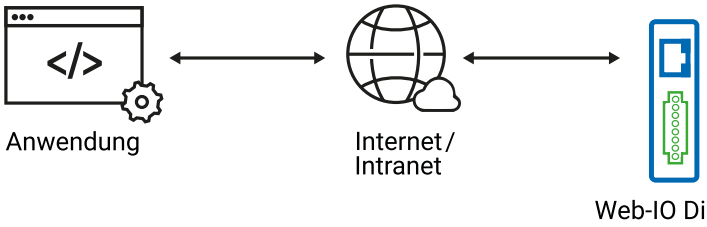
Tutorial zum Web-IO Digital:

Web-IO Digital mit Visual C# steuern und überwachen

Produktübersicht
Applikationsübersicht

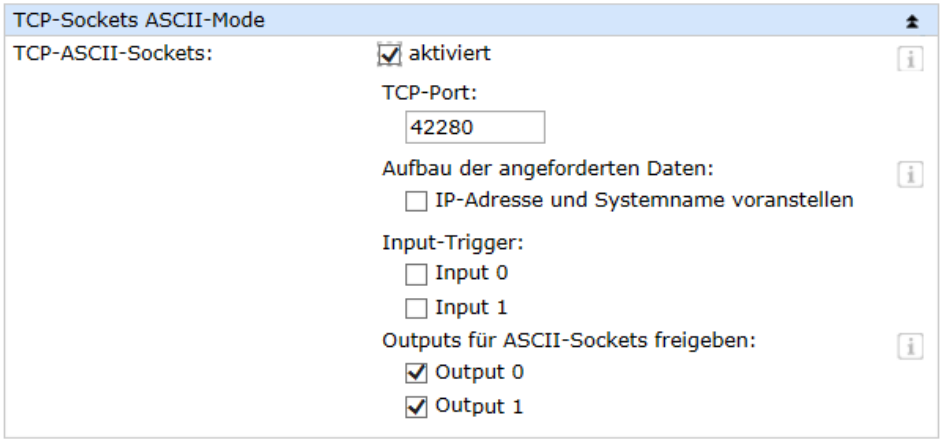
Für das Erstellen von Windows-Anwendungen war Visual C++ bis vor kurzem eine der meistgenutzten Entwicklungsplattformen. Inzwischen arbeiten immer mehr Programmierer mit dem .Net Framework und erstellen ihre Anwendungen in C# (C Sharp).



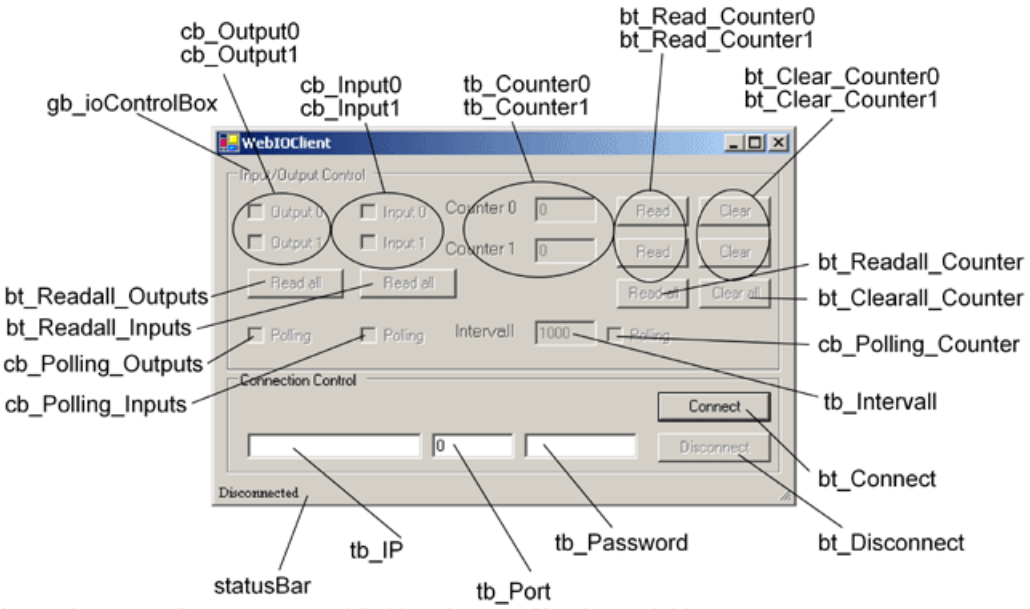
Mit dem folgenden C#-Programmbeispiel können Sie Ihr Web-IO Digital mit seinen Inputs und Outputs in einer Windows-Anwendung abbilden. Darüber hinaus können Sie die Outputs des Web-IO schalten.

