Plug-ins Using VB.NET

Creating CustomDSO Plug-ins Using Visual Basic.NET

LeCroy's CustomDSO feature, included in the XDEV Customization option, enables the creation of custom user interfaces, using programming languages capable of creating ActiveX Controls.

At its announcement, the programming language that was the best match for creating these plug-ins was Microsoft's Visual Basic 6.0. Since then, Microsoft has moved its focus onto its ".NET" technology and has made it more difficult to create "Legacy" ActiveX controls, using the latest version of Visual Basic.

However, armed with the information contained within this Application Brief, you can successfully create plug-ins using Visual Basic .NET 2003.

Note: Since part of the procedure involves typing in several lines of VB code it will be much easier to work from the electronic version of this document, where copy and paste commands can be used to eliminate typing errors.

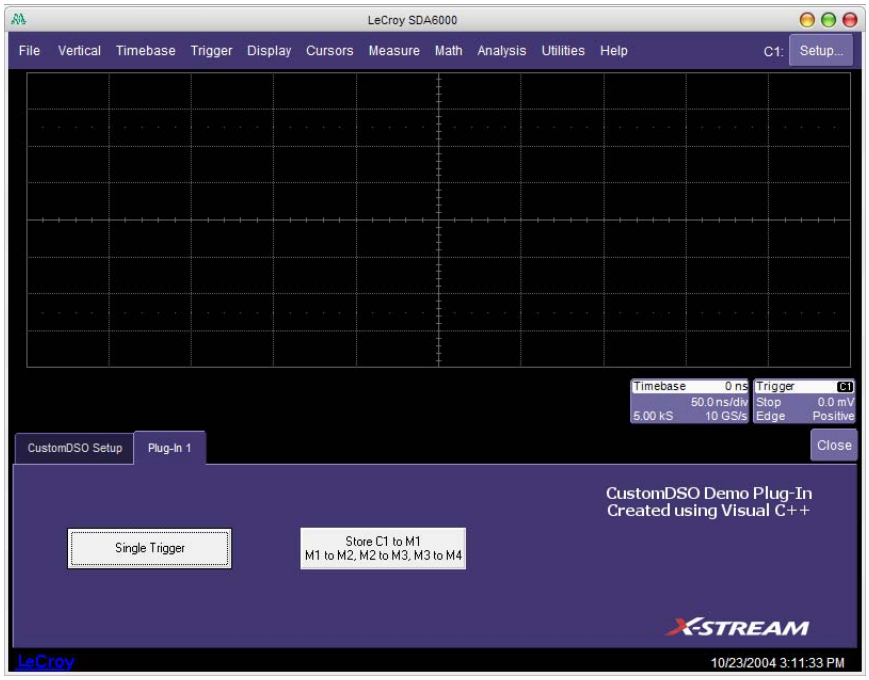


