

## Developing a C# IVI-COM Client

### REVISION HISTORY

Date	Description	Revision Letter
02/20/2003	Original Draft	A

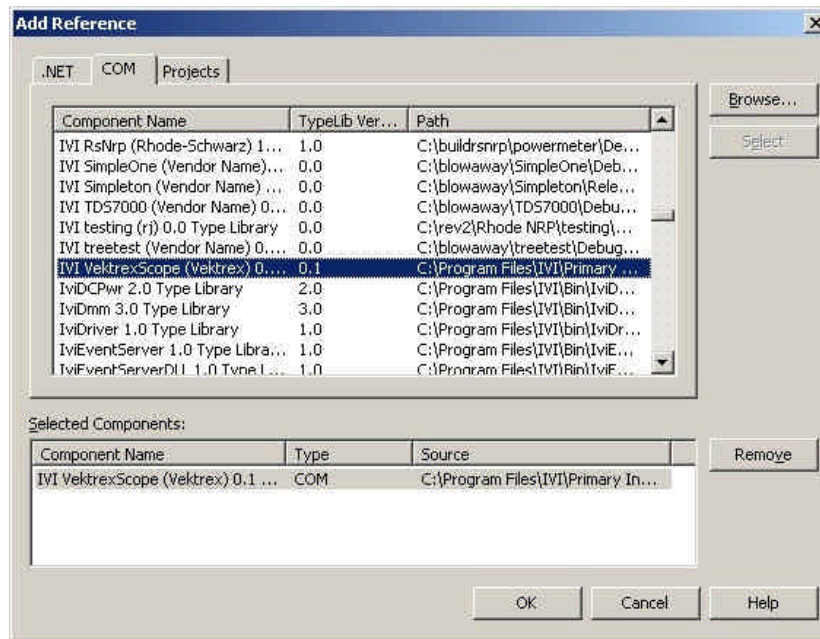
### Abstract

This application note describes the five steps required to access an IVI-COM driver from a C# client. The scope of this example is limited to building a simple project and the Vektrex Scope driver is used as an example. Specifically, this example shows how to create a client application using instrument specific interfaces in C#, but does not use interchangeability features (it does not use compliant interfaces and it does reference a particular driver, VektrexScope, directly).

The driver used in the examples in this application note (VektrexScope.dll) is available <http://www.vektrex.com/Products/VektrexScope.dll>.

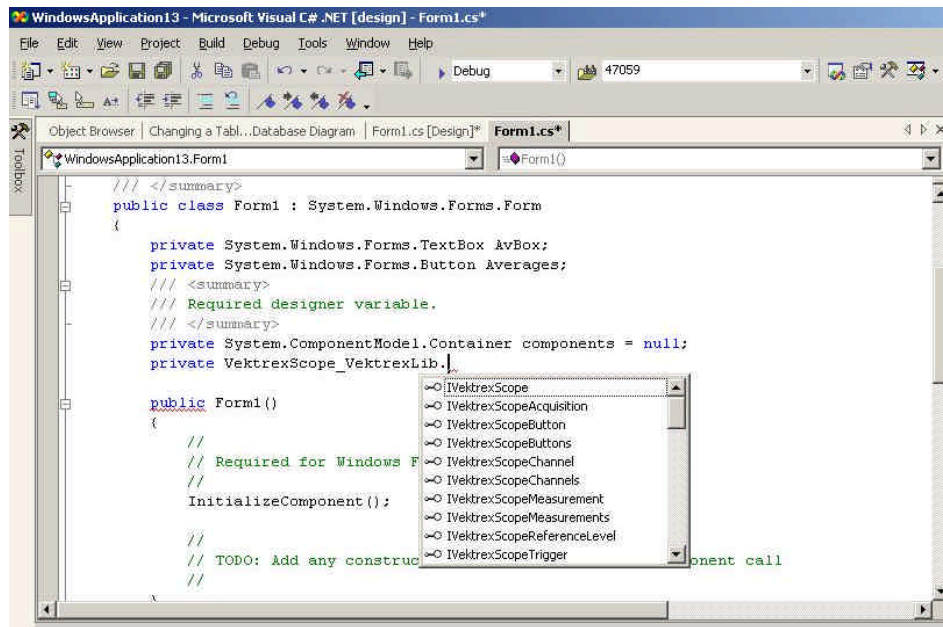
### Step 1: Make the COM object available to the Project