Vorbereitungen

- Web-IO mit Spannung versorgen und IOs verdrahten
- Web-IO mit dem Netzwerk verbinden
- IP-Adressen vergeben
- Beim Web-IO im Bereich *Kommunikationswege >> Socket-API* die *TCP-ASCII-Sockets* aktivieren und die *Outputs zum Schalten freigeben*



Zusammenstellen der verschiedenen Bedienelemente und Anzeigeobjekte im Form



Importieren von Ressourcen und Deklaration von Membervariablen

Als erstes werden alle für die Netzwerkverbindung und die GUI(Graphical User Interface) benötigten Klassen importiert.

```
using System;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Text;
```

Anschließend werden die Komponenten der Applikation und wichtige Variablen für eine TCP-Verbindung als Membervariablen der Klasse deklariert und somit den Methoden der Klasse zugänglich gemacht.

```
  public class mainWindow : System.Windows.Forms.Form
  {
    private CheckBox cb_Output0;
    private CheckBox cb_Output1;
    private CheckBox cb_Input0;
    private CheckBox cb_Input1;
    private CheckBox cb_Polling_Counter;
    private CheckBox cb_Polling_Outputs;
    private CheckBox cb_Polling_Inputs;
    private Button bt_Readall_Outputs;
    private Button bt_Readall_Inputs;
    private Button bt_Clear_Counter0;
    private Button bt_Clear_Counter1;
    private Button bt_Clearall_Counter;
    private Button bt_Read_Counter0;
    private Button bt_Read_Counter1;
    private Button bt_Readall_Counter;
    private Label lb_Counter0;
    private Label lb_Counter1;
    private Label lb_Intervall;
    private TextBox tb_Counter0;
    private TextBox tb_Counter1;
    private TextBox tb_Intervall;
    private Button bt_Connect;
    private Button bt_Disconnect;
    private TextBox tb_Password;
    private TextBox tb_Port;
    private TextBox tb_IP;
    private GroupBox gb_ioControlBox;
    private GroupBox gb_conControlBox;
    private StatusBar statusBar;

    private Socket client;
    private int intervall;
    private byte[] buffer = new byte[256];
    private System.Windows.Forms.Timer counter;
    private System.Windows.Forms.Timer outputs;
    private Label label2;
    private Label label1;
    private Label label3;
    private System.Windows.Forms.Timer inputs;
```

Programmstart

Einrichten der Bedienelemente
Die Gruppe mit den Bedienelementen für das Web-IO wird zunächst für die Bedienung gesperrt. Sobald eine Verbindung zustande kommt, werden alle Elemente freigeschaltet, welche eine sinnvolle Ausführung besitzen.