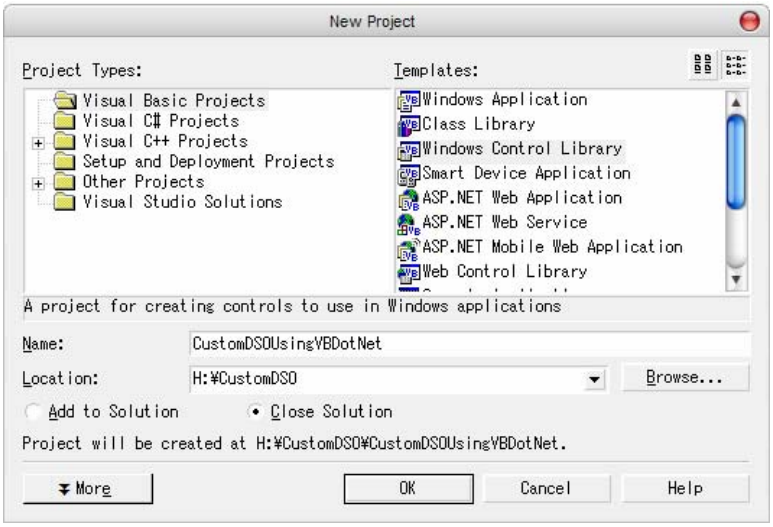
Figure 1. A simple CustomDSO Plug-in

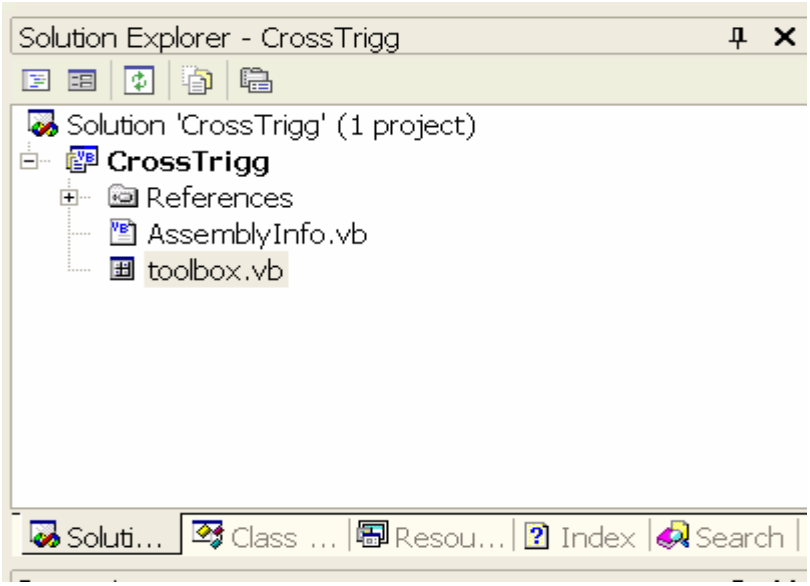
Figure 2. A plug-in created with VB .NET

1. Open Visual Studio .NET 2003 and select the **File->New->Project** menu item.

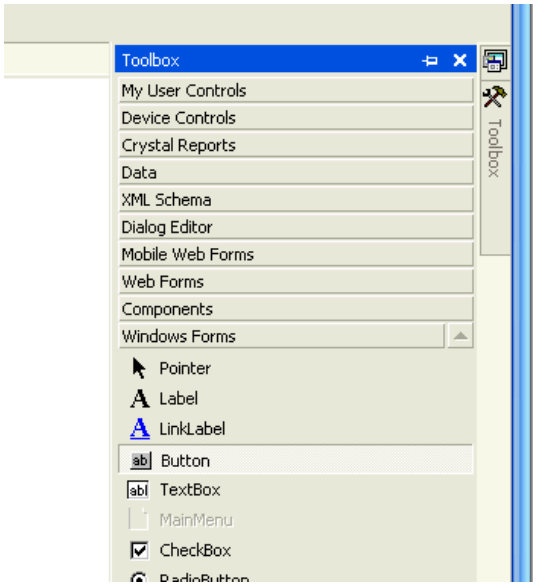
2. Select **Visual Basic Projects** on the left, and **Windows Control Library** on the right.

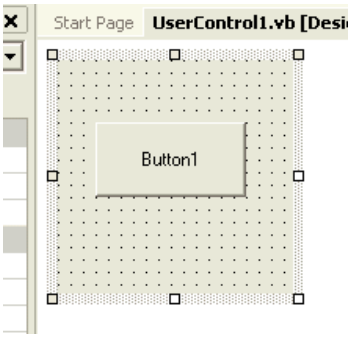
3. Enter a project name and directory into the dialog box that appears and click OK. Give the proper name to the project you want to use on the scope interface when you install it. The syntax which will be used on the CustomDSO menu of the scope will be:
   <Project Name>.<VB.NET class>
   Where *<VB.NET class>* is the name of the Code/Object of your project; for example, in the Solution Explorer below, the Project Name is "CrossTrigg" while the VB.NET class is "toolbox.vb". Therefore under CustomDSO menu of the scope, the plug-in name to install will be:
   "CrossTrigg.toolbox"