Upon creation of a new project, select **Add References...** from the **Project** menu. A dialog similar to that shown below appears. Select the **COM** tab, to provide a box listing all the registered COM objects on the computer. Select the **IVI vektrex scope (Vektrex) 0.1 Type Library**, and then click **Select**. The chosen COM object now appears in the Selected Components window. Click **OK** to save and exit.

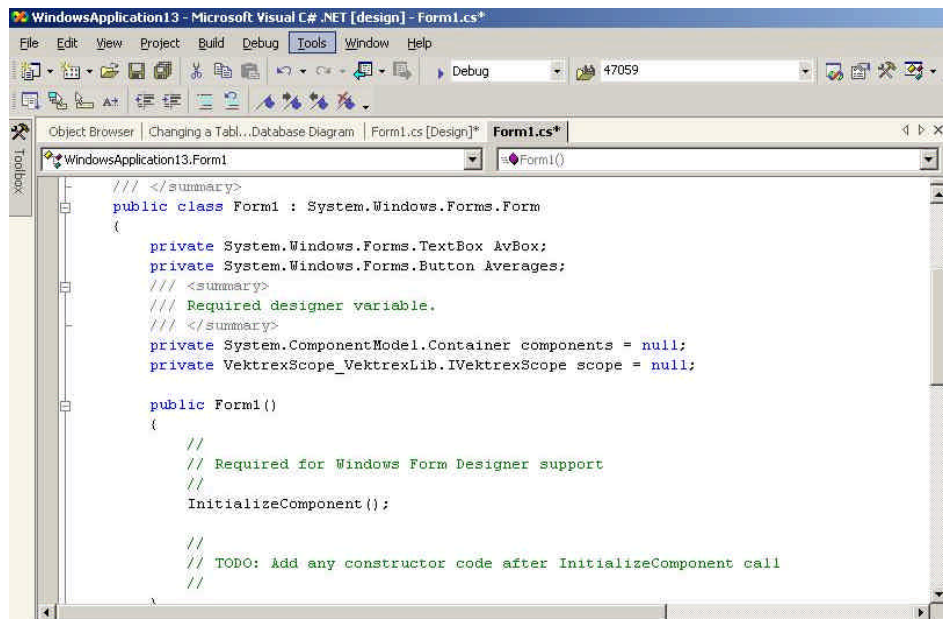


## Step 2: Create an Instance of the Object

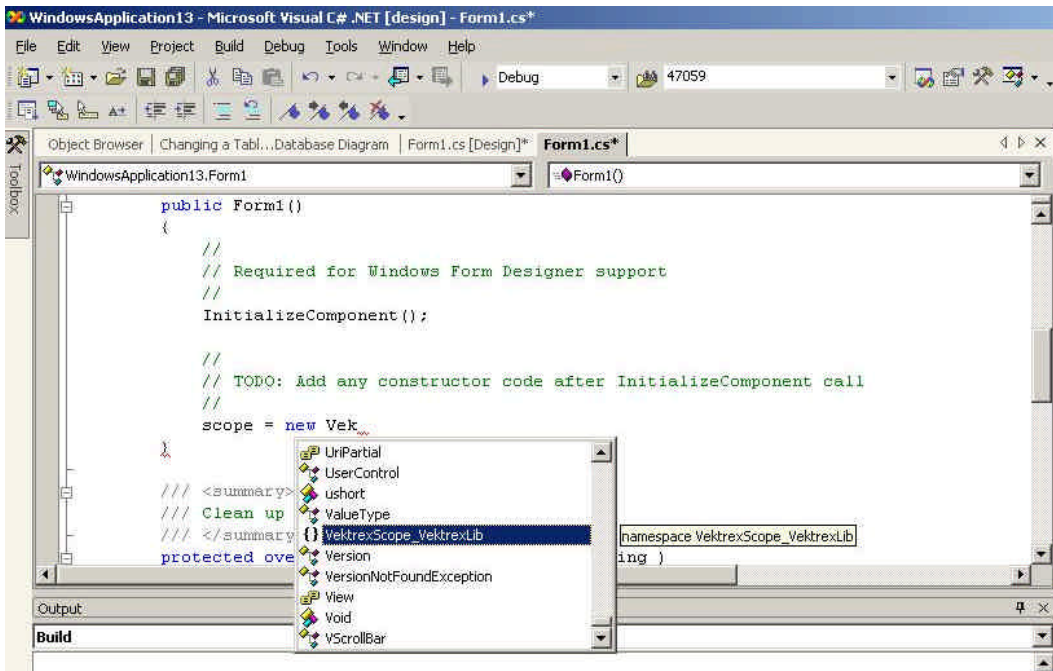
The proper location to declare the driver object is in the header file of the class for the Form (shown below), and is located beneath the comment **Required designer variable** and directly follows the line `private System.ComponentModel.Container components = null`. To declare the variable that references the scope object, type: **“private VektrexScope\_VektrexLib.”**. After typing the period, an Intellisense window appears. From this window, select **IvektrexScope**.



