A Developers Guide To The Wolfprot Protocol

WolfVision

February 17, 2014

Contents

Chapter 1

Copyright

November 2012

Chapter 2

Introduction

Wolfprot is the WolfVision protocol to control visualizers. This handbook explains how to use it by means of examples. Beginning with simple single part commands like switching on the visualizer, it moves forward to advanced multi part commands like receiving pictures from the visualizer. The examples are provided in C#. If visualizers support it, the interfaces are RS232, USB and Ethernet. The commands are described in the document command_list.pdf, which can be downloaded from the WolfVision homepage. The command list is split into GET and SET commands. GET commands retrieve information from the visualizer, SET commands change settings in the visualizer. Wolfprot commands are organized in request - reply pairs. They start with a header, followed by the command, the length and data. The header contains a set of bits to make the protocol extensible in the future. The packet structure is explained in the document command_list.pdf.
The document refers sometimes to EYE12/FB4 and PF1 platform. This is because some commands are used differently depending on these platforms. See the table below showing which visualizers belong to which platform.
To start, read the command_list.pdf document introduction to understand the packet structure. Connect the visualizer with one out of the 3 connection possibilities or all. For compiling the example application Microsoft Visual Studio Express can be used, which can be downloaded for free. The example program is divided into different panels. In the main form the connection to the visualizer is handled and links to sub forms are provided. The sub forms each handle a specific theme. We start right away with explaining the source code.

Different visualizer platforms supporting Wolfprot:

EYE12/FB4	PF1
EYE-12	VZ-8plus-4
VZ-9plus-3	VZ-8light-4
VZ-C12-3	VZ-C3D
VZ-C32-3	
VZ-P18	
VZ-P38	
VZ-3	

Table 2.1: Platforms

Chapter 3

Interfaces

The interface IComIf defines the interface types and the needed methods for the interfaces.

Listing describing the interface for the 3 interfaces RS232, USB and Ethernet:

```
1
2  using System;
3
4  namespace VZInterfaceTestHandbook
5  {
6    public enum InterfaceType
7    {
8      RS232 = 0,
9      ETH = 1,
10     USB = 2
11   };
12
13   public interface IComIf
14   {
15     InterfaceType GetInterfaceType();
16     Boolean Open();
17     Boolean Close();
18     Boolean Write(Byte[] buffer, int count);
19     int Read(out Byte[] buffer, int count);
20     Boolean ReadByte(ref Byte byRecv);
21     Boolean ReadTimeout(int timeout);
22     Boolean isConnected();
23     int BytesToRead();
24     void Flush();
25     Boolean SetBaudrate(int baudRate);
26   }
27 }
```

3.1 RS232

The RS232 uses 3 wires Rxd, Txd and Gnd. The Baudrate can be selected between 9600 and 115200. To use the RS232 connection, select the RS232 radio button and select the port the visualizer is connected to. The baudrate is set internally to 115200 baud. It is necessary to set the visualizer to the same baudrate. This is done in the Extra Menu.

Listing explaining how to set up the serial port:

```
1
2  using System;
3  using System.Collections.Generic;
4
5  namespace VZInterfaceTestHandbook
6  {
7    public partial class RS232Com : IComIf
8    {
9      private System.IO.Ports.SerialPort comPort = new System.IO.Ports.SerialPort();
10     int iTimeout = 1000;
11
12     /**
13      * Open the connection
14      */
15     public Boolean Open()
16     {
17       try
18       {
19         if (isConnected())
20           comPort.Close();
21         comPort.Open();
22       }
23       catch (Exception)
24       {
25       }
26       return !comPort.IsOpen;
27     }
28
29     /**
30      * Close the connection
31      */
32     public Boolean Close()
33     {
34       try
35       {
36         if (isConnected())
37           comPort.Close();
38       }
39       catch (Exception)
40       {
41       }
42       return false;
43     }
44
45     public InterfaceType GetInterfaceType()
46     {
47       return InterfaceType.RS232;
48     }
49
50     /**
51      * Configures the comport name and the baudrate
52      * @param strComName: comport name
53      * @param baudRate: sets the baudrate
```

```
54        */
55      public void Configure(string strComName, int baudRate)
56      {
57        try
58        {
59          comPort.PortName = strComName;
60          comPort.BaudRate = baudRate;
61        }
62        catch (Exception)
63        {
64        }
65      }
66
67      /**
68       * Sets the baudrate
69       * @param baudRate: sets the baudrate to
70       *                  the value of comPort.BaudRate
71       */
72      public Boolean SetBaudrate(int baudRate)
73      {
74        try
75        {
76          comPort.BaudRate = baudRate;
77        }
78        catch (Exception)
79        {
80        }
81        return false;
82      }
83
84      /**
85       * Check connection state
86       * @return open = true
87       */
88      public Boolean isConnected()
89      {
90        return comPort.IsOpen;
91      }
92
93      /**
94       * Write bytes
95       * @param buffer: bytes to write
96       * @param count: number of bytes
97       * @return: success = false
98       */
99      public Boolean Write(Byte[] buffer, int count)
100     {
101       if (!isConnected())
102         return true;
103       try
104       {
105         comPort.Write(buffer, 0, count);
106       }
107       catch (Exception)
108       {
109         return true;
110       }
111       return false;
112     }
113
114     /**
```

```
115       * Set the timeout
116       * @param int timeout: time in ms
117       */
118      public Boolean ReadTimeout(int timeout)
119      {
120        iTimeout = timeout;
121        return false;
122      }
123
124      /**
125       * @return available bytes for read
126       */
127      public int BytesToRead()
128      {
129        int iToRead = 0;
130
131        if (!isConnected())
132          return 0;
133        try
134        {
135          iToRead = comPort.BytesToRead;
136        }
137        catch (Exception)
138        {
139        }
140        return iToRead;
141      }
142
143      /**
144       * Clear queue
145       */
146      public void Flush()
147      {
148        if (!isConnected())
149          return;
150        try
151        {
152          comPort.DiscardInBuffer();
153        }
154        catch (Exception)
155        {
156        }
157      }
158
159      /**
160       * Read bytes currently available
161       * @param buffer: to be filled
162       * @param count: number of bytes to be read
163       * returns read bytes
164       */
165      public int Read(out Byte[] buffer, int count)
166      {
167        List<Byte> lRecvBuffer = new List<Byte>();
168
169        if (!isConnected())
170        {
171          buffer = null;
172          return -1;
173        }
174
175        try
```

```
176         {
177           comPort.ReadTimeout = iTimeout;
178           while (count-- > 0)
179             lRecvBuffer.Add((Byte)comPort.ReadByte());
180         }
181         catch
182         {
183         }
184 
185         buffer = lRecvBuffer.ToArray();
186         return lRecvBuffer.Count;
187       }
188 
189       /**
190        * Read one byte
191        * @param byteRecv: to be filled
192        */
193       public Boolean ReadByte(ref Byte byteRecv)
194       {
195         if (!isConnected())
196           return true;
197 
198         try
199         {
200           comPort.ReadTimeout = iTimeout;
201           byteRecv = (Byte)comPort.ReadByte();
202         }
203         catch
204         {
205           return true;
206         }
207 
208         return false;
209       }
210 
211     }
212 }
```

