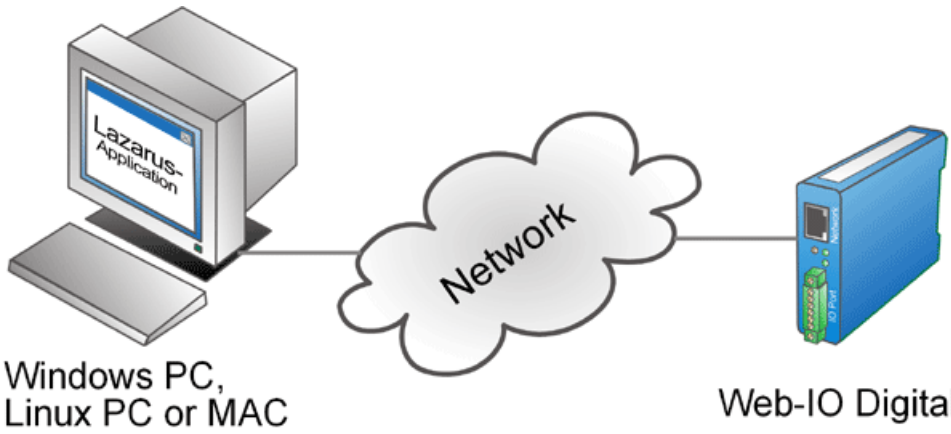
Application for the Web-IO Digital:

Controlling and monitoring the Web-IO Digital with Lazarus/Free Pascal

Lazarus is an open source development environment which emulates Delphi and which is based on Free Pascal. The interesting feature of Lazarus is that this development environment is available for Windows, Linux and some MAC systems and makes the projects usable cross-platform. Lazarus can be downloaded for free at https://www.lazarus.freepascal.org/.

As an easy to learn high level language Lazarus offers nearly everything you need to program Web-IO® applications. Processing TCP/IP communication does however require that the additional LNET control element be installed as a component package.



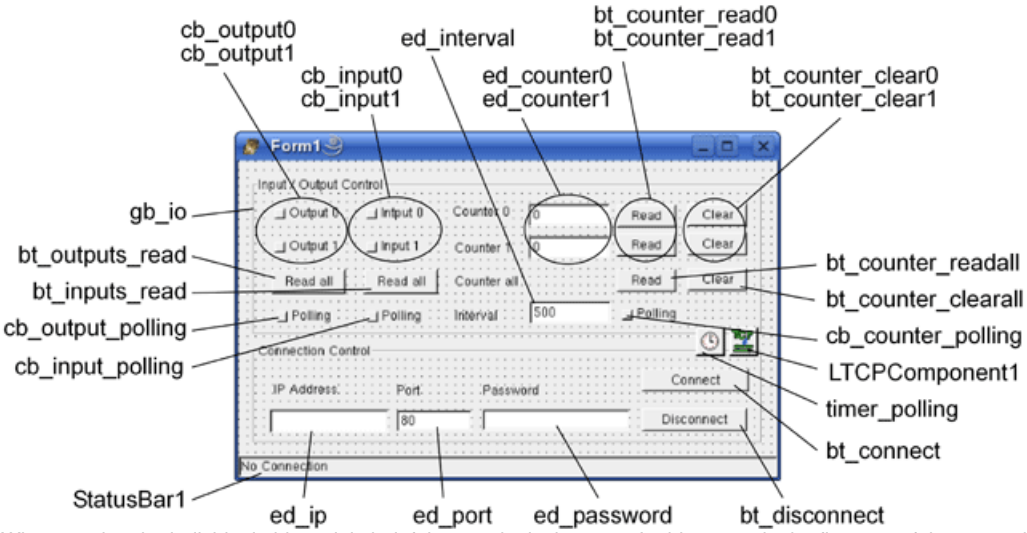
Using the following program example you can represent your Web-IO Digital with its inputs and outputs in a Windows application. You can also switch the Web-IO outputs.

Preparations

You have already provided your Web-IO Digital

- with power,
- configured the inputs and outputs,
- connected it to your network,
- assigned it an IP address - which with WuTility is no problem.

1. Assembling the various operating elements and display objects in Lazarus format



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.

