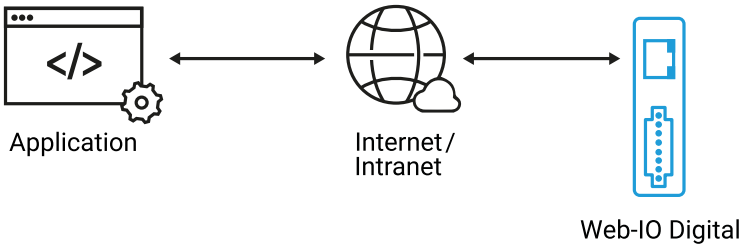
Application for the Web-IO Digital:

Control and monitor Web-IO Digital with Delphi

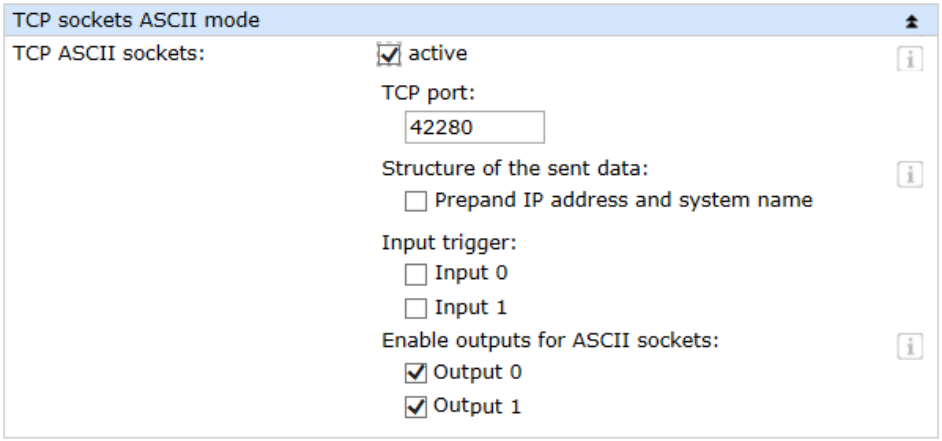
As an easy to learn higher language Delphi offers everything you need for programming TCP/IP applications. This also makes Delphi a favorite tool for creating applications that communicate with the Web-IO Digital. Additional drivers or DLLs are not required.



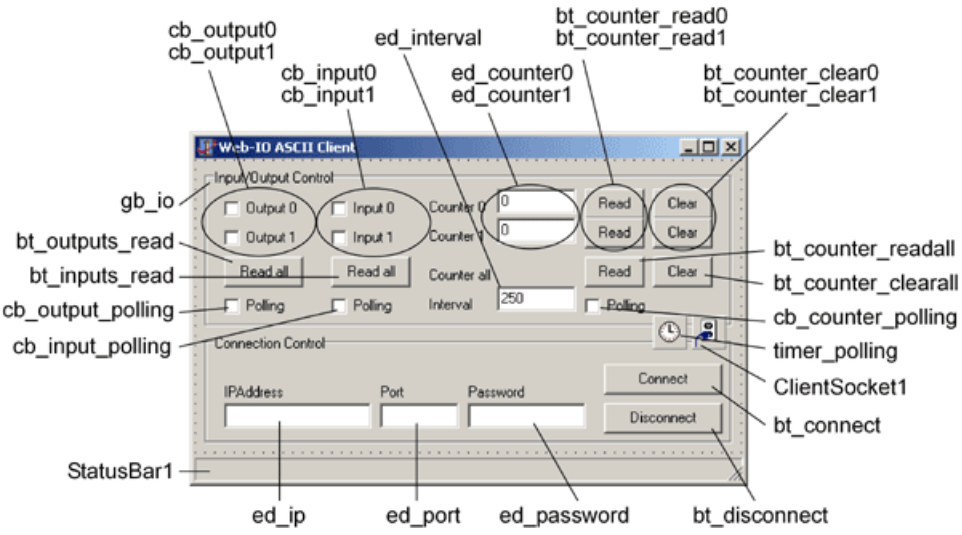
Using the following program example you can represent your Web-IO Digital with its inputs and outputs in a Windows application. You can also switch the Web-IO outputs.