3.2 Ethernet (ETH)

The Ethernet connection uses TCP/IP. The communication uses port 50915. The visualizer supports DHCP and static IP. If only visualizer and PC are connected, it is necessary to set the computer and the visualizer to static IP. In a network with DHCP server simply set computer and visualizer to use DHCP. To use the Ethernet connection, select the ETH radio button and enter the IP address the visualizer is set to. The IP address of the visualizer can be found in the OSD menu of the visualizer.

Listing explaining how to set up Ethernet:

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Windows.Forms;
10 using System.Net;
11 using System.IO;
12 using System.Net.Sockets;
13
14 namespace VZInterfaceTestHandbook
15 {
16   public partial class EthCom : IComIf
17   {
18     TcpClient tcpClient = new TcpClient();
19     Stream tcpStream;
20     string tcpIp = "127.0.0.1";
21     int tcpPort = 50915;
22
23     /**
24      * Open the connection
25      */
26     public Boolean Open()
27     {
28       try
29       {
30         if (isConnected())
31           Close();
32         tcpClient.Connect(tcpIp, tcpPort);
33         tcpStream = tcpClient.GetStream();
34       }
35       catch (Exception e)
36       {
37         Console.WriteLine(e.ToString());
38       }
39       return !isConnected();
40     }
41
42     /**
43      * Close the connection
44      */
45     public Boolean Close()
46     {
47       try
48       {
49         tcpStream.Close();
50         tcpClient.Close();
51         // Close also disposes the tcp client ... so this is a nice hack
```

```
52          tcpClient = new TcpClient();
53        }
54        catch (Exception)
55        {
56        }
57        return false;
58      }
59
60      public void Configure(string strIp, int iPort)
61      {
62        tcpIp = strIp;
63        tcpPort = iPort;
64      }
65
66      public InterfaceType GetInterfaceType()
67      {
68        return InterfaceType.ETH;
69      }
70
71      /**
72       * Required by interface
73       */
74      public Boolean SetBaudrate(int baudRate)
75      {
76        return false;
77      }
78
79      /**
80       * Check connection
81       * @return connected = true
82       */
83      public Boolean isConnected()
84      {
85        return tcpClient.Connected;
86      }
87
88      /**
89       * Write bytes
90       * @param buffer: bytes to write
91       * @param count: number of bytes
92       * @return: success = false
93       */
94      public Boolean Write(Byte[] buffer, int count)
95      {
96        if (!isConnected())
97          return true;
98        try
99        {
100         tcpStream.Write(buffer, 0, count);
101       }
102       catch (Exception)
103       {
104         return true;
105       }
106
107       return false;
108     }
109
110     /**
111      * Set the timeout
112      * @param timeout: in ms
```

```
113       */
114      public Boolean ReadTimeout(int timeout)
115      {
116        if (timeout > 0)
117          tcpStream.ReadTimeout = timeout;
118        return false;
119      }
120
121      /**
122       * @return Number of bytes available for read
123       */
124      public int BytesToRead()
125      {
126        int iToRead = 0;
127
128        if (!isConnected())
129          return 0;
130        try
131        {
132          iToRead = tcpClient.Available;
133        }
134        catch (Exception)
135        {
136        }
137
138        return iToRead;
139      }
140
141      /**
142       * Clear queue
143       */
144      public void Flush()
145      {
146        if (!isConnected())
147          return;
148        try
149        {
150          tcpStream.Flush();
151        }
152        catch (Exception)
153        {
154        }
155      }
156
157      /**
158       * Read bytes currently available
159       * @param buffer: to be filled
160       * @param count: number of bytes to be read
161       * @return read bytes
162       */
163      public int Read(out Byte[] buffer, int count)
164      {
165        Byte[] buf = new Byte[count];
166
167        int iRead;
168        if (!isConnected())
169        {
170          buffer = null;
171          return -1;
172        }
173
```

