Applikation zum Web-IO Digital:

Weitere Applikationen
Produktübersicht

Web-IO Digital mit VB.Net steuern und überwachen

Als Nachfolger von MS Visual Basic hat sich inzwischen VB.net etabliert. Visual Basic.net bietet alles, was zum Programmieren von TCP/IP-Anwendungen nötig ist. Damit ist Visual Basic.net ein beliebtes Hilfsmittel um Anwendungen zu erstellen, die mit dem Web-IO Digital kommunizieren. Zusätzliche Treiber oder DLLs werden nicht benötigt.

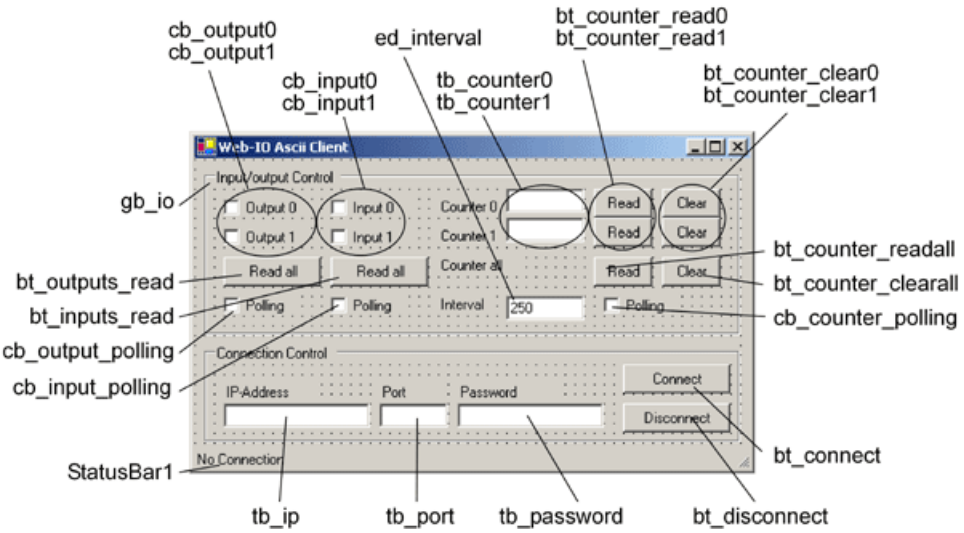
Mit dem folgenden Programmbeispiel können Sie Ihr Web-IO Digital mit seinen Inputs und Outputs in einer Windows-Anwendung abbilden. Darüber hinaus können Sie die Outputs des Web-IO schalten.

Vorbereitungen

Sie haben Ihr Web-IO Digital bereits

- mit Strom versorgt
- Eingänge und Ausgänge beschaltet
- an Ihr Netzwerk angeschlossen
- mit einer IP-Adresse versehen - mit WuTility kein Problem

1. Zusammenstellen der verschiedenen Bedienelemente und Anzeigeobjekte im VB.net-Form



Neben den hier gezeigten Objekten benötigt das Programm zusätzlichen einen Timer für das Polling (timer_polling).

Bei der Benennung der einzelnen Objekte ist es hilfreich, sinngebende Namen zu verwenden. In diesem Beipiel beschreibt der erste Teil des Namens die Art des Objektes und der zweite Teil die Funktion.

1. Programmstart

Während der Aufbau grafischer Benutzeroberfächen in VB.net genauso einfach zu bewerkstelligen ist, wie mit VB5 oder VB6, gestaltet sich die Netzwerkkommunikation unter VB.net etwas schwieriger als beim Vorgänger. Das liegt in erster Linie daran, dass Microsoft bei VB.net das Winsock-Steuerelement nicht mehr zur Verfügung stellt. Statt dessen muss für den Netzwerkzugriff über das *Import* Statement der Namespace für die verwendeten Socket-Klassen im Kopf des Quelltextes instanziert werden.

Darüber hinaus muss der Socket, auf dem die Kommunikation abgewickelt werden soll, angelegt werden und es muss ein Buffer für die Eingangsdaten definiert werden.

```
Imports System.Net
Imports System.Net.Sockets
Imports System.IO
Imports System.Windows.Forms
Public Class Form1
 Inherits System.Windows.Forms.Form
 Dim TCP_client As Socket
 Dim receivebuffer(511) As Byte
```

2. Die Verbindungskontrolle

Einleiten der Verbindung
Durch Eingabe der IP-Adresse des Web-IO in das Textfeld *ed_ip* und Klick auf den Button *bt_connect* wird der Verbindungsaufbau gestartet.

```
 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
    closeconnections()
    MessageBox.Show(ex.Message, "Socket Error", MessageBoxButtons.OK, _
    MessageBoxIcon.Error)
   End Try
  End If
 End Sub
```

Verbindungsaufbau
Für die Abwicklung des TCP/IP-Handlings wird zunächst ein IPEndPoint aus Ip-Adresse und TCP-Port definiert und damit der TCP_client Socket initialisiert. Im Zuge der Verbindungsanforderung wird ein Verweis auf eine Callback Prozedur geschaffen.

