2006-938: CREATING WEB BASED APPLICATIONS FOR INSTRUMENT DATA TRANSFER USING VISUAL STUDIO.NET

David Hergert, Miami University

Creating Web Based Applications for Instrument Data Transfer Using Visual Studio.NET

This paper discusses various techniques that allow the user to create applications that read from a data acquisition card and transfer the data over the web to an application like Microsoft Excel or Access. The techniques that are described in the paper use the Basic component of Visual Studio.NET. The code described in this paper could be used in a Basic programming course or an instrumentation based lab.

As part of the paper, a simple DLL is described that allows the user to read from a port. Transferring data over the web comparisons are made between Visual Basic 6.0 and Visual Studio.NET.

Visual Basic 6.0 vs. Visual Studio.NET

In the 1990s the author wrote a series of I/O routines in Visual Basic 6 that read data from temperature and pressure transducers on an HVAC trainer, routed the data over a TCP/IP connection, and displayed the data in Excel at a satellite campus. This paper is based on the his experience when attempting to switch from Visual Basic 6 to Visual Studio.NET. Programs with I/O routines written in Visual Basic 6 were fairly difficult to implement in Visual Studio.NET. For example, many DLLs that accessed I/O ports no longer worked, programming an RS-232 interface was different, DDE data exchange to Excel not longer existed, and TCP/IP data transmission programming had changed. This paper describes various methods to accomplish all four tasks using Visual Studio.NET. For those new to Visual Studio.NET, the O'Brien[1] book listed in the bibliography provides a good introduction.

This paper is divided into three separate parts, namely, retrieving data from the instrument, sending the data over TCP/IP to a client, and routing the data from the client into Excel.

Part I: Retrieving Instrument Data

This section will cover three methods to receive data from an instrument, namely through an RS-232 Port, an I/O port, or using DAQmx (the driver for National Instrument cards).

Retrieving Instrument Data from an RS-232 Port

Neither Visual Basic 6.0 nor Visual Studio.NET has the ability to read or write data to I/O ports. To compound this problem, many of the DLLs written for Visual Basic 6.0 to access files no longer work with .NET. However Microsoft has provided a DLL that will allow the programmer to read either from a serial or parallel port. The DLL is called Interop.MSComctLib.DLL and is available from:

http://www.microsoft.com/downloads/details.aspx?FamilyID=075318ca-e4f1-4846-912c-b4ed37a1578b&DisplayLang=en

There is also a sample Basic.NET program called "How To-Using the Comm Port" at this site. Although the example provided is intended for use on a modem, it can be easily modified to work on a serial instrument as well. The function m_CommPort.Open is used to configure an

RS-232 port. An example that sets the port for 600 baud, 7 data bits, no parity, 2 stop bits, and a 13 character buffer would be:

```
m_CommPort.Open(1, 600, 7, Rs232.DataParity.Parity_None, _
        Rs232.DataStopBit.StopBit_2, 13)
```

To write a character to the port use the "m_CommPort.Write" function. An example that writes the character D to the port would be:

```
m_CommPort.Write("D")
```

The m_CommPort.Read() function assumes that bytes are being read. An examples that would read 13 bytes would be:

```
m_CommPort.Read(13)
```

Finally m_CommPort.Close()is used to close the RS-232 port.

Retrieving Instrument Data from an I/O Port

There are a few utilities offered over the web that allow users to access I/O ports in Visual Studio.NET. One DLL that works particularly well is IONET from SSNET. It can be downloaded from:
http://ourworld.compuserve.com/homepages/richard_grier/ionet.htm

To use this DLL, first download and unzip it from the web site, then create a Visual Basic.NET program and add IONET.DLL as a reference. Ionet1.ReadAddress can then be used to refer to a port address, and Ionet1.ReadIO.ToString can be used to read the port into a string. An example that reads the printer port into a text box is shown below:

```
Ionet1.ReadAddress = &H379
TextBox1.Text = Ionet1.ReadIO.ToString
```