Der Name des jeweiligen Bedienelementes ist dem Kontext nach vom Element selbst abgeleitet. Die beiden ersten Zeichen im Namen stehen für den Typ des Elementes (cb -> Checkbox, bt -> Button, gb -> Groupbox und tb-> TextBox).

```
  public mainWindow()
  {
    InitializeComponent();
    gb_ioControlBox.Enabled = false;
    bt_Disconnect.Enabled = false;
    cb_Input0.Enabled = false;
    cb_Input1.Enabled = false;
    tb_Counter0.Enabled = false;
    tb_Counter1.Enabled = false;
  }
```

Die Verbindungskontrolle

Einleiten der Verbindung
Nach Eingabe der IP-Adresse des Web-IO in das Textfeld *tb_IP* und dem Port 42280 in das Textfeld *tb_Port* kann durch Betätigung des Buttons *bt_Connect* eine Verbindung aufgebaut werden. Falls keine IP-Adresse oder kein Port eingetragen wird, erfolgt eine Meldung durch die Applikation in der Statusleiste.

Verbindungsaufbau
Um nun eine TCP-Verbindung aufbauen zu können, wird ein neues Socket angelegt und initialisiert. IP-Adresse und Portnummer werden zu einem IP-Endpoint zusammengefasst. Mit dem *BeginConnect*-Aufruf wird der IP-Endpoint übergeben.

Damit das Programm asynchron arbeiten kann, wartet es nicht auf Ereignisse, sondern arbeitet mit Callback-Routinen. Callback-Methoden werden initialisiert, wenn ein Prozess gestartet wird und aufgerufen, wenn das entsprechende Ereignis eintritt, also z. B. beim Verbindungsaufbau, beim Senden oder Empfangen.

```
private void bt_Connect_Click(object sender, System.EventArgs e)
{
  try
  {
    if((tb_IP.Text != "") && (tb_Port.Text != ""))
    {
      client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
      IPEndPoint ipep = new IPEndPoint(IPAddress.Parse(tb_IP.Text), int.Parse(tb_Port.Text));
      client.BeginConnect(ipep, new AsyncCallback(connectCallback), client);
    }
    else
      statusBar.Text = "IP and Port needed!";
  }
 catch(SocketException)
 { statusBar.Text = "Connection not possible!"; }
}
```

Im Folgenden wird die Callback-Routine dargestellt, die bei Zustandekommen der Verbindung aufgerufen wird. Die Applikation beginnt sofort damit, sich empfangsbereit zu stellen. Dafür wird die Callback-Routine "receiveCallback" initialisiert, welche weiter unten näher erläutert wird.

```
private void connectCallback(IAsyncResult ar)
{
  try
  {
    client = (Socket) ar.AsyncState;
    client.EndConnect(ar);
    connectupdatedisplay();
    intervall = 1000;
    client.BeginReceive(buffer, 0, 255, SocketFlags.None, new AsyncCallback(receiveCallback), cli
  }
  catch(Exception)
  { disconnect();
  }
}
```

Außerdem werden über Aufruf der Prozedur `connectupdatedisplay()` alle benötigten Bedienelemente der Applikation freigschalten und der Connect-Button deaktiviert.

In der aktuellen Version von C# ist es nicht mehr erlaubt, von einem Thread (hier die Callback-Funktion) auf grafische Elemente eines anderen Threads (Form) zuzugreifen.

Eine Lösung bietet hier die Benutzung von Delegaten und Invoke.

Im Folgenden wird die Callback-Routine dargestellt, die bei Zustandekommen der Verbindung aufgerufen wird. Die Applikation beginnt sofort damit, sich empfangsbereit zu stellen. Dafür wird die Callback-Routine "receiveCallback" initialisiert, welche weiter unten näher erläutert wird.

```
delegate void delegateSub();

private void connectupdatedisplay()
{
  if (InvokeRequired)
  {
    BeginInvoke(new delegateSub(connectupdatedisplay), new object[] { });
    return;
  }
  statusBar.Text = "Connected!";
  gb_ioControlBox.Enabled = true;
  bt_Disconnect.Enabled = true;
  bt_Connect.Enabled = false;
}
```

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

```
private void bt_Disconnect_Click(object sender, System.EventArgs e)
{
  try
  {
    client.Shutdown(SocketShutdown.Both);
    client.Close();
    disconnectupdatedisplay();
  }
  catch (Exception)
  {
    statusBar.Text = "Not able to disconnect!";
  }
}
```

