

## Developing a VB.NET IVI-COM Driver

### REVISION HISTORY

Date	Description	Revision Letter
02/19/2003	Original Draft	A

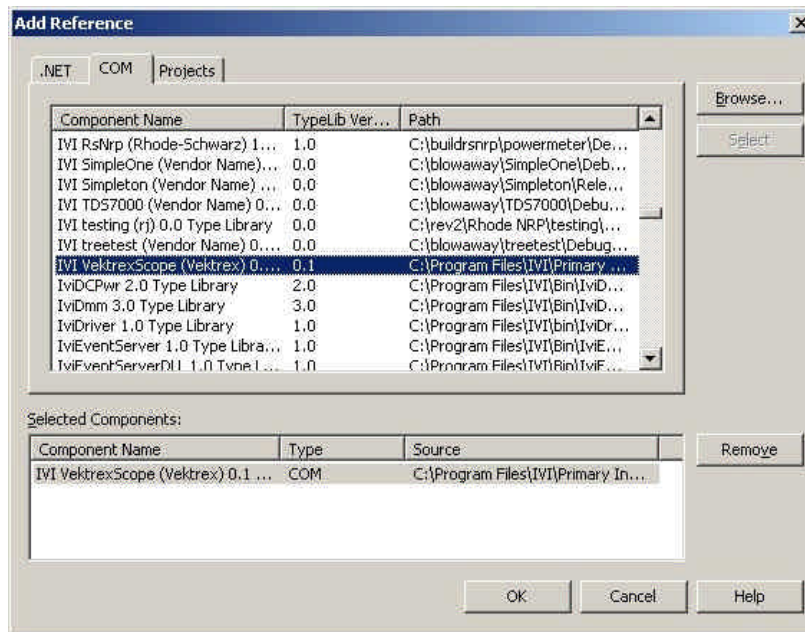
### Abstract

This Application Note describes the five steps required to access an IVI-COM driver using VB.NET. The scope of this example is limited to building a simple project and illustrating some subtle differences between VB.NET and Visual Basic 6 (VB6). The Vektrex Scope driver is used as an example. Specifically, this example shows how to create a client application using VB.NET instrument specific interfaces, but does not use interchangeability features (it does not use the 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> .

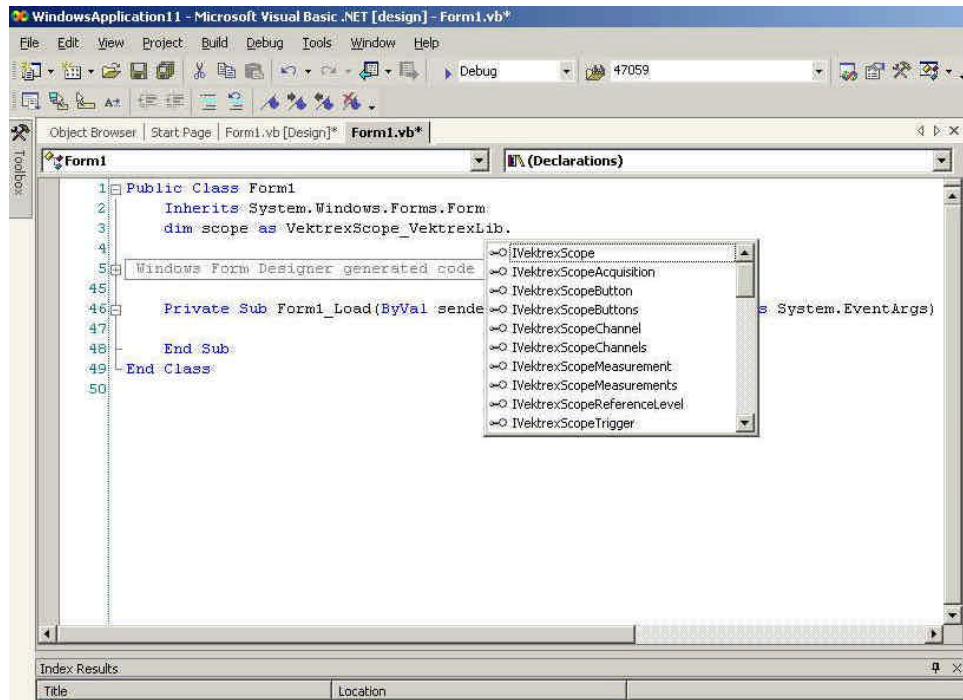
### Step 1: Make the COM Object Available to the Project

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.

