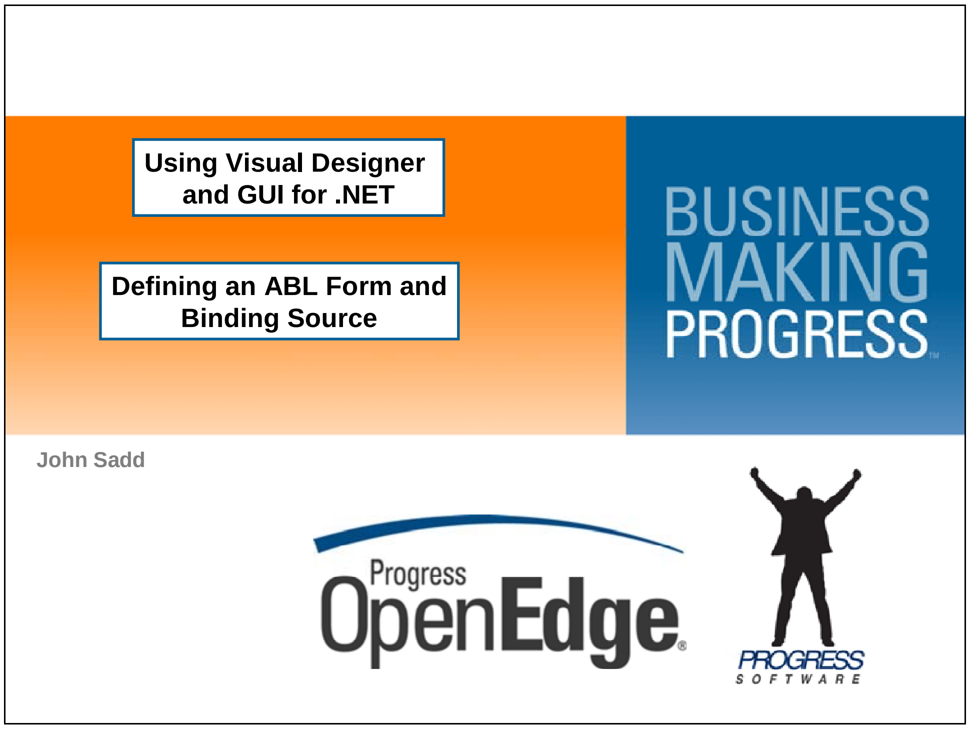
DEFINING AN ABL FORM AND BINDING SOURCE

John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
November 2009



Using Visual Designer and GUI for .NET

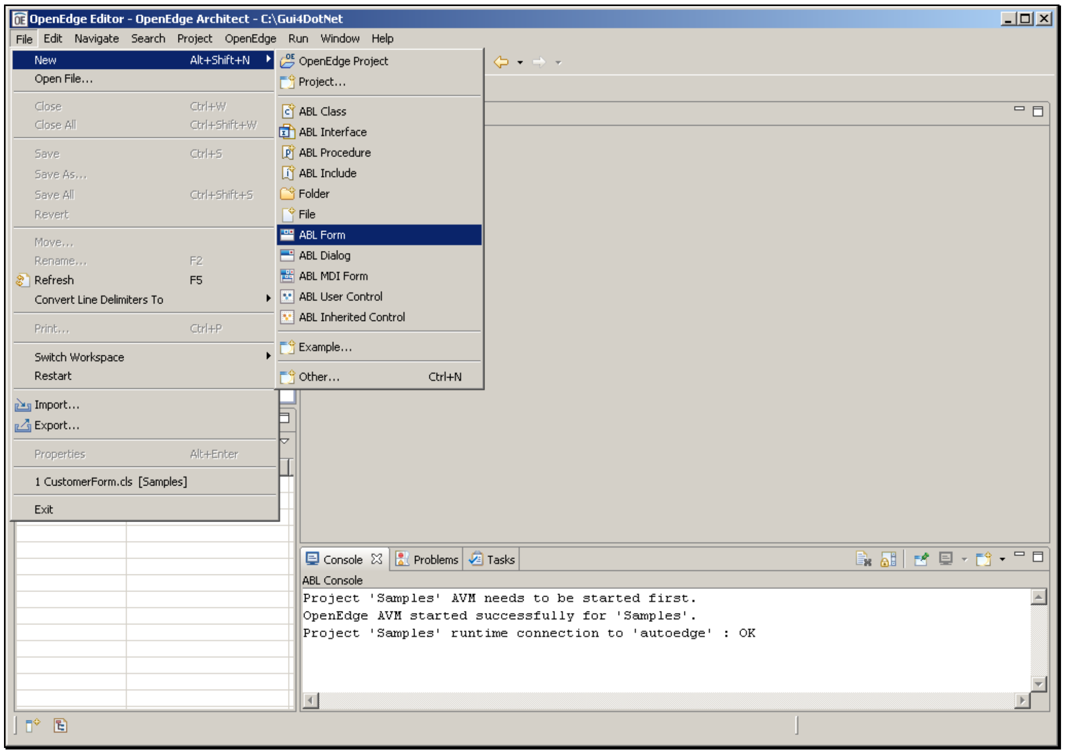
Defining an ABL Form and Binding Source

BUSINESS MAKING PROGRESS™
John Sadd
Progress OpenEdge®
PROGRESS SOFTWARE

DISCLAIMER

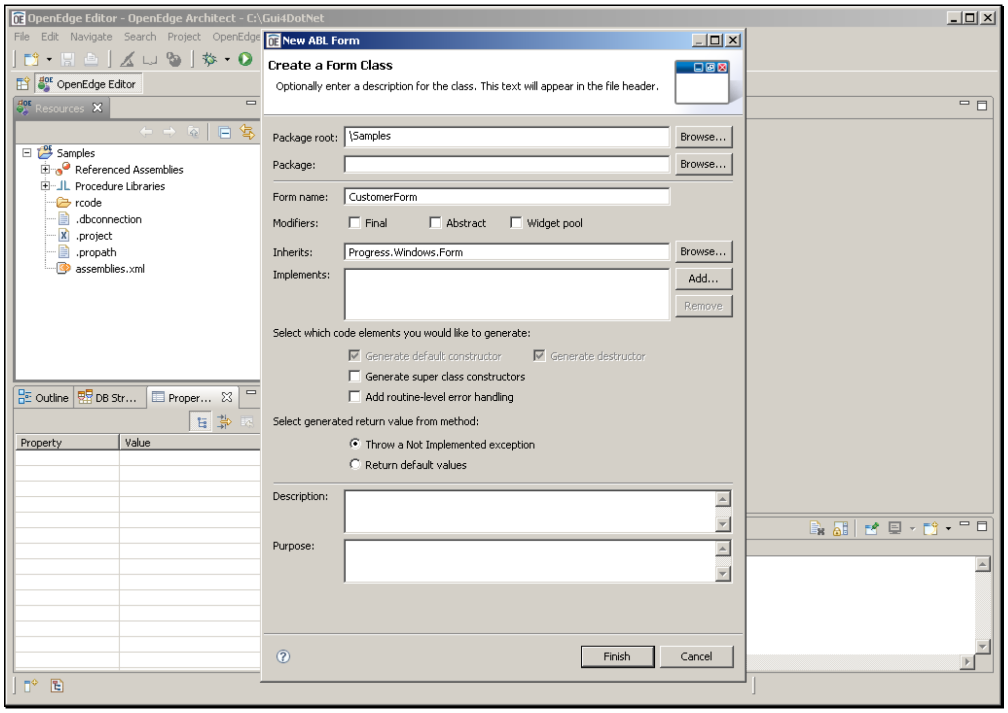
These notes accompany the first in a series of presentations to introduce you to the Visual Designer in OpenEdge Architect, and to the principles of using the support for GUI for .NET in OpenEdge 10 to provide a modern and compelling user interface for your ABL applications. This is the first of two sessions on **Creating a Form and a ProBindingSource**. In this first session, I create a new ABL Form, add a ProBindingSource control to it to provide data to the user interface, and take a look at the code that gets generated to support the form and its binding source.