Verbindung kommt zustande
Sobald das Web-IO die Verbindung annimmt, wird die Callback Prozedur ausgeführt. In der Statuszeile wird das Zustandekommen der Verbindung angezeigt, die Bedienelemente werden zur Benutzung freigegeben und der Disconnect-Button wird bedienbar. Der Connect-Button wird für die Bedienung gesperrt. Darüber hinaus wird ein Verweis auf eine Callbackroutine für den Datenempfang erstellt.

```
 Private Sub callback_connect(ByVal ar As IAsyncResult)
  bt_connect.Enabled = False
  gb_io.Enabled = True
  bt_disconnect.Enabled = True
  StatusBar1.Text = "Connected to " + tb_ip.Text + " : " + tb_port.Text
  timer_polling.Enabled = True
   Try
   TCP_client.EndConnect(ar)
   TCP_client.BeginReceive(receivebuffer, 0, 512, SocketFlags.None, New AsyncCallback(Address
   Catch ex As Exception
   closeconnections()
   StatusBar1.Text = "error on connecting"
   End Try
 End Sub
```

Trennen der Verbindung
Die Verbindung bleibt solange bestehen, bis sie vom Benutzer durch Klick auf den Disconnect-Button beendet wird oder das Web-IO die Verbindung beendet.

```
 Private Sub bt_disconnect_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles bt_disconnect.Click
  bt_disconnect.Enabled = False
  closeconnections()
 End Sub
```

In diesem Fall wird eine entsprechende Prozedur aufgerufen.

```
Private Sub closeconnections()
 Dim ar As IAsyncResult
 timer_polling.Enabled = False
 Try
  TCP_client.EndReceive(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
 bt_connect.Enabled = True
 gb_io.Enabled = False
 bt_disconnect.Enabled = False
 StatusBar1.Text = "no connection"
End Sub
```

Verbindungsfehler
Alle die TCP/IP-Kommunikation betreffenden Aktionen werden innerhalb der Try-Anweisung ausgeführt. Treten Fehler auf, wird ebenfalls die CloseConnection Prozedur aufgerufen.

3. Bedienung und Kommunikation von Client-Seite

Sobald eine Verbindung mit dem Web-IO zustande gekommen ist, kann der Anwender durch Bedienung der entsprechenden Programmelemente Kommandos an das Web-IO senden.

```
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
```

Setzen der Outputs
Das Setzen der Outputs wird dem Anwender über zwei Checkboxen cb_outputx ermöglicht. Das Programm nutzt dazu das MouseUP-Ereignis dieses Objektes. Wird ein MouseUp, also ein Loslassen der Output-Checkbox registriert, führt das Programm die entsprechende Prozedur aus und gibt - je nach dem, ob die Checkbox gesetzt ist oder nicht - das passende Kommando an das Web-IO weiter.

```
Private Sub cb_output0_MouseUp(ByVal sender As Object, _
ByVal e As System.Windows.Forms.MouseEventArgs) Handles cb_output0.MouseUp
 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) Handles cb_output1.MouseUp
 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
```

Output/Input-Status abfragen
Den Status der Outputs und Inputs kann der Anwender durch Anklicken des zugehörigen Buttons anfordern.

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

Counter abfragen/löschen
Auch die Zählerstände der Input-Counter lassen sich abfragen bzw. löschen.

```
Private Sub bt_counter_read0_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles bt_counter_read0.Click
 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_read0.Click
 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_clear0.Click
 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_clear0.Click
 sendcommand("GET /counterclear1?PW=" + tb_password.Text + "&")
End Sub
```

Natürlich lassen sich auch alle Counter zeitgleich lesen bzw. löschen.