2. 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 TForm1.bt_connectClick(Sender: TObject);
begin
 if ed_ip.Text <> '' then
 begin
  bt_connect.Enabled := false;
  LTCPComponent1.Connect(ed_ip.Text, strtoint(ed_port.Text));
 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 TForm1.LTCPComponent1Connect(aSocket:TLSocket);
begin
 StatusBar1.SimpleText :='Connected to ' + ed_ip.Text;
 bt_connect.Enabled := False;
 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 TForm1.bt_disconnectClick(Sender:TObject);
begin
 StatusBar1.SimpleText :='No Connection';
 bt_connect.Enabled := True;
 bt_disconnect.Enabled := False;
 gb_io.Enabled := False;
 LTCPComponent1.Disconnect;
end;
```

Here again the ClientSocket control element invokes a corresponding procedure

```
procedure TForm1.LTCPComponent1Disconnect(aSocket:TLSocket);
begin
 StatusBar1.SimpleText :='No Connection';
 bt_connect.Enabled := True;
 bt_disconnect.Enabled := False;
 gb_io.Enabled := False;
 LTCPComponent1.Disconnect;
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 TForm1.LTCPComponent1Error(const msg: string; aSocket: TLSocket);
begin
 StatusBar1.SimpleText :='Error: '+ msg;
 bt_connect.Enabled := True;
 bt_disconnect.Enabled := False;
 gb_io.Enabled := False;
 LTCPComponent1.Disconnect;
end;
```

3. 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 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 TForm1.cb_output0MouseUp(Sender:TOBject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 if cb_output0.Checked then
 LTCPComponent1.SendMessage('GET/outputaccess0?PW=' + ed_password.Text+ '&State=ON&',nil)
 else
 LTCPComponent1.SendMessage('GET/outputaccess0?PW=' + ed_password.Text + '&State=OFF&',nil)
end;


procedure TForm1.cb_output1MouseUp(Sender:TOBject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 if cb_output1.Checked then
 LTCPComponent1.SendMessage('GET/outputaccess1?PW=' + ed_password.Text + '&State=ON&',nil)
 else
 LTCPComponent1.SendMessage('GET/outputaccess1?PW=' + ed_password.Text + '&State=OFF&',nil)
end;
```

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

```
procedure TForm1.bt_outputs_readClick(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/output?PW=' + ed_password.Text + '&',nil);
end;


procedure TForm1.bt_inputs_readClick(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/input?PW=' + ed_password.Text + '&',nil);
end;
```

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

```
procedure TForm1.bt_counter_read0Click(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/counter0?PW=' + ed_password.Text + '&',nil);
end;


procedure TForm1.bt_counter_read1Click(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/counter1?PW=' + ed_password.Text + '&',nil);
end;


procedure TForm1.bt_counter_clear0Click(Sender: TObject);
begin
 LTCPComponent1.SendMessage('GET/counterclear0?PW=' + ed_password.Text + '&',nil);
end;


 procedure TForm1.bt_counter_clear1Click(Sender:TObject);
 begin
  LTCPComponent1.SendMessage('GET/counterclear1?PW=' + ed_password.Text + '&',nil);
 end;
```

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

```
procedure TForm1.bt_counter_readallClick(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/counter?PW=' + ed_password.Text + '&',nil);
end;
```

```
procedure TForm1.bt_counter_clearallClick(Sender:TObject);
begin
 LTCPComponent1.SendMessage('GET/counterclear?PW=' + ed_password.Text + '&',nil);
end;
```

4. 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.
- If data are received from the ClientSocket control element, the latter invokes the corresponding procedure

```
procedure TForm1.LTCPComponent1Receive(aSocket:TLSocket);
var ReceiveString : String;
SemiPosition : integer;
OutputValue : word;
InputValue : word;
begin
 aSocket.GetMessage(ReceiveString);
 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;
```

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.

5. Polling

The desired interval can be entered in the corresponding text field.

```
procedure TForm1.timer_pollingTimer(Sender:TObject);
begin
 if LTCPComponent1.Connected and cb_output_polling.Checked then
  LTCPComponent1.SendMessage('GET/output?PW=' + ed_password.Text + '&',nil);
 if LTCPComponent1.Connected and cb_input_polling.Checked then
  LTCPComponent1.SendMessage('GET/input?PW=' + ed_password.Text + '&',nil);
 if LTCPComponent1.Connected and cb_counter_polling.Checked then
  LTCPComponent1.SendMessage('GET/counter?PW=' + ed_password.Text + '&',nil);
end;
```

When changes are made the timer interval is automatically adjusted.

```
procedure TForm1.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. 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

You don't have a Web-IO yet but would like to try the example out sometime?
No problem: We will be glad to send you the Web-IO Digital 2xInput, 2xOutput at no charge for 30 days. Simply fill out a sample ordering form, and we will ship the Web-IO for testing on an open invoice. If you return the unit within 30 days, we will simply mark the invoice as paid.

To sample orders

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