Starting out in the **Editor Perspective** in Architect, I select the **File** menu and select **New ABL Form**. This creates a new ABL class file to support the creation of a form that I can then drop .NET controls onto, along with my own supporting ABL code.



```
Project 'Samples' AVM needs to be started first.
OpenEdge AVM started successfully for 'Samples'.
Project 'Samples' runtime connection to 'autoedge' : OK
```

I call the class **CustomerForm**, so the file name will be **CustomerForm.cls**. I can enter a **Description** for the form, which will appear in the comments section up at the top. And you can see there are other options for the initial code generation for the class that are left set to their defaults for now. I just click **Finish** to complete the wizard.



Create a Form Class

Optionally enter a description for the class. This text will appear in the file header.

Package root: \Samples

Package:

Form name: CustomerForm

Modifiers: ☐ Final ☐ Abstract ☐ Widget pool

Inherits: Progress.Windows.Form

Implements:

Select which code elements you would like to generate:

☑ Generate default constructor ☑ Generate destructor

☐ Generate super class constructors

☐ Add routine-level error handling

Select generated return value from method:

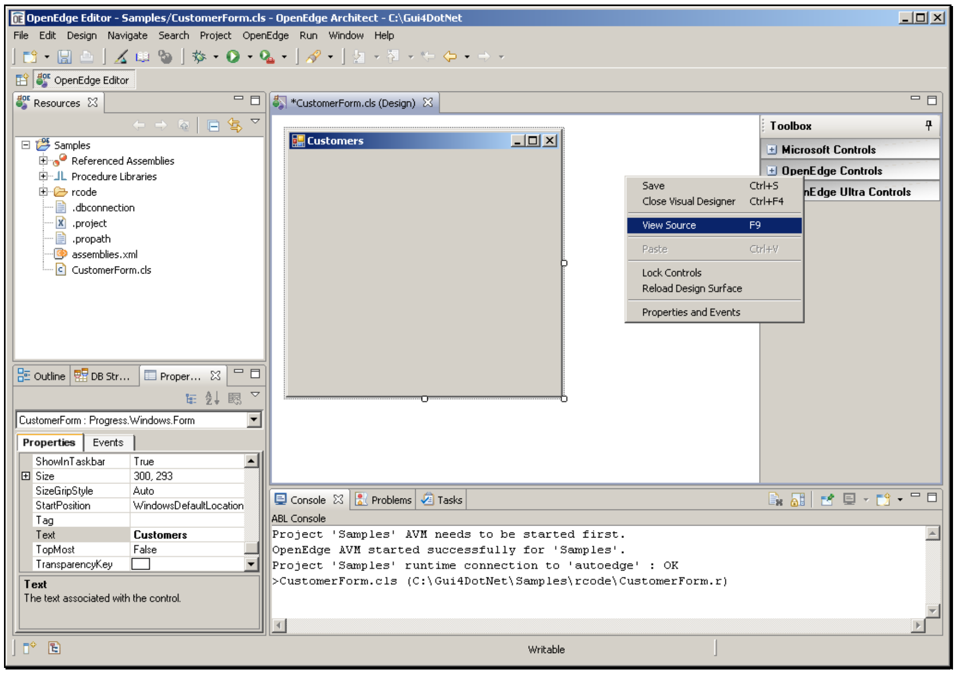
● Throw a Not Implemented exception

○ Return default values

Description:

Purpose:

Because I created a new Form, Architect automatically opens that form in **Visual Designer**, which has a visual design canvas for the form and any controls I drop onto it. In addition to manipulating controls visually, I can see and set their properties in the **Properties View**. I can change the form's **Text** property, which controls its title, and see that the change is reflected in the form as it's displayed. But I can also switch back and forth between the design and the code that is being generated to create the form at runtime, by pressing **F9** or bringing up the context menu and selecting **View Source**.