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

3. Datenempfang vom Web-IO

- Auswerten und Anzeigen der empfangenen Daten
  Alle Kommandos und Anfragen an das Web-IO werden mit einem Antwort-String quittiert. Dabei haben die Antworten je nach Type einen spezifischen Aufbau.
- Für die Outputs: output;<Binärwert des Outputstatus im hexadezimalen Format>
- Für die Inputs: input;<Binärwert des Outputstatus im hexadezimalen Format>
- Für die Counter: counterx;<dezimaler Zählerstand>
- oder counter;<dezimaler Zählerstand 0 >; <dezimaler Zählerstand 0 >; ..... wenn alle Counter auf einmal gelesen werden sollen.
- Alle Antwort-Strings sind mit einem 0-Byte abgeschlossen.
- Bei Datenempfang wird die entsprechende Callback Prozedur aufgerufen

```
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(Address
 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
  cb_input0.Checked = True
  Else
  cb_input0.Checked = False
  End If
  If (Val(Mid(receivestring, 7, 1)) And 2) = 2 Then
  cb_input1.Checked = True
  Else
  cb_input1.Checked = False
  End If
 Case "o"
  If (Val(Mid(receivestring, 8, 1)) And 1) = 1 Then
  cb_output0.Checked = True
  Else
  cb_output0.Checked = False
  End If
  If (Val(Mid(receivestring, 8, 1)) And 2) = 2 Then
  cb_output1.Checked = True
  Else
  cb_output1.Checked = False
  End If
 Case "c"
  Dim tabpos
  If Mid(receivestring, 8, 1) = "0" Then _
   tb_counter0.Text = Mid(receivestring, 10)
  If Mid(receivestring, 8, 1) = "1" Then _
   tb_counter1.Text = Mid(receivestring, 10)
  If Mid(receivestring, 8, 1) = ";" Then
   tabpos = InStr(9, receivestring, ";")
   tb_counter0.Text = Mid(receivestring, 9, tabpos - 9)
   tb_counter1.Text = Mid(receivestring, tabpos + 1, _
   Len(receivestring) - tabpos - 1)
  End If
 End Select
 End If
 End If
End Sub
```

Die Empfangsprozedur überprüft anhand des ersten Zeichens der Empfangsdaten, ob es um Input-, Output- oder Counter-Meldungen geht. Abhängig davon wird z.B. festgestellt, welcher Output welchen Status hat. Bei den Countern ist es sowohl möglich, einzelne Zählerwerte abzufragen, als auch alle Counter in einem Zug auszulesen. Die einzelnen Zählerstände werden dann dezimal mit Semikolon getrennt in einem String ausgegeben.

4. Polling

- Zyklisches Abfragen bestimmter Werte
  Um auch eine automatische Aktualisierung der Anzeige zu ermöglichen, wird ein Timer benutzt.
- In Abhängigkeit der Checkboxen für Output-, Input- und Counter-Polling werden die entsprechenden Informationen im eingestellten Intervall vom Web-IO abgerufen.

```
Private Sub timer_polling_Elapsed(ByVal sender As System.Object, _
ByVal e As System.Timers.ElapsedEventArgs) Handles timer_polling.Elapsed
 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
```

Das gewünschte Intervall kann in das entsprechende Textfeld eingegeben werden. Bei Änderung wird das Timer-Intervall dann automatisch angepasst.

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

Das Beispiel-Programm unterstützt alle gängigen Funktionen des Web-IO im Kommando-String Modus, optimiert für das Web-IO 2x Digital Input, 2x Digital Output PoE. Für die anderen Web-IO Modelle müssen ggf. Anpassungen am Programm vorgenommen werden. Weitere Programmbeispiele zur Socket-Programmierung finden Sie auf den Tool-Seiten zum Web-IO. Eine detaillierte Beschreibung zur Socketschnittstelle der Web-IO Digital Modelle finden Sie im Referenzhandbuch.

Programmbeispiel herunterladen

Sie haben noch kein Web-IO® und möchten das vorgestellte Beispiel einfach mal ausprobieren?
Kein Problem: Wir stellen Ihnen das Web-IO Digital 2xInput, 2xOutput gerne kostenlos für 30 Tage zur Verfügung. Einfach Musterbestellung ausfüllen, wir liefern das Web-IO zum Test auf offene Rechnung. Wenn Sie das Gerät innerhalb von 30 Tagen zurück schicken, schreiben wir die Rechnung komplett gut.

Zur Musterbestellung

W&T
www.WuT.de

Wir sind gerne persönlich für Sie da:

Wiesemann & Theis
GmbH
Porschestr. 12
42279 Wuppertal
Tel.: 0202/2680-110 (Mo-Fr. 8-17 Uhr)
Fax: 0202/2680-265
info@wut.de