```
174         try
175         {
176           iRead = tcpStream.Read(buf, 0, count);
177         }
178         catch (Exception e)
179         {
180           System.Console.WriteLine(e.ToString()); //Exception-outline
181           buffer = null;
182           return -1;
183         }
184 
185         buffer = buf;
186         return iRead;
187       }
188 
189       /**
190        * Read one byte
191        * @param byteRecv: to be filled
192        */
193 
194       public Boolean ReadByte(ref Byte byteRecv)
195       {
196         if (!isConnected())
197           return true;
198         try
199         {
200           byteRecv = (Byte)tcpStream.ReadByte();
201         }
202         catch (Exception)
203         {
204           return true;
205         }
206 
207         return false;
208       }
209     }
210 }
```

3.3 USB

The USB connection uses the Microsoft USB dll. To use the USB connection, select the USB radio button and press scan for available visualizers. From the combobox select the visualizer that you want to connect to.

Listing explaining how to set up USB:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.InteropServices;
4  using System.Collections;
5
6  namespace VZInterfaceTestHandbook
7  {
8    public partial class UsbCom : IComIf
9    {
10     private WinUsbNet.UsbDevice usbDevice;
11
12     private void DebugMsg(string strMSg)
13     {
14       //System.Console.WriteLine("USBCom: " + strMSg);
15     }
16
17     /**
18      * Open the connection
19      */
20     public Boolean Open()
21     {
22       try
23       {
24         usbDevice = USBManager.currUSBDevice;
25
26         if (!usbDevice.IsOpen)
27           usbDevice.Open();
28       }
29       catch (Exception)
30       {
31       }
32       return !usbDevice.IsOpen;
33     }
34
35     /**
36      * Close the connection
37      */
38     public Boolean Close()
39     {
40       try
41       {
42         if (isConnected())
43         {
44           USBManager.currUSBDevice.Close();
45         }
46       }
47       catch (Exception)
48       {
49       }
50       return false;
51     }
52
53     public void Configure(WinUsbNet.UsbDevice usbDev)
54     {
```

```
55        this.usbDevice = usbDev;
56      }
57
58      public InterfaceType GetInterfaceType()
59      {
60        return InterfaceType.USB;
61      }
62
63      public Boolean SetBaudrate(int baudRate)
64      {
65        return false;
66      }
67
68      public Boolean isConnected()
69      {
70        return USBManager.currUSBDevice.IsOpen;
71      }
72
73      /**
74       * Write bytes
75       * @param buffer: bytes to write
76       * @param count: number of bytes
77       * @return: success = false
78       */
79      public Boolean Write(Byte[] buffer, int count)
80      {
81        Boolean bError = false;
82
83        try
84        {
85          USBManager.SendPipe.Write(buffer, 0, count);
86          //DebugMsg("Wrote " + count + " bytes (" +
              Toolbox.ByteArryToString(buffer, count) + ")");
87        }
88        catch (Exception)
89        {
90          return true;
91        }
92
93        return bError;
94      }
95
96      /**
97       * Set the timeout
98       * @param int timeout: time in ms
99       */
100     public Boolean ReadTimeout(int timeout)
101     {
102       USBManager.ReceivePipe.ReadTimeout = timeout;
103       return false;
104     }
105
106     /**
107      * Available bytes
108      * return 0 because not implemented
109      */
110     public int BytesToRead()
111     {
112       return 0;
113     }
114
```

```
115     public void Flush()
116     {
117     }
118 
119     /**
120      * Read bytes currently available
121      */
122     public int Read(ref Byte[] buffer)
123     {
124       if (!isConnected())
125         return -1;
126       USBManager.ReceivePipe.Read(buffer, 0, buffer.Length);
127       return buffer.Length;
128     }
129 
130     /**
131      * Read bytes currently available
132      * @param buffer: to be filled
133      * @param count: number of bytes to be read
134      * return read bytes
135      */
136     public int Read(out byte[] buffer, int count)
137     {
138       Byte[] buf = new Byte[count];
139       int iRead;
140 
141       if (!isConnected())
142       {
143         buffer = null;
144         return -1;
145       }
146       USBManager.ReceivePipe.ReadTimeout = 1000;
147       iRead = USBManager.ReceivePipe.Read(buf, 0, count);
148 
149       buffer = buf;
150       return iRead;
151     }
152 
153     /**
154      * Read a single byte
155      */
156     public Boolean ReadByte(ref Byte byteRecv)
157     {
158       Boolean bError = false;
159       if (!isConnected())
160         return true;
161 
162       USBManager.ReceivePipe.ReadTimeout = 1000;
163       byteRecv = (Byte)USBManager.ReceivePipe.ReadByte();
164 
165       return bError;
166     }
167 
168     #region IComIf Members
169 
170     #endregion
171   }
172 }
```

3.3.1 USB Scan Devices

Listing showing how to search for connected visualizers:

```
1     private void buttonScanUSBDevices_Click(object sender, EventArgs e)
2     {
3       uint uiDeviceCount;
4       int i = 0;
5
6
7       cbUSbDevices.Items.Clear();
8
9       uiDeviceCount = USBManager.USBFindAllDevs();
10      if (uiDeviceCount == 0)
11      {
12        MessageBox.Show("No Visualizer found on USB", "No Visualizer found",
              MessageBoxButtons.OK, MessageBoxIcon.Stop);
13        return;
14      }
15
16      foreach (WinUsbNet.UsbDevice USBDevice in USBManager.USBMan.UsbDevices)
17      {
18        Byte[] baSerial = new Byte[100];
19
20        if (USBDevice.IsOpen) USBDevice.Close();
21        USBManager.currUSBDevice = USBDevice;
22        try
23        {
24          if (!USBDevice.IsOpen)
25            USBDevice.Open();
26        }
27        catch (Exception)
28        {
29          MessageBox.Show("USB device " + Environment.NewLine +
                USBDevice.DeviceName + Environment.NewLine + "already opened by
                another application (probably WolfVision Connectivity Software 2)" +
                Environment.NewLine + "Please close this application to grant access
                to the USB device", "USB Error", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
30          continue;
31        }
32
33        i += 1;
34        USBDevice.Tag = USBManager.GetTypeSerialVersion(USBDevice);
35        string[] DeviceInfo = USBDevice.Tag.ToString().Split(new char[] { ';' });
36
37        cbUSbDevices.Items.Add(i + " - " + DeviceInfo[0] + " (" + DeviceInfo[1] +
              ") " + DeviceInfo[2]);
38
39      }
40
41      if (cbUSbDevices.Items.Count > 0)
42        cbUSbDevices.SelectedIndex = 0;
43
44    }
```

Chapter 4

Wolfprot

4.1 Package

To execute a command a method that can send and receive Wolfprot packages is needed. For sending commands instances of send and reply buffers with the appropriate length are needed. The send buffer holds the bytes that make up a command. The reply buffer holds the answer from the visualizer.

4.1.1 execCmd

The execCmd method sends out a command and parses the answer byte for byte to verify the validity of the reply. The method is useful for Wolfprot commands that have a small payload of up to 256 bytes.

Listing of execCmd:

```
1     public Boolean execCmd(Byte[] baCommand, ref Byte[] baReceived)
2     {
3       Boolean bError = false;
4
5       ResetParser();
6       bError |= comIf.ReadTimeout(0);
7
8       // Remove all bytes from queue
9       comIf.Flush();
10
11      bError |= comIf.Write(baCommand, baCommand.Length);
12      bError |= comIf.ReadTimeout(3000);  //3 seconds
13
14      Byte byRecv = 0;
15      do
16      {
17        bError |= comIf.ReadByte(ref byRecv);
18      } while (!ParseData(byRecv, comIf) && !bError);
19
20      baReceived = GetData();
21
22      return bError;
23    }
```

4.1.2 execCmdFast

To receive large chunks of bytes like pictures the method execCmdFast should be used. This command tries to receive a data package as a whole. This command is useful for Wolfprot commands with a payload of more than 256 bytes.