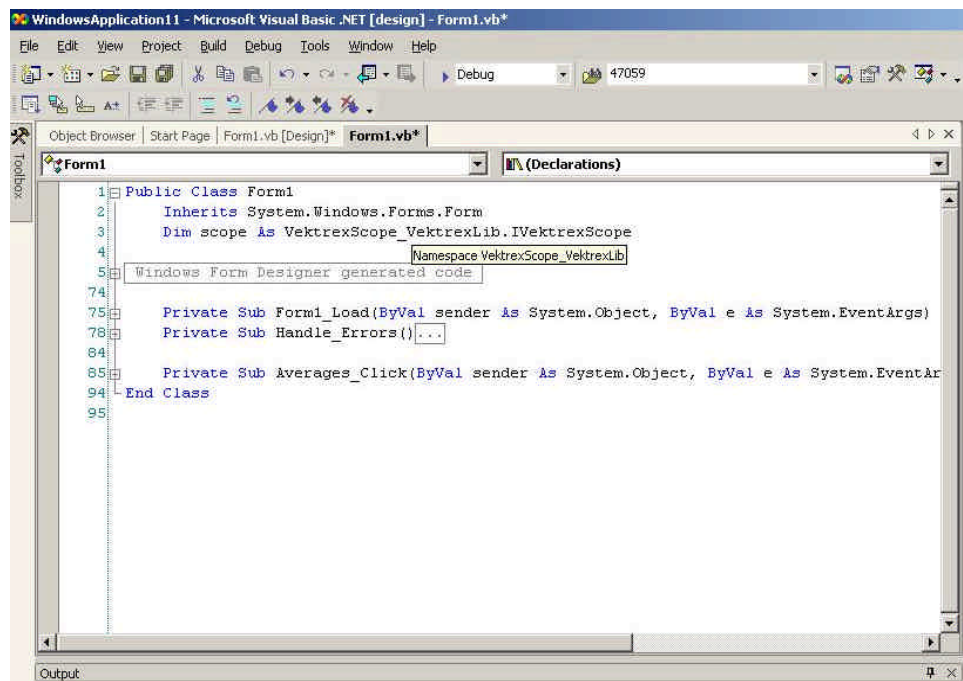


## Step 2: Create an Instance of the Object

At the top of the of the VB.NET code window, type: **“Dim scope as Vek”**. At this point you should have an Intellisense window highlighted with **vektrexScope\_VektrexLib**. Now type a period, and a new Intellisense window appears, as shown below.

