the binding source a name of **BSCountry**.

Let me just point out that you can select controls in the form from the drop-down list as shown here as well as selecting them visually.

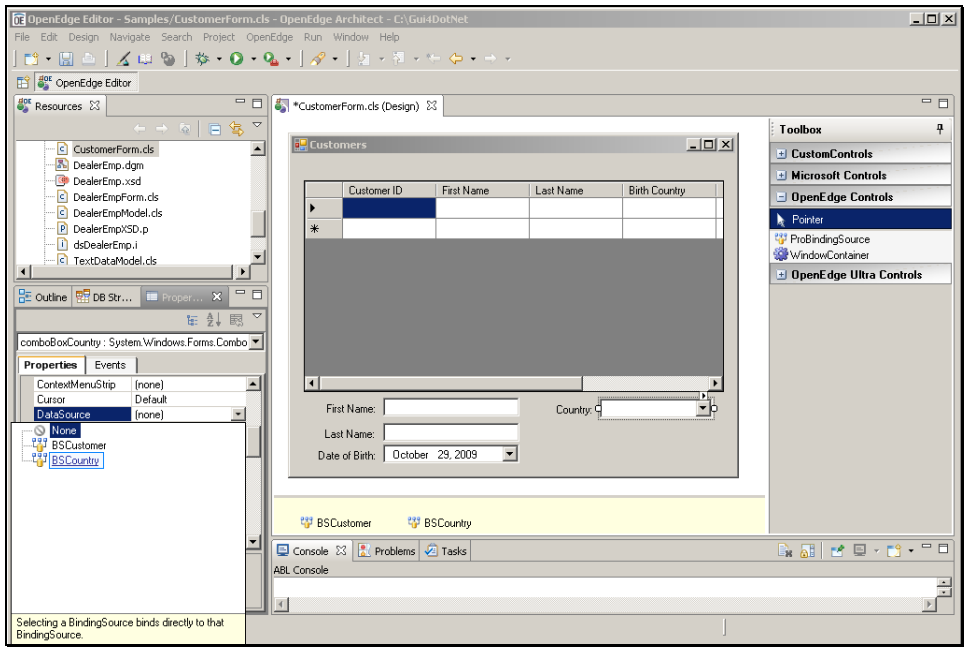


Here's the **Country ComboBox** control, for instance.

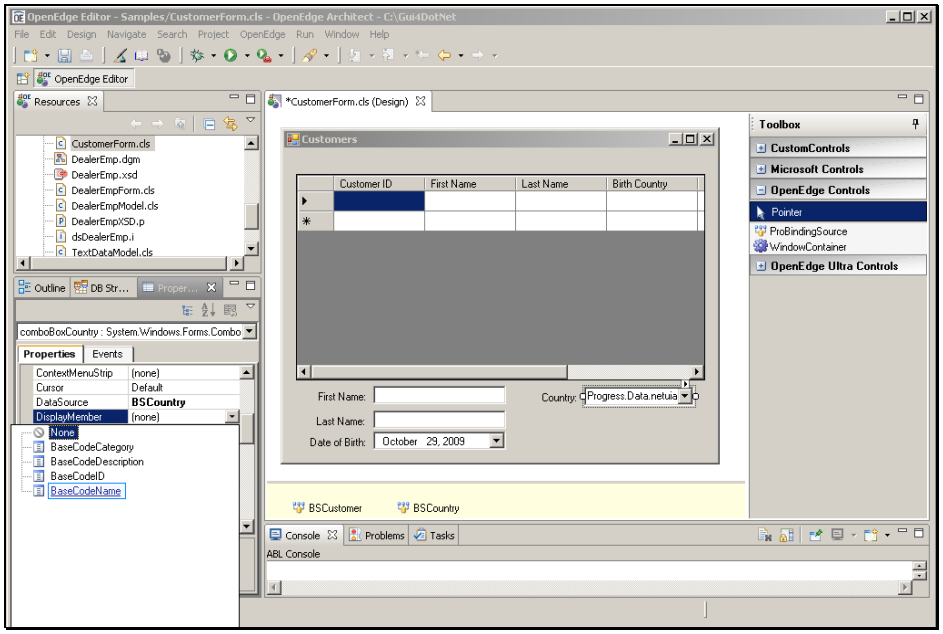


I can set the same properties here in the Properties View as I could by opening the control's **SmartTag**.

Here's the **DataSource** property, which I set to the new **BSCountry** binding source.



Below it is where I specify the field to display in the ComboBox. From the BaseCode field list I select the **BaseCodeName**.



The control has a **ValueMember** property where I specify what field the combo box should actually use as its value when I select an entry from the display list. In this case it's the same **BaseCodeName** field. In other cases it might be a key field or other coded value, of course. Having completed this, I switch back to the Editor to show you an alternative to the dynamic query I used for the Customer table.