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 Delphi form



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

Setting up the operating elements
At first the group with the operating elements for the Web-IO is blocked from use. The status line indicates that there is not yet a connection.

```
procedure Twebio_ascii_client.FormCreate(Sender:TObject);
begin
  StatusBar1.SimpleText := 'No Connection';
  bt_disconnect.Enabled := False;
  gb_io.Enabled := False;
end;
```

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.

```
procedure Twebio_ascii_client.bt_connectClick(Sender:TObject);
begin
  if ed_ip.Text <> '' then
  begin
    ClientSocket1.Host := ed_ip.Text;
    ClientSocket1.Port := strtoint(ed_port.Text);
    ClientSocket1.Active := True;
    end;
end;
```

Connection is made
As soon as the Web-IO accepts the connection, the ClientSocket control element carries out the corresponding procedure. The status line indicates that the connection has been established, the control elements are enabled for use and the Disconnect button is active again.

```
procedure Twebio_ascii_client.ClientSocket1Connect(Sender:TObject;Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText := 'Connected to ' + ed_ip.Text;
  bt_disconnect.Enabled := True;
  gb_io.Enabled := True;
end;
```

Opening the connection
The Delphi ClientSocket control is used for executing TCP/IP handling.
This control allows procedures to be specified and started when various connection states occur, here for example for attempting to open a connection. The procedure enters a corresponding message in the status line and deactivates the bt_connect button so that the user cannot start another connection attempt while the first one is still running.

```
procedure Twebio_ascii_client.ClientSocket1Connecting(Sender:TObject;Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText := 'Try to connect to ' + ed_ip.Text;
  bt_connect.Enabled := False;
end;
```

Connection is made
As soon as the Web-IO accepts the connection, the ClientSocket control element carries out the corresponding procedure. The status line indicates that the connection has been established, the control elements are enabled for use and the Disconnect button is active again.

```
procedure Twebio_ascii_client.ClientSocket1Connect(Sender:TObject;Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText := 'Connected to ' + ed_ip.Text;
  bt_disconnect.Enabled := True;
  gb_io.Enabled := True;
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.

```
procedure Twebio_ascii_client.bt_disconnectClick(Sender:TObject);
begin
  ClientSocket1.Active := False;
end;
```

Here again the ClientSocket control element invokes a corresponding procedure

```
procedure Twebio_ascii_client.ClientSocket1Disconnect(Sender:TObject;Socket: TCustomWinSocket);
begin
  ClientSocket1.Active := False;
   StatusBar1.SimpleText := 'No Connection';
  bt_connect.Enabled := True;
  bt_disconnect.Enabled := False;
  gb_io.Enabled := False;
end;
```

Connection error

Also in case of a connection error the ClientSocket control element carries out a corresponding procedure which is essentially like the Disconnect procedure. The status line also indicates the Winsock error number and the ErrorCode is then set to 0 so that no runtime error occurs.

```
procedure Twebio_ascii_client.ClientSocket1Error(Sender:TObject;Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;va
begin
  ClientSocket1.Active := False;
  StatusBar1.SimpleText := 'Error ' + inttostr(ErrorCode) + ' - No Connection';
  ErrorCode := 0;
  bt_connect.Enabled := True;
  bt_disconnect.Enabled := False;
  gb_io.Enabled := False;
end;
```

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

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 Mouse Up, i.e. releasing the output check box, is recorded, 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 Twebio_ascii_client.cb_outputMouseUp(Sender:TObject;Button: TMouseButton; Shift: TShiftState; X, Y: Integer)
begin
  if sender = cb_output0 then
    if cb_output0.Checked then
      ClientSocket1.Socket.SendText('GET/outputaccess0?PW=' + ed_password.Text + '&State=ON&')
    else
      ClientSocket1.Socket.SendText('GET/outputaccess0?PW=' + ed_password.Text + '&State=OFF&')
  else
    if cb_output1.Checked then
      ClientSocket1.Socket.SendText('GET/outputaccess1?PW=' + ed_password.Text + '&State=ON&')
    else
      ClientSocket1.Socket.SendText('GET/outputaccess1?PW=' + ed_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 Twebio_ascii_client.bt_outputs_readClick(Sender:TObject);
begin
  ClientSocket1.Socket.SendText('GET/output?PW=' + ed_password.Text + '&');
end;

procedure Twebio_ascii_client.bt_inputs_readClick(Sender:TObject);
begin
  ClientSocket1.Socket.SendText('GET/input?PW=' + ed_password.Text + '&');
end;
```

