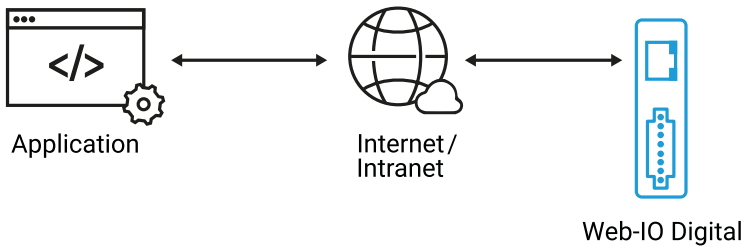
Tutorial for the Web-IO Digital:

Control and monitor WEB-IO Digital with current Visual Basic versions

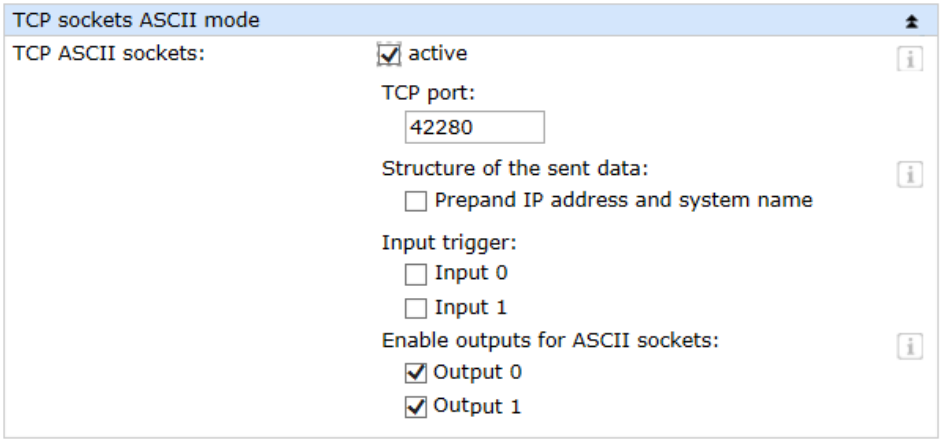
As successor to MS Visual Basic 5 and 6 there is now VB.Net in various versions up to VB 2019. The current Visual Basic version also offers everything you need for programming TCP/IP applications. This makes Visual Basic a favorite tool for creating applications that communicate with the Web-IO Digital, especially since the Express version of Visual Basic is offered from Microsoft free for downloading. Additional drivers or DLLs are not needed.



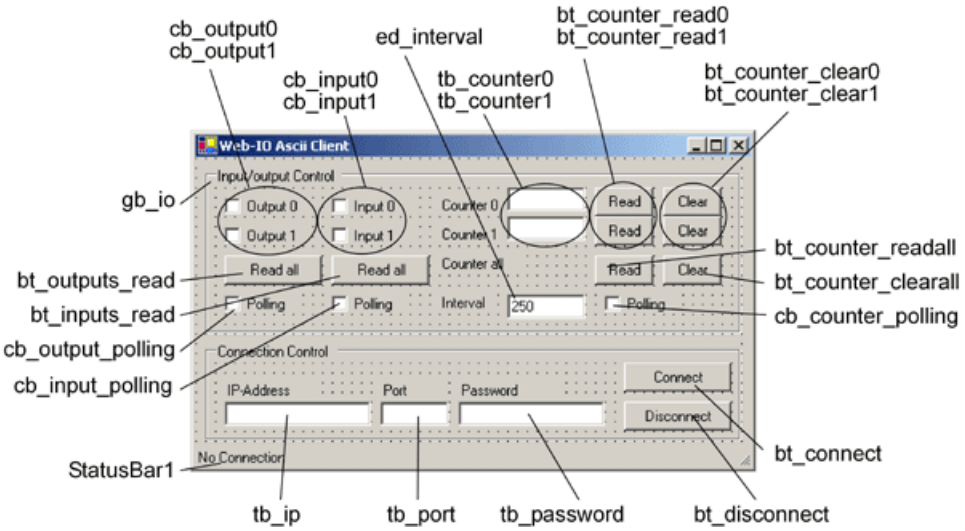
Using the following program example you can represent your Web-IO Digital with its inputs and outputs in a Windows application. You can also switch the Web-IO outputs.