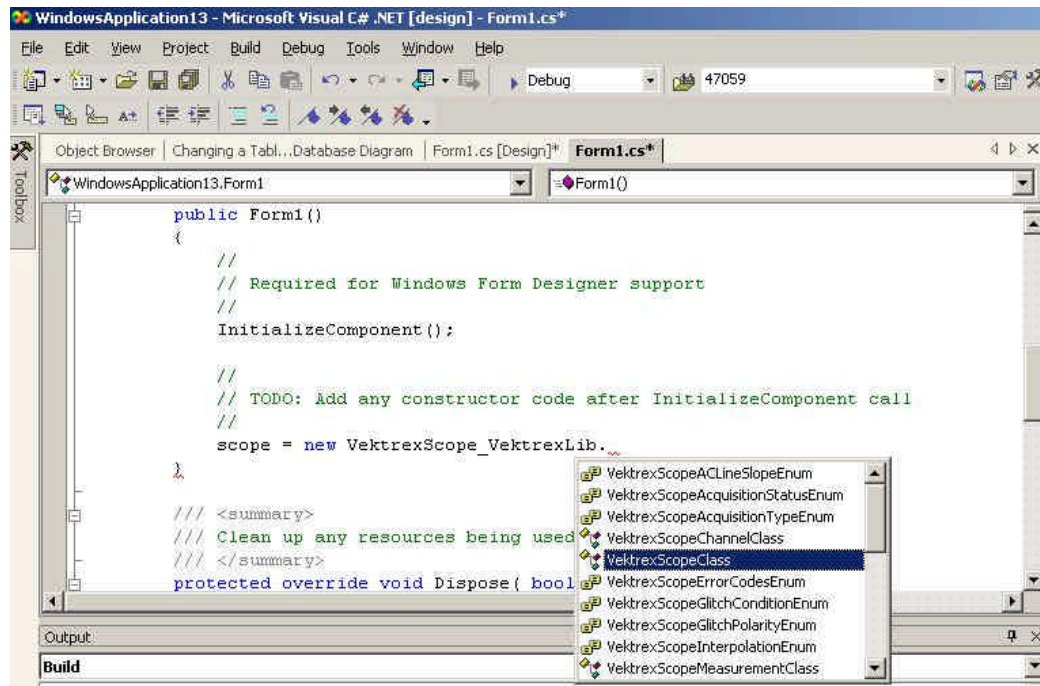
Next, click the **space bar** and then type: **“scope = null;”** The following figure depicts the declared driver object.



To create the instance of the object, move the cursor underneath the comment **TODO: Add any constructor code after...**, and type: **“scope = new Vek”** You should now see an Intellisense window with **VektrexScope\_VektrexLib** highlighted, as shown below.