CustomerForm : Progress.Windows.Form

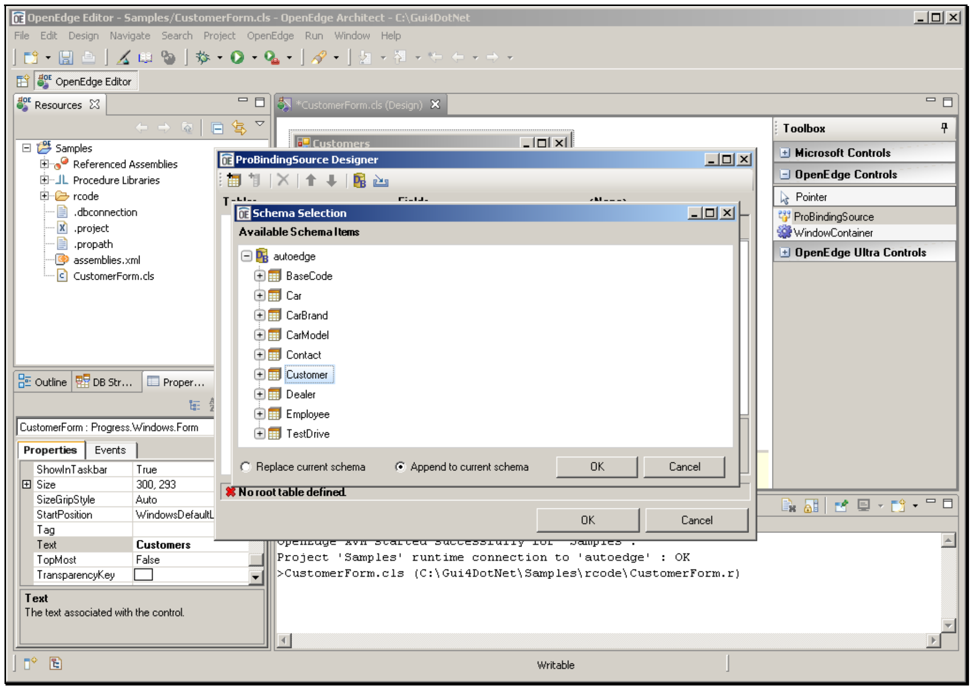
Properties	Events
ShowInTaskbar	True
Size	300, 293
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	**Customers**
TopMost	False
TransparencyKey	

Text
The text associated with the control.

ABL Console

```
Project 'Samples' AVM needs to be started first.
OpenEdge AVM started successfully for 'Samples'.
Project 'Samples' runtime connection to 'autoedge' : OK
>CustomerForm.cls (C:\Gui4DotNet\Samples\rcode\CustomerForm.r)
```

The first control I add to the form is a non-visual control to manage data for visual controls that can be added later on. It’s one of the built-in controls created specifically as part of OpenEdge to support ABL applications, so I expand the list of OpenEdge controls and select the **ProBindingSource**, dragging it onto the form. As soon as I drop the control onto the form, the **ProBindingSource Designer** opens, so that I can define the data I want the control to manage. I can get that data definition from an XSD file – an XML schema definition file that I could create very easily from ABL, to correspond to a temp-table or ProDataSet definition for instance. Or I can select the DB icon and create schema for the binding source directly from a connected database. In this case I have the AutoEdge sample database connected, so I can expand that selection and pick the Customer table as the one I want the control to manage, and then click OK.



No root table defined.