Preparations

- Provide power to the Web-IO and connect the IOs
- Connect the Web-IO to the network
- Assign IP addresses
- On the Web-IO in *Communication channels >> Socket API* activate *TCP-ASCII sockets* and *enable outputs for switching*



Combining the various operating elements and display objects in the VB.net form



In addition to the objects shown here, the program also needs a timer for polling (timer_polling).

When naming the individual objects it is helpful to use logical names. In this example the first part of the name describes the type of object and the second part the function.

Starting the program

While constructing graphical user interfaces in Visual Basic Express is just as simple to accomplish as with the previous versions, other tasks may at first take some getting used to.

VB in its current version no longer uses the familiar Winsock control element consisting of VB5 and VB6.

Instead, you must use the *Import* statement to instance the namespace for the socket classes in the header of the source text. In addition, the socket over which you want communication to be handled must be created and you must define a buffer for the input data.

Furthermore, encapsulation of individual objects in VB.Net was advanced so far that objects which are created within a thread do not permit access from another thread. For example, a checkbox which was created in Thread A cannot be set from Thread B.

To still make object properties accessible for other threads, sub-procedures for object changing need to be created in a separate thread. Delegates are then formed for these sub-procedures through which the other threads can access.

For example in order to enable control elements in the form after successfully opening a connection or to block them after the connection is closed, but also in order to adjust the input/output checkboxes to the actual I/O states after data is received.

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

Public Class Form1
  Public Structure IO_State
    Dim outputstate0 As Boolean
    Dim outputstate1 As Boolean
    Dim inputstate0 As Boolean
    Dim inputstate1 As Boolean
    Dim countervalue0 As Long
    Dim countervalue1 As Long
  End Structure

  Private Delegate Sub DelegateSub()
  Private connectenable As New DelegateSub(AddressOf connect_enable)
  Private disconnectenable As New DelegateSub(AddressOf disconnect_enable)
  Private formupdate As New DelegateSub(AddressOf form_update)
  Dim TCP_client As Socket
  Dim connection_ar As IAsyncResult
  Dim receivebuffer(511) As Byte
  Dim IOState As IO_State

  Private Sub connect_enable()
    bt_connect.Enabled = False
    gb_io.Enabled = True
    bt_disconnect.Enabled = True
    Timer_polling.Enabled = True
    ToolStripStatusLabel1.Text = "Connected to " + tb_ip.Text + " : " + tb_port.Text
  End Sub

  Private Sub disconnect_enable()
    bt_connect.Enabled = True
    gb_io.Enabled = False
    bt_disconnect.Enabled = False
    timer_polling.Enabled = False
    ToolStripStatusLabel1.Text = "No Connection"
  End Sub

  Private Sub form_update()
    If IOState.inputstate0 = True Then
      cb_input0.Checked = True
    Else
      cb_input0.Checked = False
    End If
    If IOState.inputstate1 = True Then
      cb_input1.Checked = True
    Else
      cb_input1.Checked = False
    End If
    If IOState.outputstate0 = True Then
      cb_output0.Checked = True
    Else
      cb_output0.Checked = False
    End If
    If IOState.outputstate1 = True Then
      cb_Output1.Checked = True
    Else
      cb_Output1.Checked = False
    End If
    tb_counter0.Text = IOState.countervalue0
    tb_counter1.Text = IOState.countervalue1
  End Sub
```

Connection control

Establishing the connection

The connection is opened by entering the IP address of the Web-IO in the text field *ed_ip* and clicking on the *bt_connect* button.

```
Private Sub bt_connect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_connect.Click
  Dim WebIOep As New IPEndPoint(IPAddress.Parse(tb_ip.Text), Val(tb_port.Text))
  If tb_ip.Text <> "" And tb_port.Text <> "" Then
    TCP_client = New Socket(AddressFamily.InterNetwork, SocketType.Stream, _ ProtocolType.Tcp)
    bt_connect.Enabled = False
    Try
      TCP_client.BeginConnect(WebIOep, New AsyncCallback(AddressOf callback_connect), TCP_client)
    Catch ex As Exception
    End Try
  End If
End Sub
```

Opening the connection

For TCP/IP handling first an IPEndPoint is defined from IP address and TCP port and used to initialize the TCP_client socket. As part of the connection request a reference to a callback procedure is created.

Connection is made

As soon as the Web-IO accepts the connection, the callback procedure is run. An invoke is used to invoke the delegate for the procedure, which enables the control elements and displays the connect status in the status line. In addition a reference to a callback routine for data reception is created.