Listing of execCmdFast:

```
1     public Boolean execCmdFast(Byte[] baCommand, ref Byte[] baReceived, int
          blocklength)
```

```
2     {
3       Boolean bError = false;
4       Byte[] bufRec = new Byte[blocklength];
5
6       int cnt = blocklength;
7       bError |= comIf.ReadTimeout(0);
8
9       // Remove all bytes from queue
10      comIf.Flush();
11
12      bError |= comIf.Write(baCommand, baCommand.Length);
13      bError |= comIf.ReadTimeout(10000);
14
15      int check = 0;  // Length of chunck read
16
17      int ptr = 0;
18      do // Read loop
19      {
20        check = comIf.Read(out bufRec, cnt);
21        cnt = cnt - check;
22
23        Array.Copy(bufRec, 0, baReceived, ptr, check);
24        //Check the header
25        if((ptr == 0) && (check <= 3))     // Wolfprot error message (1 byte
              header + 1 byte cmd)
26        {
27          if (check == 0)                  // No bytes received, device switched
                off
28          {
29            baReceived = null;
30            bError |= true;
31            break;
32          }
33          if ((baReceived[0] & 0x80) == 0x80)
34            break;
35        }
36
37        ptr += check;
38        if (check != -1)
39          bError |= false;
40
41      } while ((cnt>0) && !bError);
42
43      return bError;
44    }
```

4.2 Error Handler

It is recommended to check for errors. The following code decodes the errors and shows them in a message box.

Listing of the error handler:

```
1     public void ErrorSummary(Byte ByteError)
2     {
3       switch (ByteError)
4       {
5         case 0x01:
6           MessageBox.Show("Timeout");
7           break;
8         case 0x02:
9           MessageBox.Show("Unknown command");
10          break;
11        case 0x03:
12          MessageBox.Show("Unknown parameter");
13          break;
14        case 0x04:
15          MessageBox.Show("Invalid length");
16          break;
17        case 0x05:
18          MessageBox.Show("Fifo full");
19          break;
20        case 0x06:
21          MessageBox.Show("Firmware update");
22          break;
23        case 0x07:
24          MessageBox.Show("Access denied");
25          break;
26        case 0x08:
27          MessageBox.Show("Auth required");
28          break;
29        case 0x09:
30          MessageBox.Show("Busy");
31          break;
32        default:
33          MessageBox.Show("Unknown error");
34          break;
35      }
36    }
```

Chapter 5

Mainform

The Mainform contains elements to set up the communication to the visualizer as well as buttons to access the different example tasks.

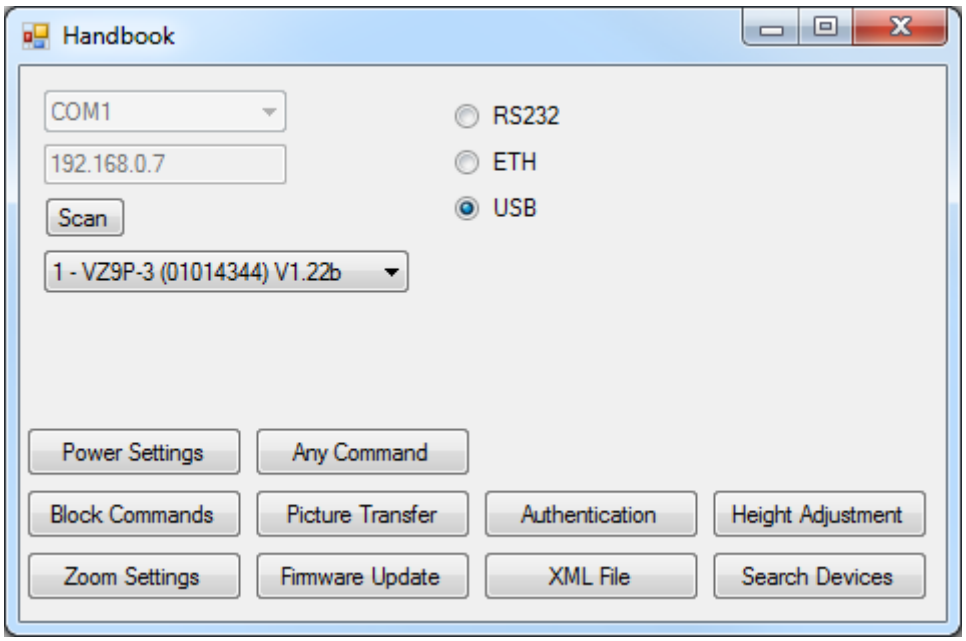

Figure 5.1: Mainform

Chapter 6

PowerSettings

As an example of a simple GET/SET Wolfprot command GET/SET Power is chosen. Most of the Wolfprot commands work like this command. The commands enable you to switch the visualizer on or off as well as reading the power state.

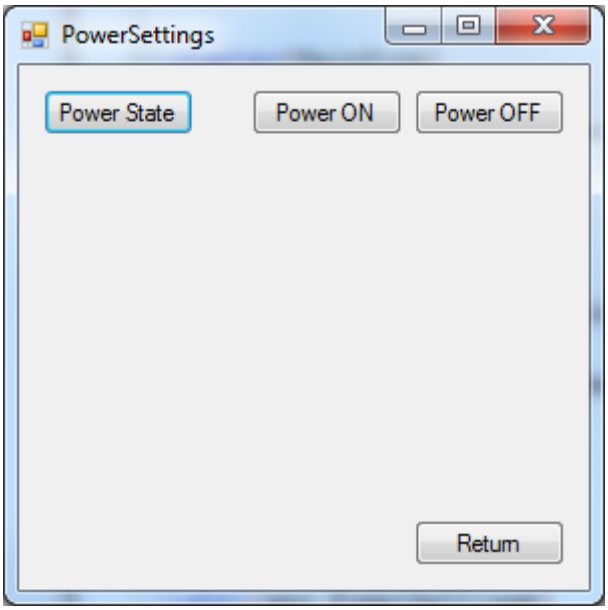

Figure 6.1: Form power settings

This form is opened from the mainform by clicking the button "PowerSettings". The form includes a button for every single command (power on; power off; get power state;) as well as a button to return to the mainform.
The constructor takes the interface as parameter and passes it to the class.

```
1       IComIf comIf;
2       Boolean isConnected = false;
3
4       /**
5        * constructor
6        * @param com: uses Interface IComIf
7        */
8       public FormPowerSettings(IComIf com)
9       {
10        comIf = com;
11        InitializeComponent();
12      }
```

6.1 Power On

The command gets executed by pressing Power On.