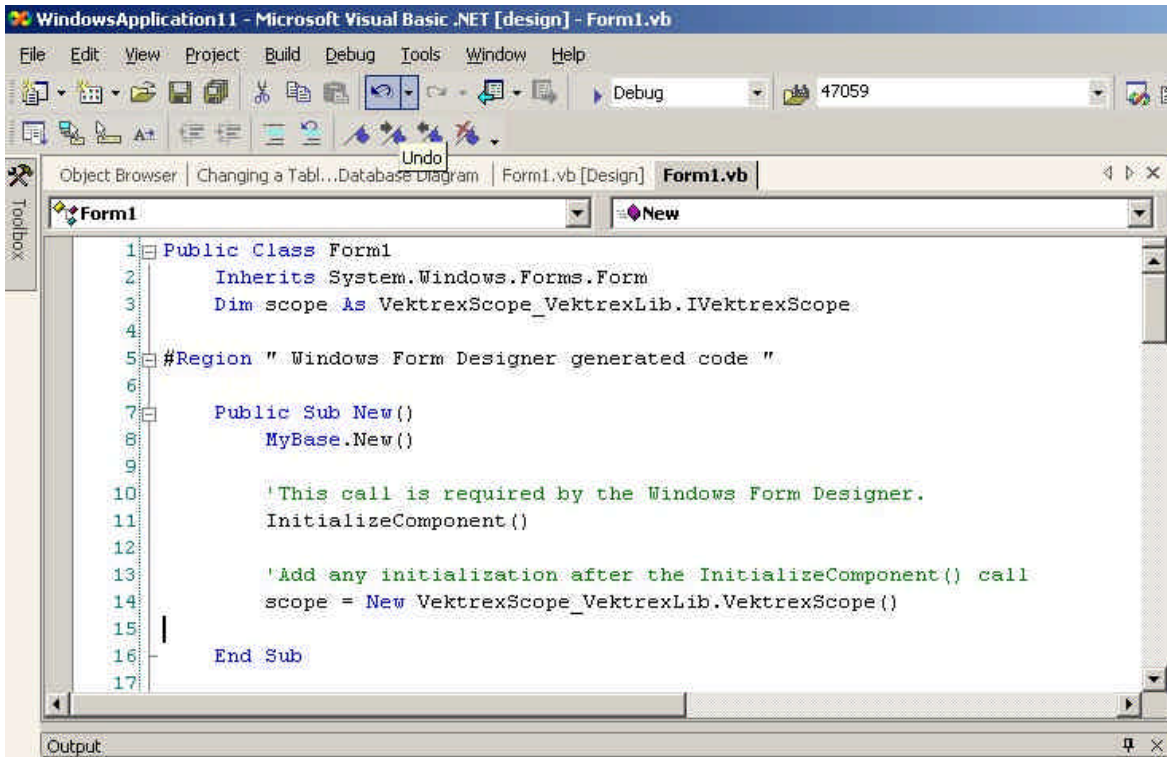


Select **IVektrexScope** and then hit the Enter key. The figure below shows the scope declaration.



One subtle difference between VB6 and VB.NET is that VB.NET incorporates a class-like structure to the Visual Basic code. This structure includes both a New and Dispose procedure. Since the New procedure initializes all the components of the form, it is the logical place for instantiating the scope. The Tidy-Up step of this section discusses the Dispose procedure.

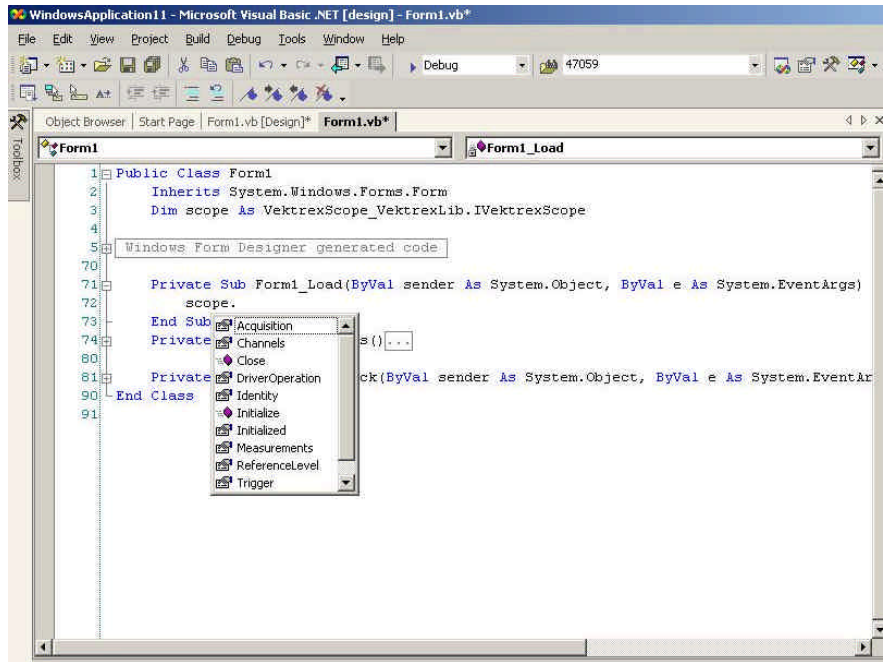
Now click the + sign adjacent to **Windows Form Designer generated code**, to expand the automatically generated code and reveal the New procedure. Within this procedure notice the comment **'Add any initialization after the InitializeComponent() call**. To instantiate the scope, on the following line, type: **"scope = New VektrexScope\_VektrexLib.VektrexScope()"**.