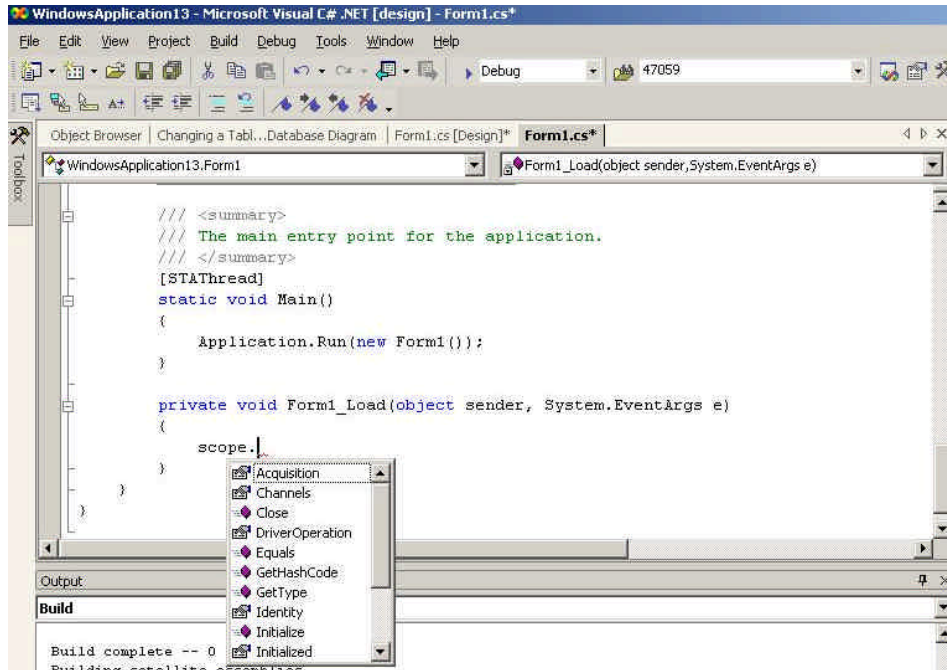
Now type a period, and a new Intellisense menu appears as shown below:



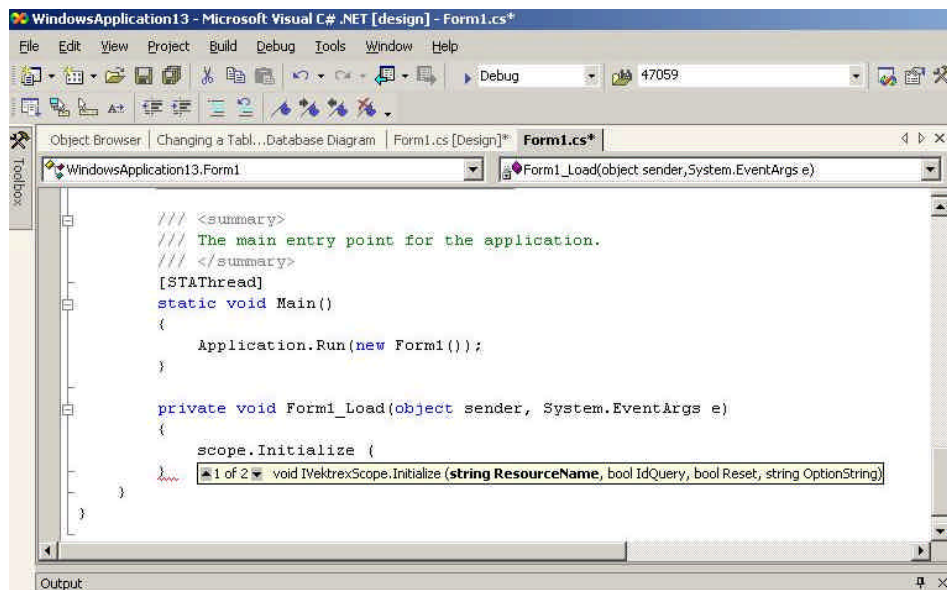
Finally, select the **VektrexScopeClass** and then type: **“(;)”**. This completes step two.