Listing showing the buttonPowerOn_Click method:

```
1     private void buttonPowerOn_Click(object sender, EventArgs e)
2     {
3       /*Instantiate class*/
4       PowerSettings pS = new PowerSettings(comIf);
5       /*Open interface*/
6       if (connOpen())
7         return;
8       /*Send command*/
9       pS.PowerON();
10      /*Close interface*/
11      connClose();
12    }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.
2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

To create a new command instantiate a new Wolfprot2 class. Define two bytearrays for sending and receiving data.

1. bufSnd: For sending a command to the visualizer
2. bufRecv: For receiving the information from the visualizer.

The command for power is found in the document "command_list.pdf" The packet for the Power On command has following byte sequence: 01 30 01 01

1. 01: Bit 0 is the direction bit. 1 equals SET
2. 30: The command for power on/off setting
3. 01: Length of data
4. 01: Change to power on (data)

Listing to illustrate how to send the generic SET command:

```
1     public void PowerON()
2     {
3       Wolfprot2 wp2 = new Wolfprot2(comIf);
4       Boolean iReturn;
5       /*Declaration*/
6       Byte[] bufSnd = new Byte[4];
7       Byte[] bufRecv = new Byte[3];
8
9       /*Assignment*/
10      bufSnd[0] = 0x01;
11      bufSnd[1] = 0x30;
12      bufSnd[2] = 0x01;
13      bufSnd[3] = 0x01;
14
15      /*Send command*/
16      iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17      if (iReturn == true)
18      {
19        MessageBox.Show("No communication!", "Error",
20                        MessageBoxButtons.OK,
21                        MessageBoxIcon.Error);
22      }
23      else
```

```
24        {
25          if ((bufRecv[0] & 0x80) == 0x80)
26            wp2.ErrorSummary(bufRecv[2]);
27          else
28            MessageBox.Show("Command successful");
29        }
30      }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x01: Bit 0 is the direction bit. 1 equals SET
2. 0x30: The command for power on/off setting
3. 0x00: Length of data

If the command was not successful the reply packet looks like this.
Note: The error reply packet is decribed only once but applies to every Wolfprot command.

1. 0x80: Bit 7 is the error bit. 1 means error
2. 0x30: The command for power on/off setting
3. 0x08: Error code (in this case AUTH_REQUIRED)

6.2 Power Off

For Power off replace byte[3] in Power on with 0x00.

6.3 Get Power State

Instead of setting a state of the visualizer also its current value can be retrieved. This is done by GET commands.

Listing of button "GetPowerState":

```
1       private void buttonGetPowerState_Click(object sender, EventArgs e)
2       {
3         /*Instantiate class*/
4         PowerSettings pS = new PowerSettings(comIf);
5         /*Open interface*/
6         if (connOpen())
7           return;
8         /*Send command*/
9         pS.GetPowerState();
10        /*Close interface*/
11        connClose();
12      }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.
2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

Get Power State command has following byte sequence: 00 30 00. (it's a get command)

1. 00: Bit 0 is the direction bit. 0 equals GET
2. 30: The command for power on/off setting
3. 00: Length of data

Listing illustrating how to send a generic GET command:

```
1     public void GetPowerState()
2     {
3       Wolfprot2 wp2 = new Wolfprot2(comIf);
4       Boolean iReturn;
5       /*Declaration*/
6       Byte[] bufSnd = new Byte[3];
7       Byte[] bufRecv = new Byte[4];
8
9       /*Assignment*/
10      bufSnd[0] = 0x00;
11      bufSnd[1] = 0x30;
12      bufSnd[2] = 0x00;
13
14      /*Send command*/
15      iReturn = wp2.execCmd(bufSnd, ref bufRecv);
16      if (iReturn == true)
17      {
18        MessageBox.Show("No communication!", "Error",
19                        MessageBoxButtons.OK,
20                        MessageBoxIcon.Error);
21      }
22      else if ((bufRecv[0] & 0x80) == 0x80)
23        wp2.ErrorSummary(bufRecv[2]);
24      else
25      {
26        ShowState(bufRecv[3]);
27      }
28    }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x00: Bit 0 is the direction bit. 0 equals GET
2. 0x30: The command for power on/off setting
3. 0x00: Length of data

Chapter 7

Block Commands

Block commands are used to get a lot of information from the visualizer at once. They exist only as GET commands. These special GET commands work like ordinary GET commands with the difference that they all use the same command number 0x10 followed by a block number.

Listing showing how to get the Info & Flags block:

```
1     public Byte[] FlagsAndUnits2()
2     {
3       Wolfprot2 wp2 = new Wolfprot2(comIf);
4       Boolean iReturn;
5       /*Declaration*/
6       Byte[] bufSnd = new Byte[4];
7       Byte[] bufRecv = new Byte[128 + 4]; //header,cmd,len,blk,128 data
8
9       /*Assignment*/
10      bufSnd[0] = 0x00;
11      bufSnd[1] = 0x10;
12      bufSnd[2] = 0x01;
13      bufSnd[3] = 0x0B; //block number
14
15      /*Send command*/
16      iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17      if (iReturn == true)
18      {
19        MessageBox.Show("No communication!", "Error",
20                        MessageBoxButtons.OK,
21                        MessageBoxIcon.Error);
22        return null;
23      }
24      else
25      {
26        if ((bufRecv[0] & 0x80) == 0x80)
27        {
28          wp2.ErrorSummary(bufRecv[2]);
29          return null;
30        }
31        else
32          return bufRecv;
33      }
34    }
```

Chapter 8

Zoom Settings

With the zoom settings it is possible to change the zoom factor. There are three ways to change the zoom factor:

1. zoom wide/far: These commands start the zooming towards the end of the zoom range. This commands need to be followed by a stop command.
2. zoomspeed: The zoom factor is changed with a chosen speed
3. zoomposition: The zoom factor is set to a fixed position