Auch wenn die Verbindung vom Web-IO abgebaut wird, soll die Anwendung das verwendete Socket schließen und es sollen alle Elemente wieder in ihre Ausgangsstellung gebracht werden. Es soll dann nicht mehr möglich sein, den Disconnect-Button zu betätigen.

```
private void disconnect()
{
  try
  {
    client.Shutdown(SocketShutdown.Both);
    client.Close();
    disconnectupdatedisplay();
  }
  catch (Exception)
  {
     statusBar.Text = "Not able to disconnect!";
  }
}
```

Für einen Thread-sicheren Zugriff wurde das Zurücksetzen der Formular-Elemente in die Prozedur`disconnectupdatedisplay()` ausgelagert.

```
private void disconnectupdatedisplay()
{
  if (InvokeRequired)
  {
    BeginInvoke(new delegateSub(disconnectupdatedisplay), new object[] { });
    return;
  }
  statusBar.Text = "Disconnected!";
  gb_ioControlBox.Enabled = false;
  bt_Disconnect.Enabled = false;
  bt_Connect.Enabled = true;
}
```

Nun ist die Verbindung wieder beendet und die Applikation auf ihren Ausgangszustand zurückgesetzt.

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.

Kommandos senden
Beim Senden einer Nachricht an das Web-IO wird genauso wie beim Empfang eine Callback-Routine aufgerufen.

```
     private void sendCallback(IAsyncResult ar)
private void sendCallback(IAsyncResult ar)
{
  try
  {
    Socket tmp_client = (Socket) ar.AsyncState;
    int bytessend = tmp_client.EndSend(ar);
  }
  catch(Exception)
  {
    statusBar.Text = "Error while sending";
  }
}
```

Setzen der Outputs
Die Outputs des Web-IO können mit Hilfe der beiden Checkboxen cb_Output0 und cb_Output1 geschaltet werden. Die Checkbox löst, wenn sie geklickt wird, eine Aktion aus. Abhängig davon, ob die Checkbox bereits gesetzt ist, wird der Output entweder auf ON oder auf OFF gesetzt.

```
private void cb_Output0_CheckedChanged(object sender, System.EventArgs e)
{
  if(cb_Output0.Checked)
    send("GET /outputaccess0?PW=" + tb_Password.Text + "&State=ON&");
  else
    send("GET /outputaccess0?PW=" + tb_Password.Text + "&State=OFF&");
}

private void cb_Output1_CheckedChanged(object sender, System.EventArgs e)
{
  if(cb_Output1.Checked)
    send("GET /outputaccess1?PW=" + tb_Password.Text + "&State=ON&");
  else
    send("GET /outputaccess1?PW=" + tb_Password.Text + "&State=OFF&");
}
```

Output/Input-Status abfragen

```
private void bt_Readall_Outputs_Click(object sender, System.EventArgs e)
{
  send("GET /output?PW=" + tb_Password.Text + "&");
}

private void bt_Readall_Inputs_Click(object sender, System.EventArgs e)
{
  send("GET /input?PW=" + tb_Password.Text + "&");
}
```

Counter abfragen oder löschen
Die folgende Methode sendet eine Anfrage an einen bestimmten Counter und fordert eine Antwort mit dem aktuellen Stand des Zählers bzw. löscht den Zählerstand eines Counters.

```
private void bt_Read_Counter0_Click(object sender, System.EventArgs e)
{
  send("GET /counter0?PW=" + tb_Password.Text + "&");
}

private void bt_Read_Counter1_Click(object sender, System.EventArgs e)
{
  send("GET /counter1?PW=" + tb_Password.Text + "&");
}

private void bt_Clear_Counter0_Click(object sender, System.EventArgs e)
{
  send("GET /counterclear0?PW=" + tb_Password.Text + "&");
}

private void bt_Clear_Counter1_Click(object sender, System.EventArgs e)
{
  send("GET /counterclear1?PW=" + tb_Password.Text + "&");
}
```