### Step 3: The Form\_Load Procedure

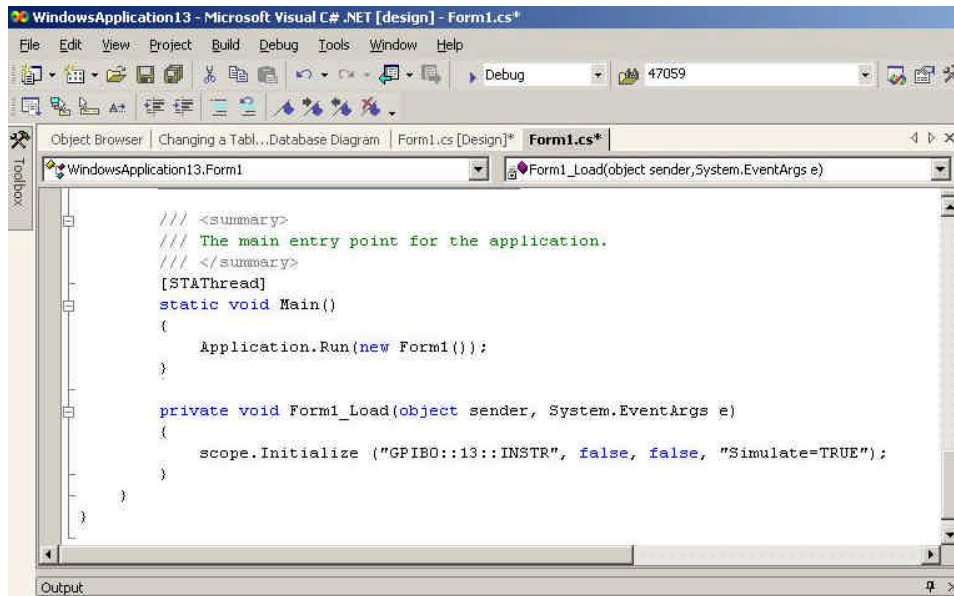
When a C# program runs, the Form\_Load procedure executes as part of the application's initialization routine. Hence, the Form\_Load procedure is an excellent place to locate the call to the driver's initialize function. The figure below shows the Form\_Load procedure, and it shows the developer starting to enter code that will call the initialize function.



Next, type: “**scope.Initialize (** “ and then press the spacebar. The development environment displays a tooltip, as shown below.



The following figure displays the completed Form\_Load procedure.

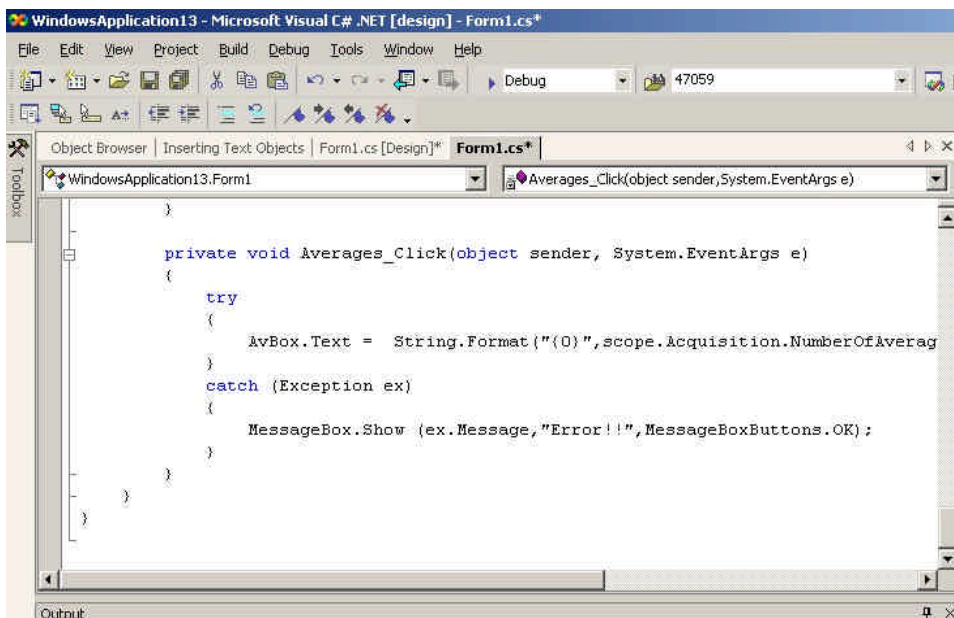


```
WindowsApplication13 - Microsoft Visual C# .NET [design] - Form1.cs*
File Edit View Project Build Debug Tools Window Help
Debug 47059
Object Browser Changing a Tabl...Database Diagram Form1.cs [Design]* Form1.cs*
WindowsApplication13.Form1 Form1_Load(object sender, System.EventArgs e)
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
    scope.Initialize ("GPIB0::13::INSTR", false, false, "Simulate=TRUE");
}
}
Output
```