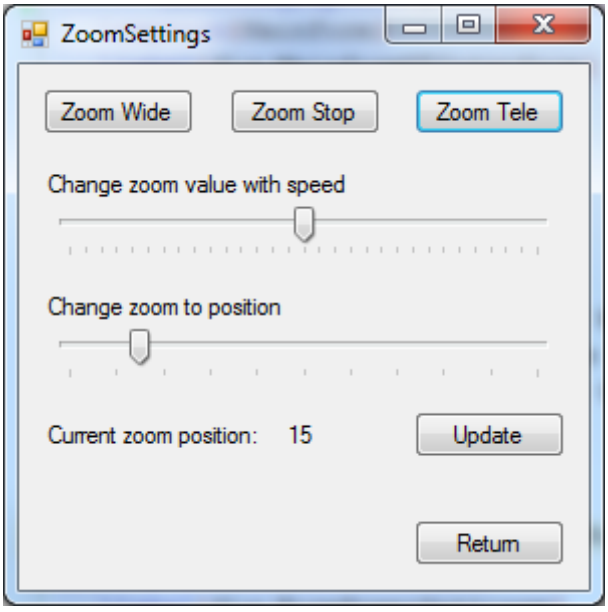

Figure 8.1: Form zoom settings

This form is opened by clicking the button "Zoom Settings" in the mainform. The new form includes 3 buttons (zoom wide/far/stop), two trackbars(zoomspeed, zoomposition) and a button to return to the mainform.
The constructor takes the interface as parameter and passes it to the class.

```
1       const int STEP = (0x2000 / 100) + 1;  // (Digital Zoom Range + Analog Zoom
            Range)/Scale of Slider
2       const int STEP2 = (0x1000 / 100) + 1; // (Combined Digital and Analog Zoom
            Range)/Scale of Slider
3       IComIf comIf;
4
5       /**
6        * Zoom commands
7        * @param com: uses Interface IComIf
8        */
9       public ZoomSettings(IComIf com)
10      {
```

```
11        comIf = com;
12      }
```

8.1 Start Zoom Wide

With the buttons "Zoom Wide" and "Zoom Tele" zooming runs towards the end of the zoom range. With the button "Zoom Stop" the zoom stops at the current position. To move from analog to digital zoom range it is necessary to send out a "Zoom Stop" command at the end of the analog range. With "Zoom Tele" it is then possible to move into digital zoom range.

Listing showing the "StartZoomWide" method:

```
1       private void buttonStartZoomWide_Click(object sender, EventArgs e)
2       {
3         /*Instantiate class*/
4         ZoomSettings zS = new ZoomSettings(comIf);
5         /*Open interface*/
6         if (connOpen())
7           return;
8         /*Send command*/
9         zS.StartZoomWide();
10        /*Close interface*/
11        connClose();
12      }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.
2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

To create a new command instantiate a new Wolfprot2 class. Define two bytearrays for sending and receiving data.

1. bufSnd: For sending a command to the visualizer
2. bufRecv: For receiving the information from the visualizer.

The command for "Start Zoom Wide" can be found in the document "command_list.pdf". The packet for the Zoom Wide command has following byte sequence: 01 20 01 11

1. 01: Bit 0 is the direction bit. 1 equals SET
2. 20: The command for zoom settings
3. 01: Length of data
4. 11: Start zoom in direction wide

Listing illustrating how to send a SET ZOOM WIDE command:

```
1       public void StartZoomWide()
2       {
3         Wolfprot2 wp2 = new Wolfprot2(comIf);
4         /*Declaration*/
5         Byte[] bufSnd = new Byte[4];
6         Byte[] bufRecv = new Byte[3];
7         Boolean iReturn;
8
9         /*Assignment*/
10        bufSnd[0] = 0x01;
11        bufSnd[1] = 0x20;
12        bufSnd[2] = 0x01;
13        bufSnd[3] = 0x11;
14
```

```
15        /*Send command*/
16        iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17        if (iReturn == true)
18        {
19          MessageBox.Show("No communication!", "Error",
20                          MessageBoxButtons.OK,
21                          MessageBoxIcon.Error);
22        }
23        else
24        {
25          if ((bufRecv[0] & 0x80) == 0x80)
26            wp2.ErrorSummary(bufRecv[2]);
27        }
28      }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x00: Bit 0 is the direction bit. 0 equals GET
2. 0x20: The command for zoom settings
3. 0x00: Length of data

8.2 Start Zoom Tele

For "Start Zoom Tele" replace byte[3] in "Start Zoom Wide" with 0x12.

8.3 Zoom Slider Speed

In "change zoom value with speed" move the thumb to left or right and hold it. The visualizer will zoom with the selected speed until the thumb gets released. "change zoom to position" will show the final value.

Listing showing the implementation of the Zoom Slider Speed: The constructor takes the interface as parameter and passes it to the class.

```
1       private void trackBarZoomSpeed_Scroll(object sender, EventArgs e)
2       {
3         /*Instantiate class*/
4         ZoomSettings zS = new ZoomSettings(comIf);
5         /*Open interface*/
6         if (connOpen())
7           return;
8         /*Send command*/
9         zS.ZoomSliderSpeed(trackBarZoomSpeed);
10        // Sets the trackbar(zoomPos) on new Value
11        trackBarZoomPos.Value = zS.ZoomSliderGetPos(trackBarZoomPos);
12        /*Close interface*/
13        connClose();
14        labelActPos.Text = trackBarZoomPos.Value.ToString();     // change text of
              label to position
15      }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.

2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

The eventhandler of the trackbar contains a event called "Mouse up". On Mouse Up the speed is set to 0.
To create a new command instantiate a new Wolfprot2 class. Define two bytearrays for sending and receiving data.

1. bufSnd: For sending a command to the visualizer
2. bufRecv: For receiving the information from the visualizer.

The command for zoom slider with speed is found in the document "command_list.pdf" The packet for the Zoom Slider with Speed has following byte sequence: 01 20 03 ab d0 d1

1. 01: Bit 0 is the direction bit. 1 equals SET
2. 20: The command for zoom settings
3. 02: Length of data
4. ab: Sets the zoom direction wide or tele
5. d0: Speed hi byte
6. d1: Speed lo byte

Listing illustrating how to send a SET ZOOM SLIDER SPEED command:

```
1     public void ZoomSliderSpeed(TrackBar trBar)
2     {
3       Wolfprot2 wp2 = new Wolfprot2(comIf);
4       /*Declaration*/
5       Byte[] bufSnd = new Byte[6];
6       Byte[] bufRecv = new Byte[40];
7       Boolean iReturn;
8
9       /*Assignment*/
10      bufSnd[0] = 0x01;
11      bufSnd[1] = 0x20;
12      bufSnd[2] = 0x03;
13
14      // Zoom wide
15      if (trBar.Value < 0)
16      {
17        bufSnd[3] = 0x01;
18        bufSnd[4] = 0x00;
19        bufSnd[5] = Convert.ToByte(trBar.Value * -1);
20      }
21      // Zoom tele
22      if (trBar.Value > 0)
23      {
24        bufSnd[3] = 0x02;
25        bufSnd[4] = 0x00;
26        bufSnd[5] = Convert.ToByte(trBar.Value);
27      }
28
29      /*Send command*/
30      iReturn = wp2.execCmd(bufSnd, ref bufRecv);
31      if (iReturn == true)
32      {
33        MessageBox.Show("No communication!", "Error",
34                        MessageBoxButtons.OK,
35                        MessageBoxIcon.Error);
36      }
```

```
37        else
38        {
39          if ((bufRecv[0] & 0x80) == 0x80)
40            wp2.ErrorSummary(bufRecv[2]);
41        }
42      }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x00: Bit 0 is the direction bit. 0 equals GET
2. 0x20: The command for zoom settings
3. 0x00: Length of data

8.4 Zoom Slider Position

In "change zoom to position" move the thumb to a certain position. Analog and digital zoom is implemented differently in EYE12/FB4 and PF1 platforms. In EYE12/FB4 analog and digital zoom ranges are handled separately whereas in PF1 platforms analog and digital zoom are handled together.