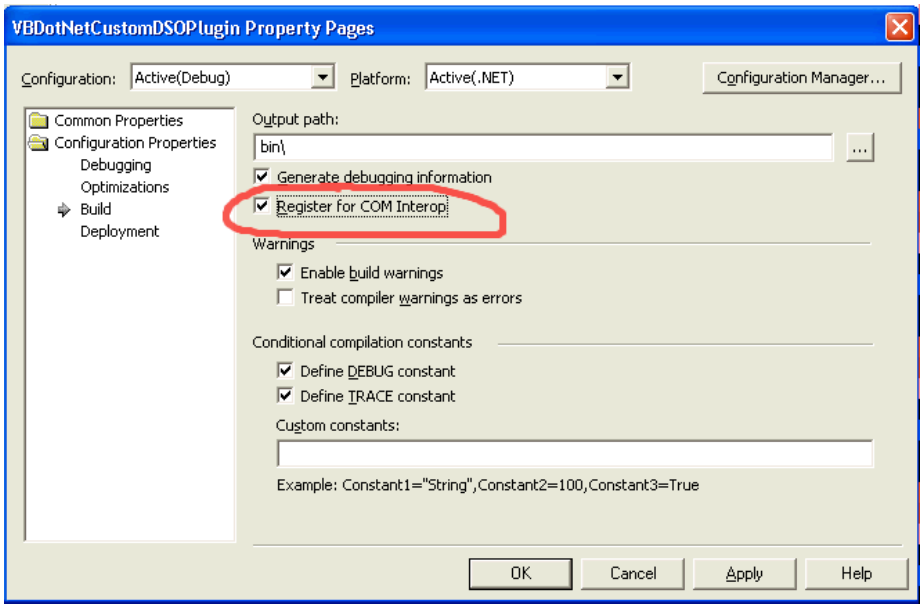


4. For demonstration purposes, drag a button from the toolbox onto the dialog:

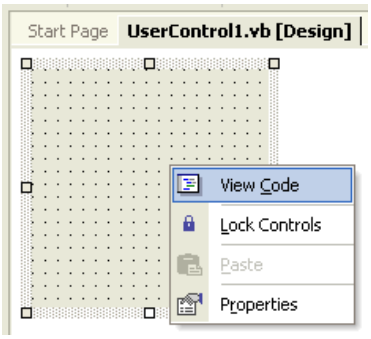




5. Open the project properties and check the **Register for COM Interop** box (be sure to do this for both Debug and Release builds):



6. Right-click on the dialog, and select **View Code**



7. Merge the code below into the source file. Note that the line that starts **Public Class UserControl1** (in the example below is **Public Class Toolbox**) and the line following it should already be in the empty project.

8. The section in red color is the GUIDs section. The GUIDs used below (ClassId, InterfaceId and EventsId) should be unique and, therefore, should be regenerated using the MS “GuidGen” tool. For demonstration purposes the ones below will work, but only for the first plug-in built; second and subsequent plug-ins built using this method require unique IDs. To generate another three IDs for ClassId, InterfaceId and EventId you have to go to Tools --> Create GUID.Then, in the pop-up window, select the GUID format number 4 (Registry Format) and click on “New GUID”.