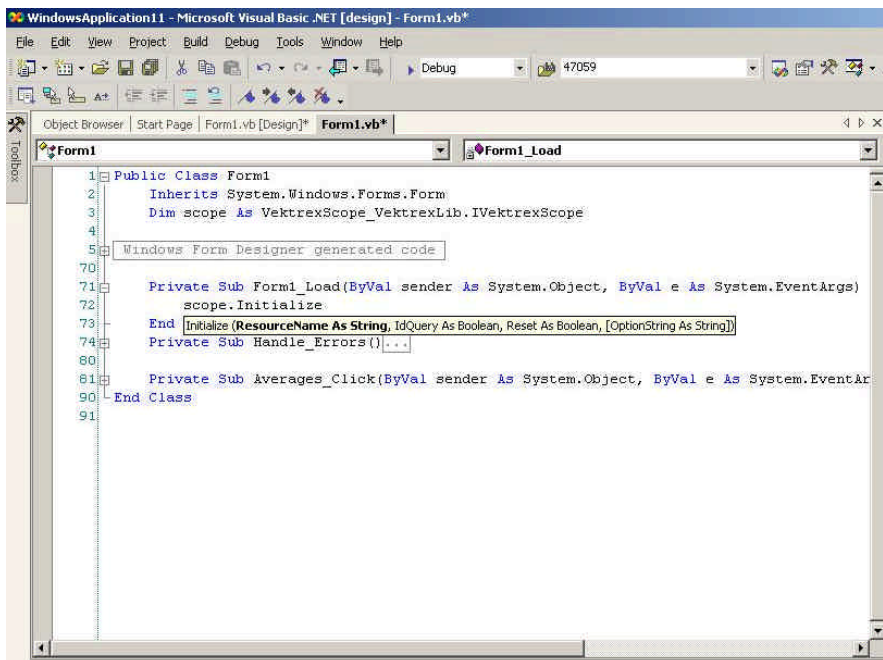
```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3     Dim scope As VektrexScope_VektrexLib.IVektrexScope
4
5     #Region " Windows Form Designer generated code "
6
7     Public Sub New()
8         MyBase.New()
9
10        'This call is required by the Windows Form Designer.
11        InitializeComponent()
12
13        'Add any initialization after the InitializeComponent() call
14        scope = New VektrexScope_VektrexLib.VektrexScope()
15
16    End Sub
17
```