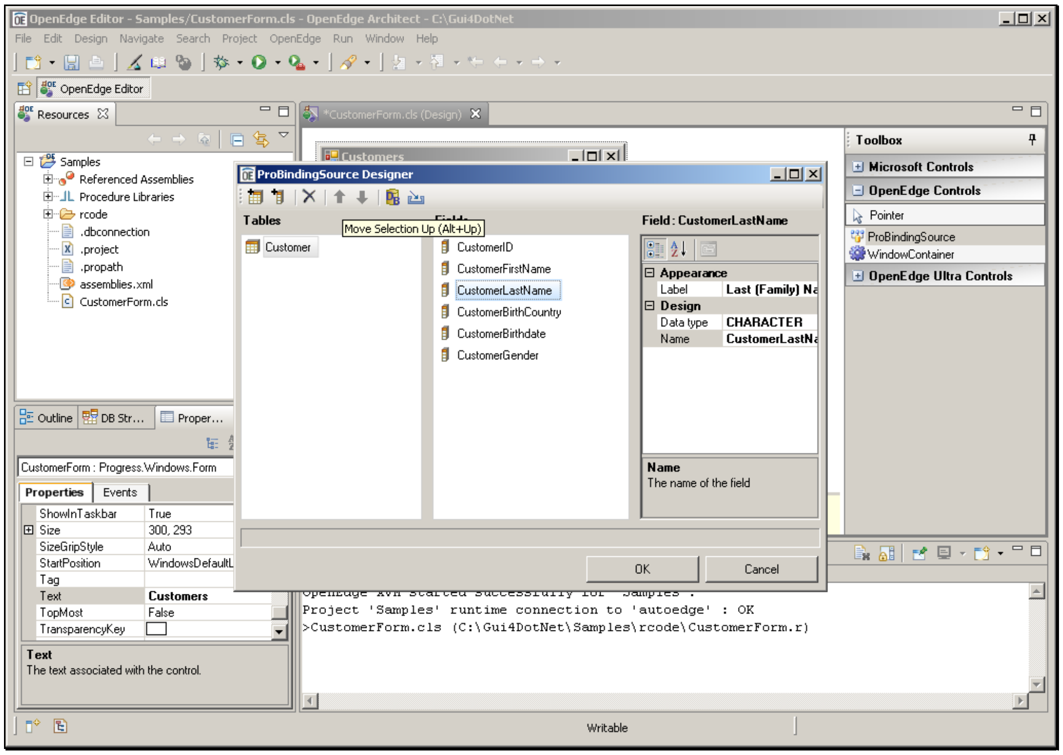
CustomerForm : Progress.Windows.Form

ShowInTaskbar	True
Size	300, 293
SizeGripStyle	Auto
StartPosition	WindowsDefaultL
Tag	
Text	Customers
TopMost	False
TransparencyKey	

Text
The text associated with the control.

```
OpenEdge AVM started successfully for 'Samples'.
Project 'Samples' runtime connection to 'autoedge' : OK
>CustomerForm.cls (C:\Gui4DotNet\Samples\rcode\CustomerForm.r)
```

If in fact I don't want to use all the fields in the table, I can delete the ones that the form won't need to display. In the Designer I can also change the order of fields so that the default display order for fields in a grid control, for instance, will be the way I want them, just by selecting a field and then the **Move Up** or **Move Down** buttons.



Appearance	
Label	Last (Family) Na
Design	
Data type	CHARACTER
Name	CustomerLastNa

Name
The name of the field

CustomerForm : Progress.Windows.Form

ShowInTaskbar	True
Size	300, 293
SizeGripStyle	Auto
StartPosition	WindowsDefaultL
Tag	
Text	Customers
TopMost	False
TransparencyKey	

Text
The text associated with the control.

```
OpenEdge AVM started successfully for 'Samples'.
Project 'Samples' runtime connection to 'autoedge' : OK
>CustomerForm.cls (C:\Gui4DotNet\Samples\rcode\CustomerForm.r)
```

Now my binding source is set up to manage fields from the Customer table in the database and make them available to any visual controls I add to the form that are "data bound", that is, controls that have properties to display data from the binding source. So this is a very powerful feature of our support for GUI for .NET, that a standard data source property of many visual controls is satisfied by a specialized control provided as part of OpenEdge.