   Now you can copy it into the clipboard with the relative button and paste it into the local field. You have to repeat this procedure three times, one for each “Id”. Note finally that when you paste the key, you have to manually modify it to remove any extra parentheses.

```
Option Strict Off
Imports System.Runtime.InteropServices
Imports System.Text
Imports System.IO
Imports System.Reflection
Imports Microsoft.Win32
Imports System
Imports System.Threading
Imports System.Math
Imports Microsoft.Office.Core
Imports Microsoft.Vbe.Interop
Imports System.Windows.Forms
Imports Office = Microsoft.Office.Core
Imports Excel = Microsoft.Office.Interop.Excel

<ComClass(Toolbox.ClassId, _
        Toolbox.InterfaceId, _
        Toolbox.EventsId)> _
Public Class Toolbox
    Inherits System.Windows.Forms.UserControl

#Region "COM GUIDs"
    ' These GUIDs provide the COM identity for this class
    ' and its COM interfaces. If you change them, existing
    ' clients will no longer be able to access the class.
    Public Const ClassId As String = "EBDB73C5-AAD9-4f7d-9979-1C9EFA684BEE"
    Public Const InterfaceId As String = "8F49323F-537B-4bc8-B48D-027FDFF71322"
    Public Const EventsId As String = "94578994-2F8C-438f-AE92-BB7CBB734DA0"

#End Region

    ' This function is called when registered (no need to change it)
    <ComRegisterFunction()> _
    Private Shared Sub ComRegister(ByVal t As Type)

        Dim keyName As String = "CLSID¥" & t.GUID.ToString("B")
        Dim key As RegistryKey = Registry.ClassesRoot.OpenSubKey(keyName, True)
        key.CreateSubKey("Control").Close()
        Dim subkey As RegistryKey = key.CreateSubKey("MiscStatus")
        subkey.SetValue("", "131457")
        subkey = key.CreateSubKey("TypeLib")
        Dim libid As Guid = Marshal.GetTypeLibGuidForAssembly(t.Assembly)
        subkey.SetValue("", libid.ToString("B"))
        subkey = key.CreateSubKey("Version")
        Dim ver As Version = t.Assembly.GetName().Version
        Dim version As String = String.Format("{0}.{1}", ver.Major, ver.Minor)
        If version = "0.0" Then version = "1.0"
        subkey.SetValue("", version)
    End Sub
```

```
' This is called when unregistering (no need to change it)
   <ComUnregisterFunction()> _
   Private Shared Sub ComUnregister(ByVal t As Type)
       ' Delete entire CLSID¥{clsid} subtree
       Dim keyName As String = "CLSID¥" + t.GUID.ToString("B")
       Registry.ClassesRoot.DeleteSubKeyTree(keyName)
   End Sub
```

9. When you start placing forms within the class (like buttons, checkboxes, etc.), a region just below the previous code will be generated. It starts with *"#Region " Windows Form Designer generated code ""*.
   The region is split into several sections but the first one is a very important subroutine, since all the code you want to execute at startup must be placed within it, just after InitializeComponent() call (as commented in the box below).

```
#Region " Windows Form Designer generated code "

   Public Sub New()
       MyBase.New()
       'This call is required by the Windows Form Designer.
       InitializeComponent()
       'Add any initialization after the InitializeComponent() call
   End Sub
```

10. Hooking the UI to the X-Stream automation interface using VB is simple, as the following example shows. This will program the DSO's horizontal scale (T/Div) when Button1 is clicked.

```
‘ Note that the following Sub is Private so if it will be the only one in the Class
‘ you must declare it Public otherwise at least another one must be Public
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
   Dim o As Object
   o = CreateObject("LeCroy.XStreamDSO.1")
   o.Acquisition.Horizontal.HorScale = 0.0000002
End Sub
```

11. To polish the component, and to integrate seamlessly into the X-Stream User Interface, change the following properties of the dialog and controls in it.

Dialog Size:	792,168 pixels	
Dialog Background:	66, 55, 114	(fairly dark blue)
Button Background:	112, 104, 160	(lighter blue)
Button Text:	255, 255, 255	(white)

12. After you finish your plug-in and you want to try it, you have to install it on the scope. In this case you need to add a solution to your project with the Setup Wizard. Follow the steps below:

i. Click on **File --> New --> Project** and configure the panel as below (New Project panel).
ii. In step 2 of 5 select the options as below (Setup Wizard 2 of 5 panels).
iii. In the step 3 of 5, configure the panel as below.
iv. Finish the setup wizard procedure.
v. Under menu **Build --> Configuration Manager**, be sure that your panel looks like the one below.
vi. Under menu Build, select **Build** solution, then you will find in the project folder, the setup directory with three files (Setup.exe, Setup.msi, and Setup.ini); those three files must be copied into a folder on the scope's hard disk and installed using Setup.Exe.
vii. Activate CustomDSO plug-in in the scope and enter the proper name for the plug-in, as explained at the beginning of this document.