Natürlich können alle Counter-Zustände auch mit einem Kommando abgefragt bzw. gelöscht werden.

```
private void bt_Readall_Counter_Click(object sender, System.EventArgs e)
{
  send("GET /counter?PW=" + tb_Password.Text + "&");
}

private void bt_Clearall_Counter_Click(object sender, System.EventArgs e)
{
  send("GET /counterclear?PW=" + tb_Password.Text + "&");
}
```

Datenempfang vom Web-IO und Auswertung

Aufbau der Empfangsdaten

- Alle Kommandos und Anfragen an das Web-IO werden mit einem Antwort-String quittiert. Die Antworten haben je nach Type einen spezifischen Aufbau:
- Für die Outputs: output;<Binärwert des Output-Status im hexadezimalen Format>
- Für einen speziellen Output: outputx;<ON oder OFF>
- Für die Inputs: input;<Binärwert des Output-Status im hexadezimalen Format>
- Für einen speziellen Input: inputx;<ON oder OFF>
- Dann gibt es noch den Antwort-String für einen Counter der folgendermaßen aussieht.
- 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.

Datenempfang
In unserer Applikation wird zum Empfang einer solchen Nachricht die Methode `receiveCallback()` aufgerufen. Das Besondere an diese Methode ist der eventgesteuerte Aufruf. Der erfolgt, sobald das Web-IO Daten an die Anwendung sendet.

```
private void receiveCallback(IAsyncResult ar)
{
  int bytesRead;
  string rcv = string.Empty;
  try
  { bytesRead = client.EndReceive(ar);
    rcv = Encoding.ASCII.GetString(buffer);
    client.BeginReceive(buffer, 0, 255, SocketFlags.None, new AsyncCallback(receiveCallback), cli
  }
  catch (Exception)
  {
    bytesRead = 0;
  }
  if (bytesRead == 0)
  {
    if (client.Connected)
    {
      disconnect();
    }
  }
  else if (rcv != null)
  {
    IOupdatedisplay(rcv.Substring(0, bytesRead - 1));
  }
}
```

Auswertung der Empfangsdaten
Der Antwort-String wird gelesen und an die Thread-sichere Prozedur `IOupdatedisplay` übergeben.

```
delegate void delegateIOSub(string rcv);

private void IOupdatedisplay(string rcv)
{
  if (InvokeRequired)
  {
    BeginInvoke(new delegateIOSub(IOupdatedisplay), new object[] { rcv });
    return;
  }
  if (rcv[0] == 'o')
  {
    int i = Int32.Parse(rcv.Substring(7), System.Globalization.NumberStyles.HexNumber);
    if((i & 1) == 1)
      cb_Output0.Checked = true;
    else
      cb_Output0.Checked = false;
    if((i & 2) == 2)
      cb_Output1.Checked = true;
    else
      cb_Output1.Checked = false;
  }
  if(rcv[0] == 'i')
  {
    int i = Int32.Parse(rcv.Substring(6), System.Globalization.NumberStyles.HexNumber);
    if ((i & 1) == 1)
      cb_Input0.Checked = true;
    else
      cb_Input0.Checked = false;
    if((i & 2) == 2)
      cb_Input1.Checked = true;
    else
      cb_Input1.Checked = false;
  }
  if(rcv[0] == 'c')
  {
    if(rcv[7] == '0')
      tb_Counter0.Text = rcv.Substring(9);
    if(rcv[7] == '1')
      tb_Counter1.Text = rcv.Substring(9);
    if(rcv[7] == ';')
    readAndClearCounter(rcv.Substring(8));
  }
}
```

