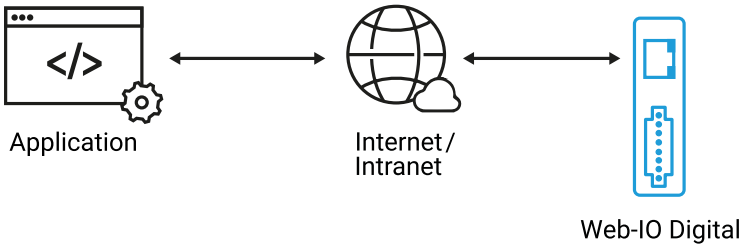
Application for the Web-IO Digital:

Control and monitor Web-IO Digital with Delphi 2005

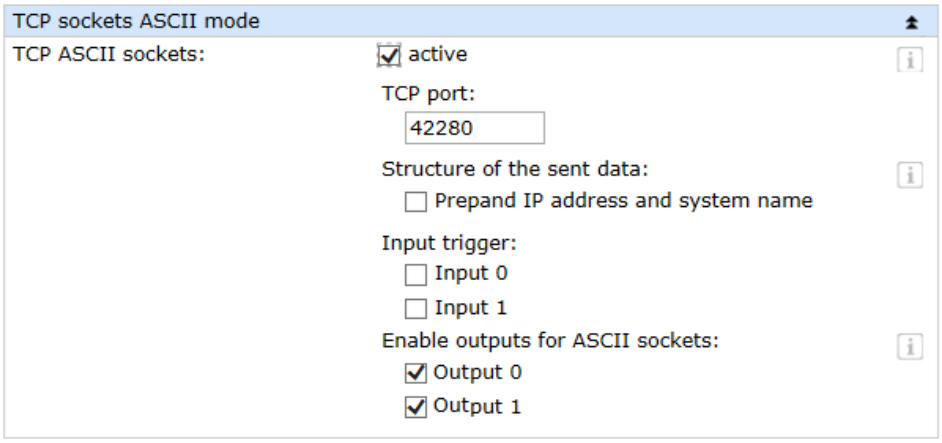
As the successor to Borland Delphi, Delphi 2005 builds on the Microsoft.net Framework. Delphi 2005 (Delphi.net) offers everything needed to program TCP/IP applications. This makes Delphi 2005 a favorite tool for creating applications that communicate with the Web-IO Digital. Additional drivers or DLLs are not required.



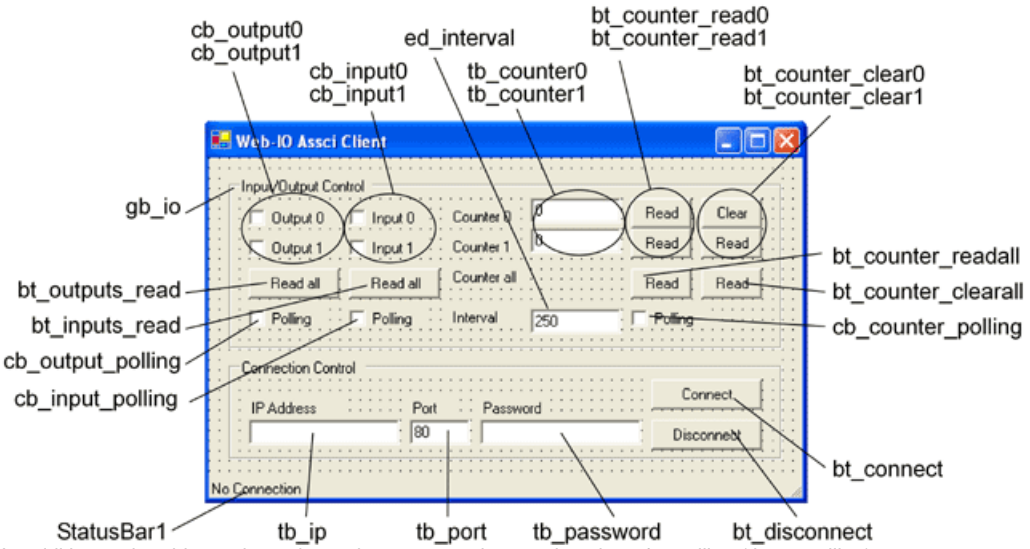
Using the following program example you can represent your Web-IO Digital with its inputs and outputs in a Windows application. You can also switch the Web-IO outputs.