Listing for tracking the zoombar position:

```
1       private void trackBarZoomPos_Scroll(object sender, EventArgs e)
2       {
3         /*Instantiate class*/
4         ZoomSettings zS = new ZoomSettings(comIf);
5         /*Open interface*/
6         if (connOpen())
7           return;
8         /*Send command*/
9         zS.ZoomSliderSetPos(trackBarZoomPos);
10        /*Close interface*/
11        connClose();
12        labelActPos.Text = trackBarZoomPos.Value.ToString();     // change text of
              label to position
13      }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.
2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

The implementation uses 2 commands. One for setting the zoom position and one for reading the current position.

1. send slider position
2. get slider position

8.4.1 Set Zoom Slider Position

The zoom slider has two different modes:

1. analog zoom
2. digital zoom

For analog and digital zoom 2 different commands are used. The packet for the Analog Zoom Position has following byte sequence: 01 20 02 d0 d1 The packet for the Digital Zoom Position has following byte sequence: 01 28 02 d0 d1

1. 01: Bit 0 is the direction bit. 1 equals SET
2. 20(28): The special command for zoom settings (digital zoom)
3. 02: Length of data
4. d0: First byte of the value
5. d1: Second byte of the value

Listing illustrating how to send a SET ZOOM SLIDER POSITION command:

```
1     public void ZoomSliderSetPos(TrackBar trBar)
2     {
3       Boolean bDigitalZoomSupported = false;
4
5       Wolfprot2 wp2 = new Wolfprot2(comIf);
6       Boolean iReturn;
7       /*Declaration*/
8       Byte[] bufSndAnalog = new Byte[5];
9       Byte[] bufRecvAnalog = new Byte[3];
10
11      Byte[] bufSndDigital = new Byte[5];
12      Byte[] bufRecvDigital = new Byte[3];
13
14      /*Assignment*/
15      // Analog
16      bufSndAnalog[0] = 0x01;
17      bufSndAnalog[1] = 0x20;
18      bufSndAnalog[2] = 0x02;
19
20      // Digital
21      bufSndDigital[0] = 0x01;
22      bufSndDigital[1] = 0x28;
23      bufSndDigital[2] = 0x02;
24
25      // Check if digital zoom is supported
26      iReturn = wp2.execCmd(bufSndDigital, ref bufRecvDigital);
27      if (iReturn == true)
28      {
29        MessageBox.Show("No communication!", "Error",
30                        MessageBoxButtons.OK,
31                        MessageBoxIcon.Error);
32      }
33      else
34      {
35        if ((bufRecvDigital[0] & 0x80) == 0x80)
36        {
37          if (bufRecvDigital[2] == Wolfprot2.ERR_UNKNOWN_CMD)
38            bDigitalZoomSupported = false;
39          else
40            wp2.ErrorSummary(bufRecvDigital[2]);
41        }
42        else
```

```
43          {
44            bDigitalZoomSupported = true;
45          }
46        }
47        if (bDigitalZoomSupported)
48        {
49          if (trBar.Value > 50)
50          {
51            // In digital zoom range
52            // i: number of bytes the digitalzoom uses for position
53            int i = (trBar.Value - 50) * STEP;
54
55            // Convert a long byte in two short bytes, which the program is able to
                 read
56            bufSndDigital[3] = Convert.ToByte((i & 0x0000FF00) >> 8);
57            bufSndDigital[4] = Convert.ToByte(i & 0x000000FF);
58
59            /*Send command*/
60            iReturn = wp2.execCmd(bufSndDigital, ref bufRecvDigital);
61            if (iReturn == true)
62            {
63              MessageBox.Show("No communication!", "Error",
64                              MessageBoxButtons.OK,
65                              MessageBoxIcon.Error);
66            }
67            else
68            {
69              if ((bufRecvDigital[0] & 0x80) == 0x80)
70                wp2.ErrorSummary(bufRecvDigital[2]);
71            }
72          }
73          else
74          {
75            // In analog zoom range
76            // i: number of bytes the digitalzoom uses for position
77            int i = trBar.Value * STEP;
78
79            // Convert a long byte in two short bytes, which the program is able to
                 read
80            bufSndAnalog[3] = Convert.ToByte((i & 0x0000FF00) >> 8);
81            bufSndAnalog[4] = Convert.ToByte(i & 0x000000FF);
82
83            /*Send command*/
84            iReturn = wp2.execCmd(bufSndAnalog, ref bufRecvAnalog);
85            if (iReturn == true)
86            {
87              MessageBox.Show("No communication!", "Error",
88                              MessageBoxButtons.OK,
89                              MessageBoxIcon.Error);
90            }
91            else
92            {
93              if ((bufRecvAnalog[0] & 0x80) == 0x80)
94                wp2.ErrorSummary(bufRecvAnalog[2]);
95            }
96          }
97        }
98        else
99        {
100         // In analog and digital zoom range
101         // i: number of bytes the digitalzoom uses for position
```

```
102         int i = trBar.Value * STEP2;
103
104         // Convert a long byte in two short bytes, which the program is able to
                read
105         bufSndAnalog[3] = Convert.ToByte((i & 0x0000FF00) >> 8);
106         bufSndAnalog[4] = Convert.ToByte(i & 0x000000FF);
107
108         /*Send command*/
109         iReturn = wp2.execCmd(bufSndAnalog, ref bufRecvAnalog);
110         if (iReturn == true)
111         {
112           MessageBox.Show("No communication!", "Error",
113                           MessageBoxButtons.OK,
114                           MessageBoxIcon.Error);
115         }
116         else
117         {
118           if ((bufRecvAnalog[0] & 0x80) == 0x80)
119             wp2.ErrorSummary(bufRecvAnalog[2]);
120         }
121       }
122     }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x00: Bit 0 is the direction bit. 0 equals GET
2. 0x20(28): The command for zoom settings
3. 0x00: Length of data

8.4.2 Get Zoom Slider Position

The command receives the current position of zoom.