Reviewing the code that results from creating the form and dropping a ProBindingSource onto it, you first see a comment section at the top with the file name, description, and other information.

```
/*------------------------------------------------------------------------
    File        : CustomerForm
    Purpose     :
    Syntax      :
    Description : Simple form to display Customers in a grid
    Author(s)   : john
    Created     : Wed Aug 19 09:11:36 EDT 2009
    Notes       :
  ----------------------------------------------------------------------*/
```

The class inherits from the top-level **Progress.Windows.Form** class. The USING statement identifies the package the Form class is in so that it can then be referenced by its simple name.

```
USING Progress.Lang.*.
USING Progress.Windows.Form.

CLASS CustomerForm INHERITS Form   :
```

Next is a variable definition for the binding source instance. This is the part of the generated code that's outside of the **InitializeComponent** method, so be careful not to change or delete these lines in the main block of the class.

```
    DEFINE PRIVATE VARIABLE BSCustomer AS Progress.Data.BindingSource NO-UNDO.
```

Visual Designer defines a variable named **components** which acts as a collection for some kinds of controls that it wants to make sure get disposed of properly when the form exits; there's no need to understand more than just to leave the definition alone when you start adding your own code to the form.

```
   DEFINE PRIVATE VARIABLE components AS System.ComponentModel.IContainer NO-UNDO.
```

There's a default constructor that runs the method **InitializeComponent**, and default error handling code that reflects some of the options in the New Form wizard.

```
        CONSTRUCTOR PUBLIC CustomerForm (  ):

         SUPER().
         InitializeComponent().
         CATCH e AS Progress.Lang.Error:
             UNDO, THROW e.
         END CATCH.
        END CONSTRUCTOR.
```

InitializeComponent is where all the generated code goes that describes the form and its controls, and Visual designer relies on this to be able to reconstruct the form when you're editing it. As the comment notes, you shouldn't edit this method directly. Your own code can go in other parts of the class file.

```
        METHOD PRIVATE VOID InitializeComponent(  ):
         /* NOTE: The following method is automatically generated.
            We strongly suggest that the contents of this method only be modified
             using the Visual Designer to avoid any incompatible modifications.
            Modifying the contents of this method using a code editor will
            invalidate any support for this file. */
```

Annotations such as this one provide design time information for Visual Designer, and should not be touched:

```
         @VisualDesigner.FormMember (NeedsInitialize="true").
```

The **TableDesc** and **ColumnPropDesc** objects are used by Visual Designer to identify at design time what tables and fields the binding source manages. They don't really affect behavior at runtime but they're needed when you add controls to the form at design time that use fields in the binding source, so like all the code in this method, be careful not to touch it:

```
         DEFINE VARIABLE tableDesc1 AS Progress.Data.TableDesc NO-UNDO.
         tableDesc1 = NEW Progress.Data.TableDesc("Customer").
```

And here is the **NEW** statement that creates an instance of the binding source.

```
         THIS-OBJECT:BSCustomer = NEW Progress.Data.BindingSource().
```

Similar to **SuspendLayout** and **ResumeLayout** for the form, there are methods called **BeginInit** and **EndInit** defined in the **ISupportInitialize** interface that let the code set all of a control's properties before it's displayed.

```
        CAST(THIS-OBJECT:BSCustomer,
          System.ComponentModel.ISupportInitialize):BeginInit().
```

The **SuspendLayout** and **ResumeLayout** methods are a standard convention to prevent the controls from being displayed when they're in the middle of being initialized.

```
        THIS-OBJECT:SuspendLayout().
```