### Step 3: The Form\_Load Procedure

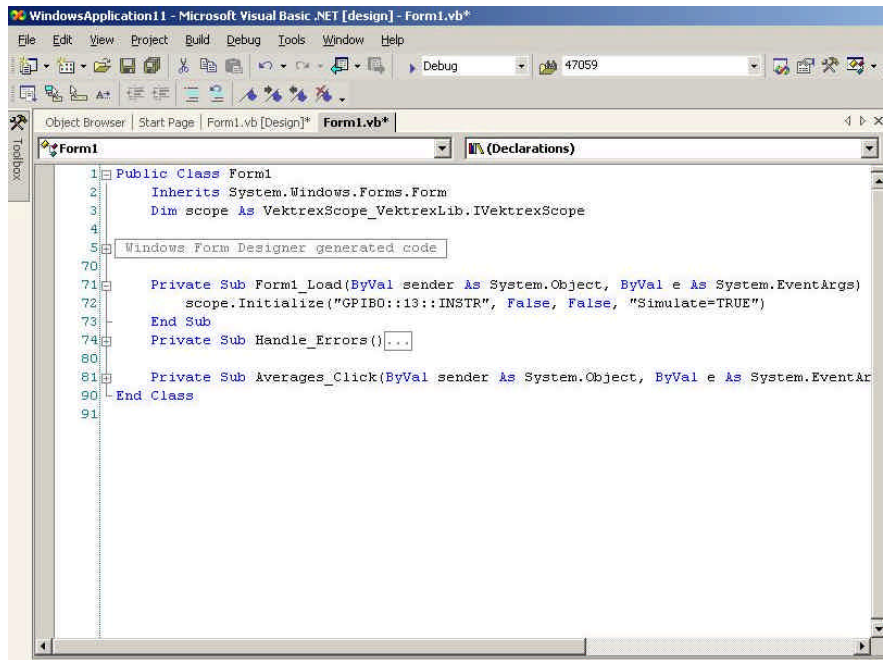
When a VB.NET program runs, the Form\_Load procedure executes as part of the application's initialization routine. Hence, the Form\_Load procedure is an excellent place to locate the call to the driver's initialize function. The figure below shows the Form\_Load procedure, and it shows the developer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



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



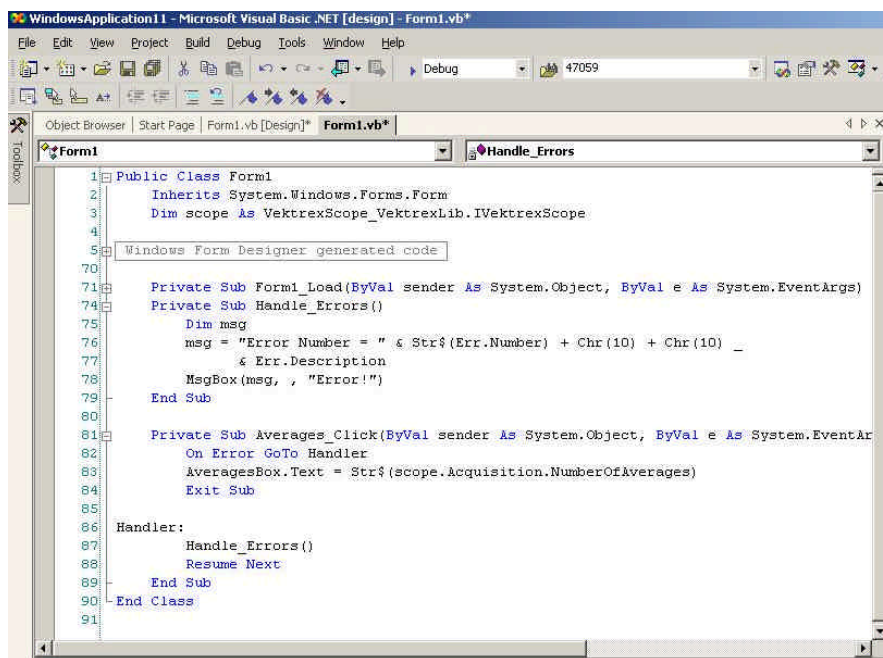
The following figure displays the completed Form\_Load procedure.



```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3     Dim scope As VektrexScope_VektrexLib.IVektrexScope
4
5     Windows Form Designer generated code
6
70
71     Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
72         scope.Initialize("GPIB0::13::INSTR", False, False, "Simulate=TRUE")
73     End Sub
74     Private Sub Handle_Errors()...
75
76
77
78
79
80
81     Private Sub Averages_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
82
83
84
85
86
87
88
89
90 End Class
91
```