Retrieving Instrument Data from a National Instrument Card

National Instruments provides a utility to provide access to their I/O cards using NI DAQmx. The Getting Started Guide NI-DAQmx for USB Devices[2] stated in the bibliography gives a description of how to use a USB data acquisition card with Visual Studio.NET. This reference is available from National Instrument's website at www.ni.com. All DAQ cards from National Instruments come with a driver card. An example of Visual Basic.Net code that reads input from a port is below:

```
myTask = New Task("aiTask")

    myTask.AIChannels.CreateVoltageChannel("Dev1/ai0", "", _
            CType(-1, AITerminalConfiguration), -10.0, _
            10.0, AIVoltageUnits.Volts)
```

```
myTask.Control(TaskAction.Verify)

reader = New AnalogMultiChannelReader(myTask.Stream)

TextBox1.Text = ToString(reader.ReadSingleSample(0))
```

In this example, the AIChannels.CreateVoltageChannel function configures the I/O device for board #1, analog input channel 0, and a range of -10 to 10 volts. The ReadSingleSample function reads in a single value to a text box. The example shown above was implemented on an NI USB-6008 card.

Part II: Sending Data Over TCP/IP

TCP/IP is a well known protocol for sending data between a server and a client. The basic communication structure for client/server communication is shown below:.

Typical TCP/IO Functions:

Client Side		**Server Side**
(get IP address and port)		
		Listen
Connect	→	Accept
Write Request	→	Read Request
Read Response	←	Write Response
Close		Close

First the server is put in listen mode. A client attempts a connection to the server and the server accepts the client. Next the client sends a request to the server. The server reads the request and sends a response back to the client.

Implementing client/server communication in Visual Studio.Net is fairly straightforward. First System.Net.Sockets and System.IO must be imported on both the server and client as shown below:

```
Imports System.Net.Sockets
Imports System.IO
```

Server Programming:
Create a listener using the TcpListener method:

```
Dim Listener As New TcpListener(7000)
```

Place the program in listen status with the function:

```
Listener.Start()
```

Next an attempt is made to accept a connection from a client, receive a request, and send data.

```
Try
   Dim DataClient As TcpClient = Listener.AcceptTcpClient()
   Dim Stream As NetworkStream = DataClient.GetStream()
   Dim ReadData As New BinaryReader(Stream)
   Dim WriteData As New BinaryWriter(Stream)
   Dim x As Integer
   Dim ClientRead As String

   'receive a request from client
   ClientRead = ReadData.ReadString
      If ClientRead = "Send" Then
      'Write instrument data located in text box to client
      WriteData.Write(TextBox1.Text.ToString)
      End If
```

If the connection is successful, this code will write one string of instrument data from the server to the client whenever the string "Send" is received from the client.

Client Programming:
On the client side, the IP address and port number of the server must be known. We have already chose 7000 as the port number in the server. If the client is being implemented on the same computer as the server, the loopback address 127.0.0.1 can be used. Otherwise the IP address of the server must be entered in Client.Connect.

```
Dim Client As New TcpClient

Try
   'Connect with loopback address and port 7000
   Client.Connect(("127.0.0.1"), 7000)
   Dim Stream As NetworkStream = Client.GetStream()
            Dim ReadData As New BinaryReader(Stream)
            Dim WriteData As New BinaryWriter(Stream)

   Dim Astring As String
   'Send the string "Send" as a request to server.
   WriteData.Write("Send")

   'Read response from server into Astring
            Astring = ReadData.ReadString()
            'Place data in text box
            TextBox1.Text = Astring
             'Close Client
   Client.Close()

Catch ex As Exception
```

```
    End Try
```

The complete server code that reads from a port is shown in Figure 2. Included in the figure is a timer that spaces the readings into one minute intervals.

Figure 2 Server Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Ionet1.ReadAddress = &H379
  End Sub

TextBox1.Text = Ionet1.ReadIO.ToString
    Dim Listener As New TcpListener(7000)
    Listener.Start()
    Try
      Dim DataClient As TcpClient = Listener.AcceptTcpClient()
      Dim Stream As NetworkStream = DataClient.GetStream()
      Dim ReadData As New BinaryReader(Stream)
      Dim WriteData As New BinaryWriter(Stream)
      Dim x As Integer
      Dim ClientRead As String
      Dim LoopTime As TimeSpan
      Dim InitialTime As Date
      Dim waitspan As TimeSpan = TimeSpan.FromSeconds(60)