#### Step 4: Code a function with Error Handling

The figure below shows a completed C# procedure utilizing a property in the instrument-specific acquisition interface. Again, Intellisense helps with each step, listing options for any enumerated types. This code assumes the C# form contains a button named Averages, and a textbox named AvBox that displays the returned value. The code also shows how to include error handling. Error handling in C# follows the same pattern as C++ (i.e., try/catch blocks).

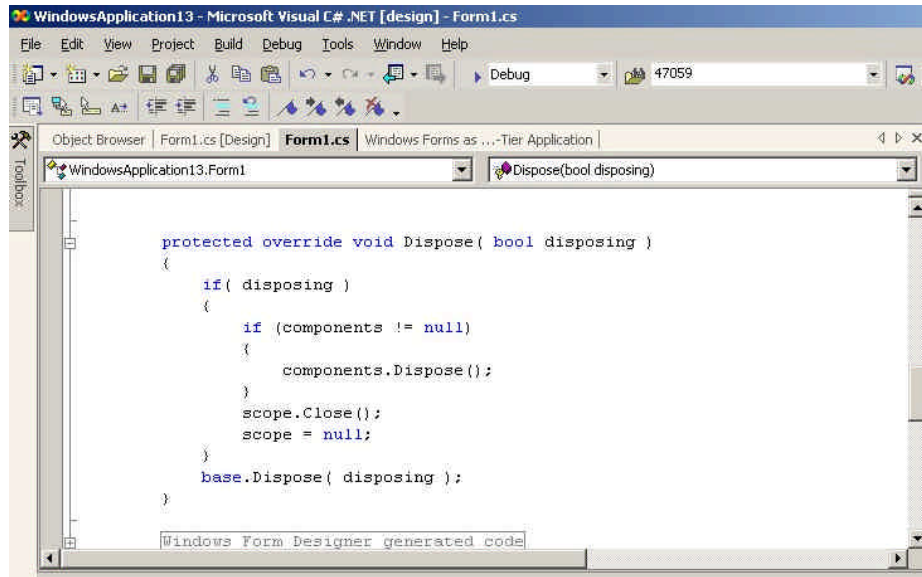


```
WindowsApplication13 - Microsoft Visual C# .NET [design] - Form1.cs*
File Edit View Project Build Debug Tools Window Help
Debug 47059
Object Browser Inserting Text Objects Form1.cs [Design]* Form1.cs*
WindowsApplication13.Form1 Averages_Click(object sender, System.EventArgs e)
}

private void Averages_Click(object sender, System.EventArgs e)
{
    try
    {
        AvBox.Text = String.Format("{0}", scope.Acquisition.NumberOfAverag
    }
    catch (Exception ex)
    {
        MessageBox.Show (ex.Message, "Error !!", MessageBoxButtons.OK);
    }
}
}
Output
```

## Step 5: Tidy-Up the Program

The C# framework automatically provides a Dispose procedure to perform cleanup. This procedure contains code to close the driver and release the reference to it, as shown in the following figure.



```
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
        scope.Close();
        scope = null;
    }
    base.Dispose( disposing );
}
```

Windows Form Designer generated code