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 Delphi 2005 is just as simple to accomplish as with the older Delphi versions, constructing network communication under Delphi.net is somewhat more difficult. This is due primarily to the fact that the ClientSocket control element for framework applications is no longer provided. Instead, you must specify the namespace for the socket classes in the uses area in order to have network access.

```
uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data, System.Net,
  System.Net.Sockets, System.IO, System.Text, System.Configuration;
```

Some procedures which are not created by Delphi itself must be declared in the header of the source text. (More on the content of these procedures later.)

```
private
  { Private-Deklarationen }
public
  constructor Create;
  procedure callback_connect(ari : IAsyncResult);
  procedure callback_readdata(ari : IAsyncResult);
  procedure sendcommand(sendstring : String);
  procedure close();
end;
```

Furthermore, the socket on which communication is supposed to take place needs to be created and a buffer for the input data and AsyncResult variable must be defined.

```
var
  TCP_Client : Socket;
  ReceiveBuffer : array [0..512] of byte;
  ar : IAsyncResult;
```

Connection control

Establishing the connection

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

```
procedure TWinForm.bt_connect_Click(sender: System.Object; e: System.EventArgs);
var
  WebioEP : IPEndPoint;
begin
  if (tb_ip.Text <> '') and (tb_port.Text <> '') then
  begin
    bt_connect.Enabled := false;
    TCP_Client := Socket.Create (AddressFamily.InterNetwork, SocketType.Stream,ProtocolType.TCP);
    StatusBar1.Text := 'Try to connect to ' + tb_ip.Text;
    timer_polling.Enabled := true;
    try
      TCP_Client.BeginConnect(WebioEP,callback_connect, TCP_Client);
    except
      on ex : Exception
      do
      begin
        StatusBar1.Text := 'ERROR on connecting';
        close();
      end;
    end;
  end;
end;
```

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 a connection is made with the Web-IO, the callback procedure is run. The status line shows that the connection has been opened, the operating elements are enabled for use and the Disconnect button becomes operable. The Connect button is blocked for operation. In addition a reference to a callback routine for data reception is created.

```
procedure TWinForm.callback_connect(ari : IAsyncResult);
begin
  gb_io.enabled := true;
  bt_disconnect.Enabled := True;
  statusbar1.Text := 'Connected to' + tb_ip.Text;
  try
    TCP_Client.EndConnect(ari);
    TCP_Client.BeginReceive(ReceiveBuffer, 0, 512, SocketFlags.None, callback_readdata, ari.AsyncState);
  except
    on ex : Exception
    do
    begin
      StatusBar1.Text := 'ERROR on connect: ' + ex.ToString;
      close();
    end;
  end;
end;
```

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

In this case a corresponding procedure is invoked.

```
procedure TWinForm.close();
begin
  timer_polling.Enabled := False;
  Try
    TCP_client.EndReceive(ar);
  except
  end;
  Try
    TCP_client.Shutdown(SocketShutdown.Both);
  except
  end;
  Try
    TCP_client.Close()
  except
  end;
  bt_connect.Enabled := True;
  gb_io.Enabled := False;
  bt_disconnect.Enabled := False;
  StatusBar1.Text := 'no connection';
end;

procedure TWinForm.bt_disconnect_Click(sender: System.Object; e: System.EventArgs);
begin
  close();
end;
```

Connection error
All actions pertaining to TCP/IP communication are executed within the Try instruction. If errors occur, the CloseConnection procedure is likewise invoked.