```
Private Sub callback_connect(ByVal ar As IAsyncResult)
  Invoke(connectenable)
  connection_ar = ar
  Try
    TCP_client.EndConnect(ar)
    TCP_client.BeginReceive(receivebuffer, 0, 512, SocketFlags.None, New AsyncCallback(AddressOf callback_readdata)
  Catch ex As Exception
    closeconnections()
  End Try
End Sub
```

Disconnecting

The connection remains open until it is ended by the user clicking on the Disconnect button, or the Web-IO ends the connection.

```
Private Sub bt_disconnect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_disconnect.
  Invoke(disconnectenable)
  closeconnections()
End Sub
```

In this case a corresponding procedure is invoked.

```
Private Sub closeconnections()
  Try
    TCP_client.EndReceive(connection_ar)
  Catch ex As Exception
  End Try
  Try
    TCP_client.Shutdown(SocketShutdown.Both)
  Catch ex As Exception
  End Try
  Try
    TCP_client.Close()
  Catch ex As Exception
  End Try
  Invoke(disconnectenable)
End Sub
```

Connection error

All the actions affecting TCP/IP communication are carried out within the Try-instruction. If errors occur, the CloseConnection procedure is also invoked.

Operation and communication from the client side

Sending commands

As soon as a connection is made with the Web-IO, the user can use the corresponding program elements to send commands to the Web-IO.

```
Private Sub sendcommand(ByVal sendstring As String)
  Dim senddata As Byte() = System.Text.Encoding.ASCII.GetBytes(sendstring)
  Try
    TCP_client.Send(senddata)
  Catch ex As Exception
    closeconnections()
  End Try
End Sub
```

Setting the outputs

The user sets the outputs by using the two check boxes *cb_outputx*. The program uses the MouseUP event of this object. If a MouseUp, i.e. releasing the output check box, is used, the program carries out the corresponding procedure and - depending on whether the check box is set or not - passes the appropriate command to the Web-IO.

```
Private Sub cb_output0_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handl
  If cb_output0.Checked Then
    sendcommand("GET /outputaccess0?PW=" + tb_password.Text + "&State=ON&")
  Else
    sendcommand("GET /outputaccess0?PW=" + tb_password.Text + "&State=OFF&")
  End If
End Sub
```

```
Private Sub cb_output1_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handl
  If cb_output1.Checked Then
    sendcommand("GET /outputaccess1?PW=" + tb_password.Text + "&State=ON&")
  Else
    sendcommand("GET /outputaccess1?PW=" + tb_password.Text + "&State=OFF&")
  End If
End Sub
```

Querying output/input status

The user can request the status of the outputs and inputs by clicking on the corresponding button.

```
Private Sub bt_outputs_read_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_outputs_r
  sendcommand("GET /output?PW=" + tb_password.Text + "&")
End Sub
```

```
Private Sub bt_inputs_read_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_inputs_rea
  sendcommand("GET /input?PW=" + tb_password.Text + "&")
End Sub
```

Read/clear counters

Also the counter states of the input counters can be read or cleared.

```
Private Sub bt_counter_read0_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counter_
  sendcommand("GET /counter0?PW=" + tb_password.Text + "&")
End Sub
```

```
Private Sub bt_counter_read1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counter_
  sendcommand("GET /counter1?PW=" + tb_password.Text + "&")
End Sub
```

```
Private Sub bt_counter_clear0_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counter
  sendcommand("GET /counterclear0?PW=" + tb_password.Text + "&")
End Sub
```

```
Private Sub bt_counter_clear1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counter
  sendcommand("GET /counterclear1?PW=" + tb_password.Text + "&")
End Sub
```

Of course all the counters can be read or cleared at the same time.

```
Private Sub bt_counter_readall_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counte
  sendcommand("GET /counter?PW=" + tb_password.Text + "&")
End Sub
```

```
Private Sub bt_counter_clearall_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles bt_counte
  sendcommand("GET /counterclear?PW=" + tb_password.Text + "&")
End Sub
```

Receiving data from the Web-IO

Process and display the received data

All commands and requests to the Web-IO are acknowledged with a reply string. The replies have a specific structure depending on the type.

- For the outputs: output;<binary value of the output status in hexadecimal format>
- For the inputs: input;<binary value of the input status in hexadecimal format>
- For the counters: counterx;<decimal counter state>
- or counter;<decimal counter state 0 >; <decimal counter state 0 >;.....if you want to read all counters at the same time.
- All reply strings are finished off with a 0 byte.
- When data are received the corresponding Callback procedure is invoked