      'receive a request from client
      ClientRead = ReadData.ReadString
      For x = 1 To 5
        If ClientRead = "Send" Then
          WriteData.Write(TextBox1.Text.ToString)
        End If
        InitialTime = DateTime.Now
        Do
          LoopTime = DateTime.Now.Subtract(InitialTime)

        Loop Until LoopTime.Ticks > waitspan.Ticks
      Next

      DataClient.Close()
      Listener.Stop()
    Catch err As Exception
      TextBox1.Text = "Error"
    End Try

  End Sub
```

Part III: Routing Data to Excel

In VB 6 and earlier, Dynamic Data Exchange (DDE) provided a simple yet powerful method of transferring data from a VB program to an Excel Spreadsheet. DDE is no longer supported in Visual Studio.NET, and little information is available on how to replace it.

In Visual Studio.NET there are two methods for transferring data to and from Excel, ADO.NET and OLE. ADO.NET requires setting up Excel as a database. Since Excel is not designed to be a database, this can be quite cumbersome. A simpler method is to use OLE. The Deitel[3] and Macdonald[4] books referenced in the bibliography both describe OLE and ADO.NET in some detail. The Deitel[3] book is particularly useful for more information on routing data to Excel using OLE.

To use OLE, first go to the Solution Explorer and select the COM tab in Add Reference. Add the Microsoft Excel Object 10.0 Library to the list.

Next set up a new Excel application and make it visible:

```
Dim App As New Excel.Application
App.Visible = True
```

Next define an workbook and sheet in Excel:

```
Dim Doc As Excel.Workbook = App.Workbooks.Add()
Dim ExcelSheet As Excel.Worksheet = Doc.Sheets(1)
```

Column headings can be created by:

```
ExcelSheet.Range("A1").Value = "Date and Time"
ExcelSheet.Range("B1").Value = "Reading"
```

Next the width of column A must be large enough to contain the data and time

```
ExcelSheet.Range("A:A").ColumnWidth = 22
```

The date and time is inserted into A2 with the method:

```
ExcelSheet.Range("A2").Value = DateTime.Now
```

Finally the data is written to cell B2 with the method:

```
ExcelSheet.Range("B2”).Value = Astring
```

The complete client code that reads an instrument string from the server and places data in Excel with the current date is shown in Figure 3. Included in the figure is a timer that spaces the readings into one minute intervals.

Figure 3 Client Code

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
```

```
    Dim i As Integer
    Dim Client As New TcpClient
    Dim App As New Excel.Application
    Dim Now As DateTime = DateTime.Now

    App.Visible = True
    Dim Doc As Excel.Workbook = App.Workbooks.Add
    Dim Sheet As Excel.Worksheet = Doc.Sheets(1)

    Sheet.Range("A1").Value = "Date"
    Sheet.Range("B1").Value = "Reading"
    Sheet.Range("A:A").ColumnWidth = 20
    Dim Days As Integer
    Try
        Client.Connect(("127.0.0.1"), 7000)
        Dim Stream As NetworkStream = Client.GetStream()
        Dim w As New BinaryWriter(Stream)
        Dim r As New BinaryReader(Stream)
        Dim LoopTime As TimeSpan
        Dim InitialTime As Date
        Dim Astring As String
        Dim waitspan As TimeSpan = TimeSpan.FromSeconds(60)
        For i = 1 To 5
            w.Write("Send")
            Astring = r.ReadString()
            Sheet.Range("A" & i + 1).Value = DateTime.Now
            Sheet.Range("B" & i + 1).Value = Astring
            InitialTime = DateTime.Now
            Do
                LoopTime = DateTime.Now.Subtract(InitialTime)

            Loop Until LoopTime.Ticks > waitspan.Ticks

        Next

        TextBox1.Text = Astring

        w.Write("Stop")
        Client.Close()

    Catch ex As Exception

    End Try

End Sub
```

The Excel sheet looks like this after five readings spaced one minute apart:

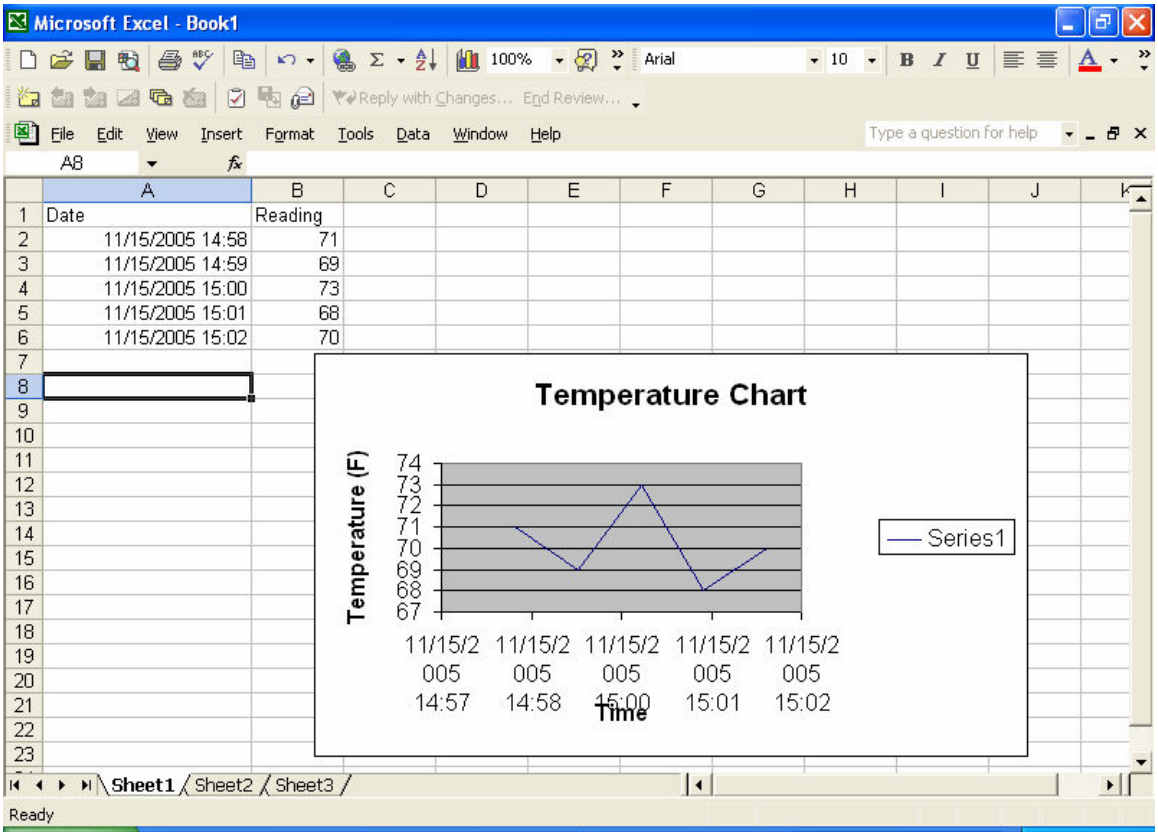

Figure 4. Excel Spreadsheet

The programs shown above could easily be modified to have a continuous stream of data sent to Excel. FOR loops could enclose WriteData.Write(TextBox1.Text.ToString) and ExcelSheet.Range("B1").Value = StringValue to allow for multiple rows to be filled in. As an example the latter function could be coded as:

```
For x = 1 to 10
        ExcelSheet.Range("B" & i+1).Value = StringValue
Next
```

This will increment the row each time data is read into Excel.

Conclusions

This paper describes a complete process for reading data from an instrument and routing it to Excel using DDE and TCP/IP. The methods shown here could be used for laboratory experiments that use large equipment (such as an HVAC trainer, a wind tunnel, or a heat exchanger) in a distance education setting. Students can control the equipment and read data at the remote site. The Visual Basic.NET programs described in this paper are presently being used

in a thermodynamics class broadcast from Miami University-Hamilton to five community colleges in the State of Ohio.

Bibliography

O'Brien B, Seaver C, Microsoft Visual Basic.NET Programming Essentials ,pp 290-353 Microsoft, 2004

National Instruments, Getting Started Guide NI-DAQmx for USB Devices, 2005

Deitel, Visual Basic.NET How to Program 2nd Edition, pp. 1097-1135, Prentice Hall, 2002

MacDonald, Microsoft Visual Basic.NET Programmers Cookbook, Microsoft, 2003