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

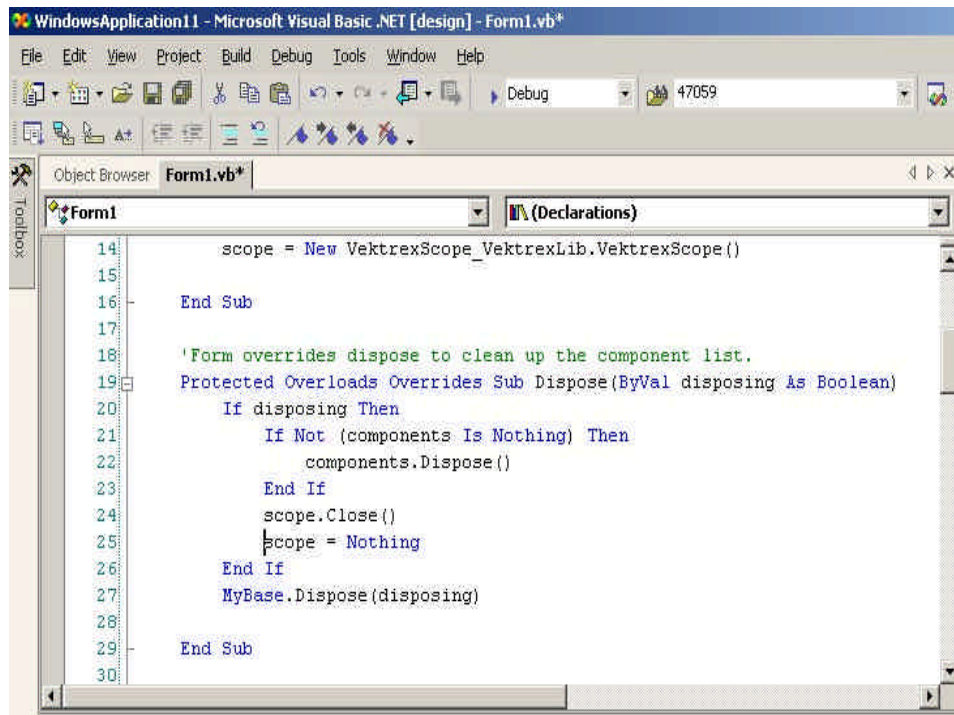
The figure below shows a completed VB.NET procedure utilizing a property in the instrument-specific acquisition interface. Again, Intellisense helps with each step, listing options for any enumerated types. This code assumes the VB.NET form contains a button named Averages, and a textbox named AvBox that displays the returned value. The code also shows how to include error handling. The program calls the routine Handle\_Errors to handle any errors.



```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3     Dim scope As VektrexScope_VektrexLib.IVektrexScope
4
5     Windows Form Designer generated code
6
70
71     Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
72
73
74     Private Sub Handle_Errors()
75         Dim msg
76         msg = "Error Number = " & Str$(Err.Number) + Chr(10) + Chr(10) _
77             & Err.Description
78         MsgBox(msg, , "Error!")
79     End Sub
80
81     Private Sub Averages_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
82         On Error GoTo Handler
83         AveragesBox.Text = Str$(scope.Acquisition.NumberOfAverages)
84         Exit Sub
85
86     Handler:
87         Handle_Errors()
88         Resume Next
89     End Sub
90 End Class
91
```

## Step 5: Tidy-Up the Program

The VB.NET framework automatically provides the Dispose procedure. This procedure contains code to close the driver and release the reference to it, as shown in the following figure. This differs slightly from VB6, because in VB6 the developer must add a cleanup routine.



The screenshot shows the Microsoft Visual Basic .NET IDE with the following code in the Form1.vb file:

```
14:         scope = New VektrexScope_VektrexLib.VektrexScope()
15:
16:     End Sub
17:
18:     'Form overrides dispose to clean up the component list.
19:     Protected Overrides Sub Dispose(ByVal disposing As Boolean)
20:         If disposing Then
21:             If Not (components Is Nothing) Then
22:                 components.Dispose()
23:             End If
24:             scope.Close()
25:             scope = Nothing
26:         End If
27:         MyBase.Dispose(disposing)
28:
29:     End Sub
30:
```