Read clear counters

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

```
procedure Twebio_ascii_client.bt_counter_readClick(Sender:TObject);
begin
  if sender = bt_counter_read0 then
    ClientSocket1.Socket.SendText('GET/counter0?PW=' + ed_password.Text + '&')
  else
    ClientSocket1.Socket.SendText('GET/counter1?PW=' + ed_password.Text + '&');
end;

procedure Twebio_ascii_client.bt_counter_clearClick(Sender:TObject);
begin
  if sender = bt_counter_clear0 then
    ClientSocket1.Socket.SendText('GET /counterclear0?PW='+ ed_password.Text + '&')
  else
    ClientSocket1.Socket.SendText('GET /counterclear1?PW='+ ed_password.Text + '&');
end;
```

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

```
procedure Twebio_ascii_client.bt_counter_readallClick(Sender:TObject);
begin
  ClientSocket1.Socket.SendText('GET/counter?PW=' + ed_password.Text + '&')
end;

procedure Twebio_ascii_client.bt_counter_clearallClick(Sender:TObject);
begin
  ClientSocket1.Socket.SendText('GET /counterclear?PW=' + ed_password.Text + '&')
end;
```

Data reception 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.
- If data are received from the ClientSocket control element, the latter invokes the corresponding procedure

```
procedure Twebio_ascii_client.ClientSocket1Read(Sender:TObject;Socket: TCustomWinSocket);
var
  ReceiveString : String;
  OutputValue : word;
  InputValue : word;
begin
  ReceiveString := ClientSocket1.Socket.ReceiveText;
  if Receivestring[1] = 'o' then
  begin
    OutputValue := HextoInt(copy(ReceiveString,pos(';',ReceiveString)+1,length(ReceiveString)-pos(';',ReceiveString)-1));
    if OutputValue and 1 = 1 then
      cb_output0.Checked := True
    else
      cb_output0.Checked := False;
    if OutputValue and 2 = 2 then
      cb_output1.Checked := True
    else
      cb_output1.Checked := False;
  end;
  if Receivestring[1] = 'i' then
  begin
    InputValue := HextoInt(copy(ReceiveString,pos(';',ReceiveString)+1,length(ReceiveString)-pos(';',ReceiveString)-1));
    if InputValue and 1 = 1 then
      cb_input0.Checked := True
    else
      cb_input0.Checked := False;
    if InputValue 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
      ed_counter0.Text := copy(ReceiveString,10,length(ReceiveString)-10);
    if copy(ReceiveString, 8, 1) = '1'then
      ed_counter1.Text := copy(ReceiveString,10,length(ReceiveString)-10);
    if copy(ReceiveString, 8, 1) = ';'then
    begin
      ReceiveString[8] := ' ';
      ed_counter0.Text := copy(ReceiveString,9, pos(';',ReceiveString)-9);
      ed_counter1.Text := copy(ReceiveString, pos(';',ReceiveString)+1,length(ReceiveString)-pos(';',ReceiveString)-1);
    end;
  end;
end;
```

The receiving procedure uses the first character of the receive data to check whether this is an input, output or counter message. Depending on this it is determined for example which output has which status. Since the output status is sent in hexadecimal format, a conversion into an integer value is first required in order to calculate subsequent steps. For the conversion a corresponding function is incorporated in the program. In the case of the counters it is also possible to read individual counter values or to read out all the counters at once. The individual counter states are then output in decimal format in a semicolon delimited string.

```
function HexToInt(HexString: String) : longWord;
var
  CharCount : integer;
  HexCharValue : longWord;
  HexValue : longWord;
begin
  HexValue := 0;
  HexString := UpperCase(HexString);
  for CharCount := 1 to length(HexString) do
  begin
    HexCharValue := ord(HexString[CharCount]);
    if HexCharValue > 57 then
      HexCharValue := HexCharValue -55
    else
      HexCharValue := HexCharValue -48;
    HexCharValue := HexCharValue shl ((length(HexString)- CharCount) * 4);
    HexValue := HexValue or HexCharValue;
  end;
  HexToInt := HexValue;
end;
```

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 Twebio_ascii_client.timer_pollingTimer(Sender:TObject);
begin
  if ClientSocket1.Active and cb_output_polling.Checked then
    ClientSocket1.Socket.SendText('GET /output?PW=' + ed_password.Text + '&');
  if ClientSocket1.Active and cb_input_polling.Checked then
    ClientSocket1.Socket.SendText('GET /input?PW=' + ed_password.Text + '&');
  if ClientSocket1.Active and cb_counter_polling.Checked then
    ClientSocket1.Socket.SendText('GET /counter?PW=' + ed_password.Text + '&');
end;
```

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

```
procedure Twebio_ascii_client.ed_intervalChange(Sender:TObject);
begin
  timer_polling.Interval := strtoint(ed_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 PoE. 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

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

Data Privacy