Operation and communication from the client side

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.

```
procedure TWinForm.sendcommand(sendstring: String);
var
  SendBuffer : array[0..512] of byte;
begin
  SendBuffer := System.Text.Encoding.ASCII.GetBytes(sendstring);
  try
    TCP_Client.Send(SendBuffer, 0, sendstring.Length, SocketFlags.None );
  except
    on ex : Exception
    do
    begin
      StatusBar1.Text := 'ERROR on send: ' + ex.ToString;
      close();
    end;
  end;
end;
```

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.

```
procedure TWinForm.cb_output0_MouseUp(sender: System.Object; e: System.Windows.Forms.MouseEventArgs);
begin
  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;

procedure TWinForm.cb_output1_MouseUp(sender: System.Object; e: System.Windows.Forms.MouseEventArgs);
begin
  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;
```

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

```
procedure TWinForm.bt_outputs_read_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /output?PW='+ tb_password.Text + '&');
end;

procedure TWinForm.bt_inputs_read_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /input?PW='+ tb_password.Text + '&');
end;
```

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

```
procedure TWinForm.bt_counter_read0_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counter0?PW='+ tb_password.Text + '&');
end;

procedure TWinForm.bt_counter_read1_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counter0?PW='+ tb_password.Text + '&');
end;

procedure TWinForm.bt_counter_clear0_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counterclear0?PW='+ tb_password.Text + '&');
end;

procedure TWinForm.bt_counter_clear1_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counterclear1?PW='+ tb_password.Text + '&');
end;
```

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

```
procedure TWinForm.bt_counter_readall_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counter?PW='+ tb_password.Text + '&');
end;

procedure TWinForm.bt_counter_clearall_Click(sender: System.Object; e: System.EventArgs);
begin
  sendcommand('GET /counterclear?PW='+ tb_password.Text + '&');
end;
```

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

```
procedure TWinForm.callback_readdata(ari : IAsyncResult);
var
  bytesread : Integer;
  receivestring : string;
begin
  try
    bytesread := TCP_Client.EndReceive(ari)
  except
  end;
    if Bytesread = 0 then
      close()
    else
    begin
      receivestring := System.Text.Encoding.ASCII.GetString(ReceiveBuffer,0,bytesread);
      try
        TCP_Client.BeginReceive(ReceiveBuffer,0, 512,SocketFlags.None, callback_readdata,ari.AsyncState);
      except
        on ex : Exception
        do
        begin
          statusbar1.Text :='ERROR on read: ' + ex.ToString;
          close();
        end;
    end;
    if receivestring[1] = 'o' then
    begin
      if(convert.ToInt16(receivestring[8]) and 1) = 1 then
        cb_output0.Checked := true
      else
        cb_output0.Checked := false;
      if(convert.ToInt16(receivestring[8]) and 2) = 2 then
        cb_output1.Checked := true
      else
        cb_output1.Checked := false;
    end;
    if receivestring[1] = 'i' then
    begin
      if(convert.ToInt16(receivestring[7]) and 1) = 1 then
        cb_input0.Checked := true
      else
        cb_input0.Checked := false;
      if(convert.ToInt16(receivestring[7]) and 2) = 2 then
        cb_input1.Checked := true
      else
        cb_input1.Checked := false;
    end;
    if receivestring[1] = 'c' then
    begin
      if copy(ReceiveString, 8, 1) = '0' then
        tb_counter0.Text := copy(ReceiveString, 10,length(ReceiveString)-10);
      if copy(ReceiveString, 8, 1) = '1' then
        tb_counter1.Text := copy(ReceiveString, 10,length(ReceiveString)-10);
      if copy(ReceiveString, 8, 1) = ';' then
      begin
        ReceiveString[8] := ' ';
        tb_counter0.Text := copy(ReceiveString, 9, pos(';',ReceiveString)-9);
        tb_counter1.Text := copy(ReceiveString,pos(';',ReceiveString)+1,length(ReceiveString)-pos(';',ReceiveString)-1);
      end;
    end;
  end;
end;
```

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.

```
procedure TWinForm.timer_polling_Tick1(sender: System.Object; e: System.EventArgs);
begin
  if TCP_Client.Connected then
  begin
    if cb_output_polling.Checked then
      sendcommand('GET/output?PW=' + tb_password.Text + '&');
    if cb_input_polling.Checked then
      sendcommand('GET/input?PW=' + tb_password.Text + '&');
    if cb_counter_polling.Checked then
      sendcommand('GET/counter?PW=' + tb_password.Text + '&');
  end;
end;
```

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

```
procedure TWinForm.tb_interval_TextChanged(sender: System.Object; e: System.EventArgs);
begin
  timer_polling.Interval := convert.ToInt16(tb_interval.Text);
end;
```

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

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