Here you can see property settings for three ProBindingSource properties that have initial values. Two of them appear in the **Properties** View. The third one, **Position**, is a good example of another important thing to understand about controls. Part of the design of a control is determining which properties even get displayed in the Properties View, and in this case the control developer has decided that Position is not one of them. So some properties that a control supports may not appear in the Properties View, perhaps because the developer does not expect them to be set by the user. Some may have default values; some may have initial values that are explicitly set and so result in these lines of code being generated. Visual Designer queries each control at design time to determine both what code to generate and what properties to expose for use.

```
        /*  */
        /* BSCustomer */
        /*  */
        THIS-OBJECT:BSCustomer:MaxDataGuess = 0.
        THIS-OBJECT:BSCustomer:NoLOBs = FALSE.
        THIS-OBJECT:BSCustomer:Position = 0.
```

Next comes more code used at design time to define the database fields that become columns in the ProBindingSource schema.

```
        @VisualDesigner.FormMember (NeedsInitialize="false",
InitializeArray="true").
        DEFINE VARIABLE arrayvar0 AS "Progress.Data.TableDesc[]" NO-UNDO.
        arrayvar0 = NEW "Progress.Data.TableDesc[]"(0).
        tableDesc1:ChildTables = arrayvar0.
        @VisualDesigner.FormMember (NeedsInitialize="false",
InitializeArray="true").
        DEFINE VARIABLE arrayvar1 AS Progress.Data.ColumnPropDesc EXTENT 6 NO-UNDO.
        arrayvar1[1] = NEW Progress.Data.ColumnPropDesc("CustomerID", "Customer
ID", Progress.Data.DataType:CHARACTER).
        arrayvar1[2] = NEW Progress.Data.ColumnPropDesc("CustomerFirstName", "First
(Given) Name", Progress.Data.DataType:CHARACTER).
        arrayvar1[3] = NEW Progress.Data.ColumnPropDesc("CustomerLastName", "Last
(Family) Name", Progress.Data.DataType:CHARACTER).
        arrayvar1[4] = NEW Progress.Data.ColumnPropDesc("CustomerBirthCountry",
"Birth Country", Progress.Data.DataType:CHARACTER).
        arrayvar1[5] = NEW Progress.Data.ColumnPropDesc("CustomerBirthdate", "Date
of Birth", Progress.Data.DataType:DATE).
        arrayvar1[6] = NEW Progress.Data.ColumnPropDesc("CustomerGender", "Gender",
Progress.Data.DataType:LOGICAL).
        tableDesc1:Columns = arrayvar1.
        THIS-OBJECT:BSCustomer:TableSchema = tableDesc1.
```

The code sets the **ClientSize** property to determine the size of the form, the **Name** property, which corresponds to the class file name, and the **Text** property for the form's title.

```
        /*   */
        /* CustomerForm */
        /*   */
        THIS-OBJECT:ClientSize = NEW System.Drawing.Size(292, 266).
        THIS-OBJECT:Name = "CustomerForm".
        THIS-OBJECT:Text = "Customers".
```

And here at the end is the **EndInit** method that signals that initialization of the control is complete.

```
        CAST(THIS-OBJECT:BSCustomer,
              System.ComponentModel.ISupportInitialize):EndInit().
        THIS-OBJECT:ResumeLayout(FALSE).
        CATCH e AS Progress.Lang.Error:
            UNDO, THROW e.
        END CATCH.
END METHOD.
```

There's also a destructor that cleans up the components object.

```

       DESTRUCTOR PUBLIC CustomerForm ( ):
               IF VALID-OBJECT(components) THEN DO:
                       CAST(components, System.IDisposable):Dispose().
               END.
       END DESTRUCTOR.
END CLASS.
```

This completes the presentation on creating a form and adding a ProBindingSource to it. In the next session on Adding a Grid Control to a Form, you'll see how easy it is to attach your binding source to the controls that display and update the data in your form.