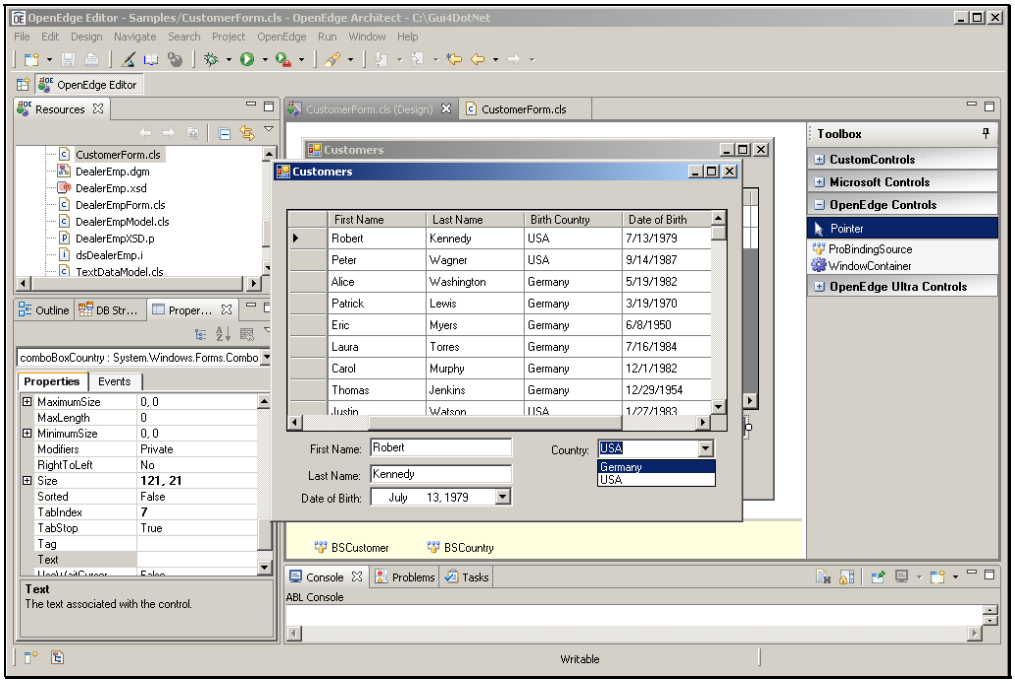
In the following line of code I define a query on the **BaseCode** table. Remember that every data member that's defined in the main block of the class, above its method definitions, can be public, private, or protected, and is public by default, so I can add the PRIVATE keyword to the definition. Also note that a query for a binding source needs to be SCROLLING. Dynamic queries are scrolling by default but for the static query I need to make it explicit.

```
DEFINE PRIVATE QUERY qBaseCode FOR BaseCode SCROLLING.
```

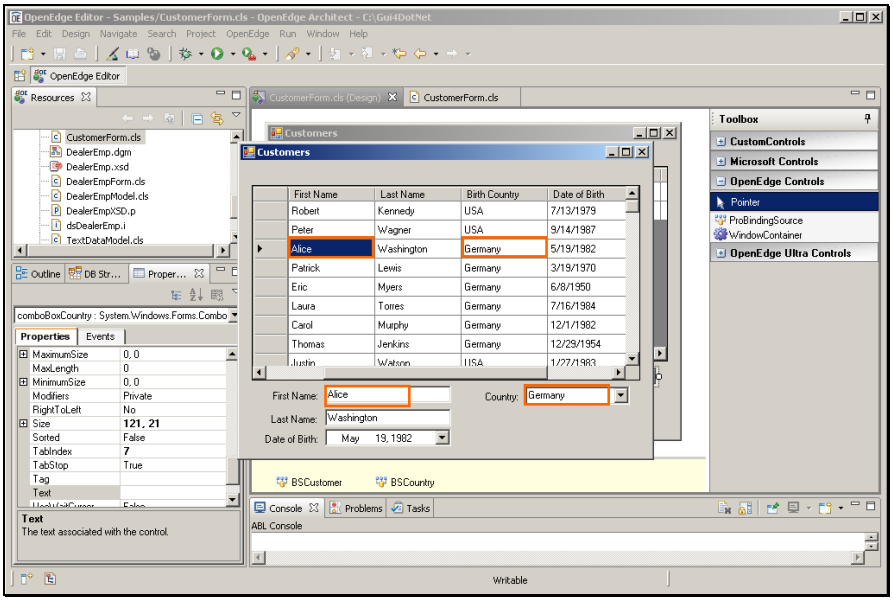
Down in the constructor I have statements to open the static BaseCode query and set the **Handle** property of the binding source that are equivalent to the dynamic statements I used for the Customer query. I'm only interested in those BaseCode rows where the category is Country.

```
CONSTRUCTOR PUBLIC CustomerForm (  ):

        SUPER().
        InitializeComponent().
        CREATE QUERY hCustQuery.
        hCustQuery:SET-BUFFERS(BUFFER Customer:HANDLE).
        hCustQuery:QUERY-PREPARE("FOR EACH Customer").
        hCustQuery:QUERY-OPEN ().
        BSCustomer:HANDLE = hCustQuery.

        OPEN QUERY qBaseCode FOR EACH BaseCode WHERE
            BaseCode.BaseCodeCategory = "Country".
        BSCountry:HANDLE = QUERY qBaseCode:HANDLE.

        CATCH e AS Progress.Lang.Error:
            UNDO, THROW e.
        END CATCH.

END CONSTRUCTOR.
```

I can save the form with my changes to see how it works now. If I run the form, scrolling over to see the fields that I'm interested in, and drop down the combo box, I see that there are just two countries in the BaseCode table for users to choose from when they enter new customers.



If I take a look at a few of the rows in the grid, you can see that the field-level controls are in sync with the grid because they're bound to the same binding source.



The value displayed in the Country combo box comes from the **BSCustomer** binding source; I made that association along with the rest of the fields. The values used to populate the combo's drop-down list come from a separate **BSCountry** binding source, on the BaseCode table.

This session has shown you how to add field-level controls to a form, provide them with labels, bind them to individual fields in a binding source, and populate a ComboBox with values from a data source of its own.