```
Private Sub callback_readdata(ByVal ar As IAsyncResult)
  If TCP_client.Connected Then
    Dim bytesread As Integer
    Try
      bytesread = TCP_client.EndReceive(ar)
    Catch ex As Exception
    End Try
    If bytesread = 0 Then
      closeconnections()
    Else
      Dim receivestring As String
      receivestring = System.Text.Encoding.ASCII.GetString(receivebuffer, 0, receivebuffer.Length)
      receivebuffer.Clear(receivebuffer, 0, 512)
      Try
        TCP_client.BeginReceive(receivebuffer, 0, 512, SocketFlags.None, New AsyncCallback(AddressOf callback_readdat
      Catch ex As Exception
        closeconnections()
      End Try
      Select Case Mid(receivestring, 1, 1)
        Case "i"
          If (Val(Mid(receivestring, 7, 1)) And 1) = 1 Then
            IOState.inputstate0 = True
          Else
            IOState.inputstate0 = False
          End If
          If (Val(Mid(receivestring, 7, 1)) And 2) = 2 Then
            IOState.inputstate1 = True
          Else
            IOState.inputstate1 = False
          End If
        Case "o"
          If (Val(Mid(receivestring, 8, 1)) And 1) = 1 Then
            IOState.outputstate0 = True
          Else
            IOState.outputstate0 = False
          End If
          If (Val(Mid(receivestring, 8, 1)) And 2) = 2 Then
            IOState.outputstate1 = True
          Else
            IOState.outputstate1 = False
          End If
        Case "c"
          Dim tabpos
          If Mid(receivestring, 8, 1) = "0" Then
            IOState.countervalue0 = Mid(receivestring, 10)
          If Mid(receivestring, 8, 1) = "1" Then _
            IOState.countervalue1 = Mid(receivestring, 10)
          If Mid(receivestring, 8, 1) = ";" Then
            tabpos = InStr(9, receivestring, ";")
            IOState.countervalue0 = Val(Mid(receivestring, 9, tabpos - 9))
            IOState.countervalue1 = Val(Mid(receivestring, tabpos + 1, Len(receivestring) - tabpos - 1))
          End If
      End Select
      Invoke(formupdate)
    End If
  End If
End Sub
```

The Receive procedure uses the first character of the receive data to check whether these are input, output or counter messages. Depending on this, a determination is made for example as to which output has which status. With the counters it is possible both to query individual counter values as well as to read out all the counters at one time. The individual counter states are then output in decimal format in a semicolon delimited string.

Polling

Cyclical polling of particular values
In order to enable automatic refreshing of the display, a timer is used.

Depending on the check boxes for output, input and counter polling, the corresponding information is obtained from the Web-IO at a set interval.

```
Private Sub timer_polling_Elapsed(ByVal sender As System.Object, ByVal e As System.Timers.ElapsedEventArgs) Handl
  If (cb_input_polling.Checked And TCP_client.Connected) Then
    sendcommand("GET /input?PW=" + tb_password.Text + "&")
  End If
  If (cb_output_polling.Checked And TCP_client.Connected) Then
    sendcommand("GET /output?PW=" + tb_password.Text + "&")
  End If
  If (cb_output_polling.Checked And TCP_client.Connected) Then
    sendcommand("GET /output?PW=" + tb_password.Text + "&")
  End If
  If (cb_counter_polling.Checked And TCP_client.Connected) Then
    sendcommand("GET /counter?PW=" + tb_password.Text + "&")
  End If
End Sub
```

The desired interval can be entered in the corresponding text field. When changes are made the timer interval is automatically adjusted.

```
Private Sub tb_interval_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb_interva
  timer_polling.Interval = Val(tb_interval.Text)
End Sub
```

The sample program supports all common functions of the Web-IO in command string mode, optimized for the Web-IO 2x Digital Input, 2x Digital Output. For the other Web-IO models you may have to adapt the program. Additional program examples for socket programming can be found on the tool pages for the Web-IO. A detailed description for the socket interface of the Web-IO Digital models can be found in the reference manual.

Download program example

Products

Web-IO 4.0 Digital
2xIn, 2xOut

Power via PoE also when needed

Web-IO 4.0 Digital
12xIn, 12xOut

12x inputs,
12x outputs

Other Web-IOs

All W&T Web-IO Digital 24V

W&T
www.WuT.de

We are available to you in person:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Phone: +49 202/2680-110 (Mon.-Fri. 8 a.m. to 5 p.m.)
Fax: +49 202/2680-265
info@wut.de