Da alle Counter-Zustände mit einem Kommando gelesen oder zurückgesetzt werden können, muss noch eine Methode implementiert werden, welche den Antwort-String des Web-IO so verarbeitet, dass jedem Counter in der Applikation seinen spezifischen Zählerstand zuordnet.

```
private void readAndClearCounter(string data)
{
  int j = 0;
  string[] counter = new string[2];
  for(int i = 0; i < data.Length; i++)
  {
    if((data[i].CompareTo(';')) == 0)
      j++;
    else
      counter[j] += data[i].ToString();
  }
  tb_Counter0.Text = counter[0];
  tb_Counter1.Text = counter[1];
}
```

7. Polling

Zyklisches Abfragen bestimmter Werte
Es ist wünschenswert, dass sich der Status einer einzelnen Komponente von selbst aktualisiert und somit die Applikation immer den aktuellen Stand aufweist. Dazu wird in diesem Progamm ein Timer verwendet, der in einem vom User betimmten Zeitintervall zyklisch Abfragen an das Web-IO schickt.

Das Zeitintervall kann im Feld *tb_Intervall* festgelegt werden.

Natürlich wird auch abgefangen, falls der Nutzer eine unsinnige Angabe, wie z. B. einen negativen Zeitwert, macht.

```
private void tb_Intervall_TextChanged(object sender, System.EventArgs e)
{
  try
  {
    if(Convert.ToInt32(tb_Intervall.Text) > 0)
    {
      intervall = Convert.ToInt32(tb_Intervall.Text);
      statusBar.Text = "New range: " + intervall.ToString() + " ms!";
    }
    else
      statusBar.Text = "Only positiv Integer allowed!";
  }
  catch(Exception)
  {
   statusBar.Text = "Only positiv Integer allowed!";
  }
}
```

Um auch das zyklische Abfragen der Zustände des Web-IO durchzuführen, was auch als Polling bezeichnet wird, besteht die Auswahlmöglichkeit zwischen dem Polling der Outputs, der Inputs oder der Counter.

Je Polling-Variante wird je ein eigener Timer initialisiert.

Betätigt man die Checkbox cb_Polling_Outputs, wird das Polling auf die Outputs angewendet. Dazu wird der entsprechende Timer initialisiert.

```
private void cb_Polling_Outputs_CheckedChanged(object sender, System.EventArgs e)
{
  if(cb_Polling_Outputs.Checked)
  {
    outputs = new System.Windows.Forms.Timer();
    outputs.Interval = intervall;
    outputs.Start();
    outputs.Tick += new EventHandler(timer_handler);
  }
  else
    outputs.Stop();
}
```

Das gleiche gilt für Inputs und Counter.

```
private void cb_Polling_Inputs_CheckedChanged(object sender, System.EventArgs e)
{
  if(cb_Polling_Inputs.Checked)
  {
    inputs = new System.Windows.Forms.Timer();
    inputs.Interval = intervall;
    inputs.Start();
    inputs.Tick += new EventHandler(timer_handler);
  }
  else
    inputs.Stop();
}

private void cb_Polling_Counter_CheckedChanged(object sender, System.EventArgs e)
{
  if(cb_Polling_Counter.Checked)
  {
    counter = new System.Windows.Forms.Timer();
    counter.Interval = intervall;
    counter.Start();
    counter.Tick += new EventHandler(timer_handler);
  }
  else
    counter.Stop();
}
```

In dieser Methode wird das jeweilige Event des Timers, der sich gerade meldet, gefangen und einer bestimmten Aktion zugewiesen.

```
private void timer_handler(object sender, System.EventArgs e)
{
  if(sender == counter) bt_Readall_Counter_Click(sender, e);
  if(sender == outputs) bt_Readall_Outputs_Click(sender, e);
  if(sender == inputs) bt_Readall_Inputs_Click(sender, e);
}
```

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

Programmbeispiel herunterladen

Produkte

Web-IO 4.0 Digital 2xIn, 2xOut

Bei Bedarf auch über PoE zu versorgen

Web-IO 4.0 Digital 12xIn, 12xOut

12x Eingänge, 12x Ausgänge

Weitere Web-IOs

Alle W&T Web-IO Digital 24V

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