Listing for updating the zoom bar position:

```
1     private void buttonUpdate_Click(object sender, EventArgs e)
2     {
3       // Sets the trackbar on new Value
4       /*Instantiate class*/
5       ZoomSettings zS = new ZoomSettings(comIf);
6       /*Open interface*/
7       if (connOpen())
8         return;
9       /*Send command*/
10      trackBarZoomPos.Value = zS.ZoomSliderGetPos(trackBarZoomPos);
11      /*Close interface*/
12      connClose();
13      labelActPos.Text = trackBarZoomPos.Value.ToString();    // change text of
            label to position
14    }
```

1. /*Instantiate class*/ Collection of Wolfprot commands used in this example.

2. /*Open interface*/ Before sending a command to the visualizer a connection must be established.
3. /*Send command*/ Call the send method.
4. /*Close interface*/ After sending a sequence of commands the interface must be closed.

The packet for "Analog Zoom Position" has following sequence: 00 20 00 The packet for "Digital Zoom Position" has following sequence: 00 28 00

1. 00: Bit 0 is the direction bit. 0 equals GET
2. 20(28): The command for zoom settings (digital zoom)
3. 00: Length of data

Listing illustrating how to send a GET ZOOM SLIDER POSITION command:

```
1     public int ZoomSliderGetPos(TrackBar trBar)
2     {
3       Boolean bDigitalZoomSupported = false;
4
5       Wolfprot2 wp2 = new Wolfprot2(comIf);
6
7       /*Declaration*/
8       Byte[] bufSndAnalog = new Byte[3];
9       Byte[] bufRecvAnalog = new Byte[4];
10
11      Byte[] bufSndDigital = new Byte[3];
12      Byte[] bufRecvDigital = new Byte[4];
13
14      Boolean iReturnAnalog;
15      Boolean iReturnDigital;
16
17      /*Assignment*/
18      bufSndAnalog[0] = 0x00;
19      bufSndAnalog[1] = 0x20;
20      bufSndAnalog[2] = 0x00;
21
22      bufSndDigital[0] = 0x00;
23      bufSndDigital[1] = 0x28;
24      bufSndDigital[2] = 0x00;
25
26      /*Send command*/
27      iReturnAnalog = wp2.execCmd(bufSndAnalog, ref bufRecvAnalog);
28      iReturnDigital = wp2.execCmd(bufSndDigital, ref bufRecvDigital);
29      if (iReturnAnalog == true || iReturnDigital == true)
30      {
31        MessageBox.Show("No communication!", "Error",
32                        MessageBoxButtons.OK,
33                        MessageBoxIcon.Error);
34        return 0;
35      }
36      if ((bufRecvAnalog[0] & 0x80) == 0x80)
37      {
38        wp2.ErrorSummary(bufRecvAnalog[2]);
39        return 0;
40      }
41      // Not all visualizers support digital zoom
42      if ((bufRecvDigital[0] & 0x80) == 0x80)
43      {
44        if (bufRecvDigital[2] == Wolfprot2.ERR_UNKNOWN_CMD)
45          bDigitalZoomSupported = false;
46        else
47        {
```

```
48            wp2.ErrorSummary(bufRecvDigital[2]);
49            return 0;
50          }
51        }
52        else
53        {
54          bDigitalZoomSupported = true;
55        }
56        if (bDigitalZoomSupported)
57        {
58          trBar.Value = (bufRecvAnalog[3] * 256
59                         + bufRecvAnalog[4]
60                         + bufRecvDigital[3] * 256
61                         + bufRecvDigital[4]) / STEP;
62        }
63        else
64        {
65          trBar.Value = (bufRecvAnalog[3] * 256
66                         + bufRecvAnalog[4]) / STEP2;
67        }
68
69        return trBar.Value;
70      }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error.

The reply packet is received in the bytearray "bufRecv".

1. 0x00: Bit 0 is the direction bit. 0 equals GET
2. 0x20(28): The command for zoom settings
3. 0x02: Length of data
4. d0: Value HB
5. d1: Value LB

Chapter 9

Picture Transfer

There are different modes for transferring pictures.

1. Singlecast. Get single pictures over TCP or USB using the Picture Header command.
2. Unicast. Send pictures over UDP to a single IP address. Transfer is started with Unicast Start and stopped with Unicast Stop command on EYE12/FB4 platforms or depending on the IP address in PF1 platforms.
3. Multicast. Send pictures over UDP to multiple clients.

9.1 Singlecast

Singlecast is chosen for receiving pictures one by one. The properties of the picture are set in the Picture Header. The Picture Header allows a user to choose between different resolutions, encodings and quality. Because every picture is getting calculated, this method is slower than multicast/unicast, but it works for multiple clients requesting different encoded pictures. The max number of clients is 64.
The picture is selected by first choosing its properties. The radio buttons determine the quality and encoding. The Get Resolution Table button lists all available resolutions. Not all available resoluions are supported at the same time. The resolutions have an aspect ratio of 4:3, 16:9 and 16:10. The supported aspect ratio depends on the Output setting of the visualizer. If the visualizer for example is set to a 16:9 resolution then only 16:9 aspect ratio is supported for picture transfer. If an unsupported resolution is chosen, then the transferred picture will automatically be converted to the supported aspect ratio.
2 commands are necessary to transfer a picture. First the Picture Header command must be issued and then a Get Picture Block command is issued. The Get Picture Block can be divided to smaller blocks if needed.
By pressing the button GetPicture, the Picture Header is sent as well as the Get Picture Block command which together make up the picture transfer. These 2 commands are included in the method "takeAPic" and "takeAPicV2".

1. PictureHeader (0xB8)
2. GetPictureBlock (0xB9) or GetPictureBlock2 (0xBA)

Listing showing the 2 commands necessary to receive a picture:

```
1     public Byte[] GetPictureV2(Byte picturesettings, int iId)
2     {
3       /*Declaration*/
4       Byte[] bufRecv;
5       Byte[] b;
6
7       /*Send command 0xB8*/
8       b = PictureHeaderV2(picturesettings, iId); // use method PictureHeader
9       if (b == null)
10        return null;
11
12      bufRecv = GetPictureBlockV2(b, iId);
13      if (bufRecv == null)
14        return null;
15
16      return bufRecv;
17    }
```

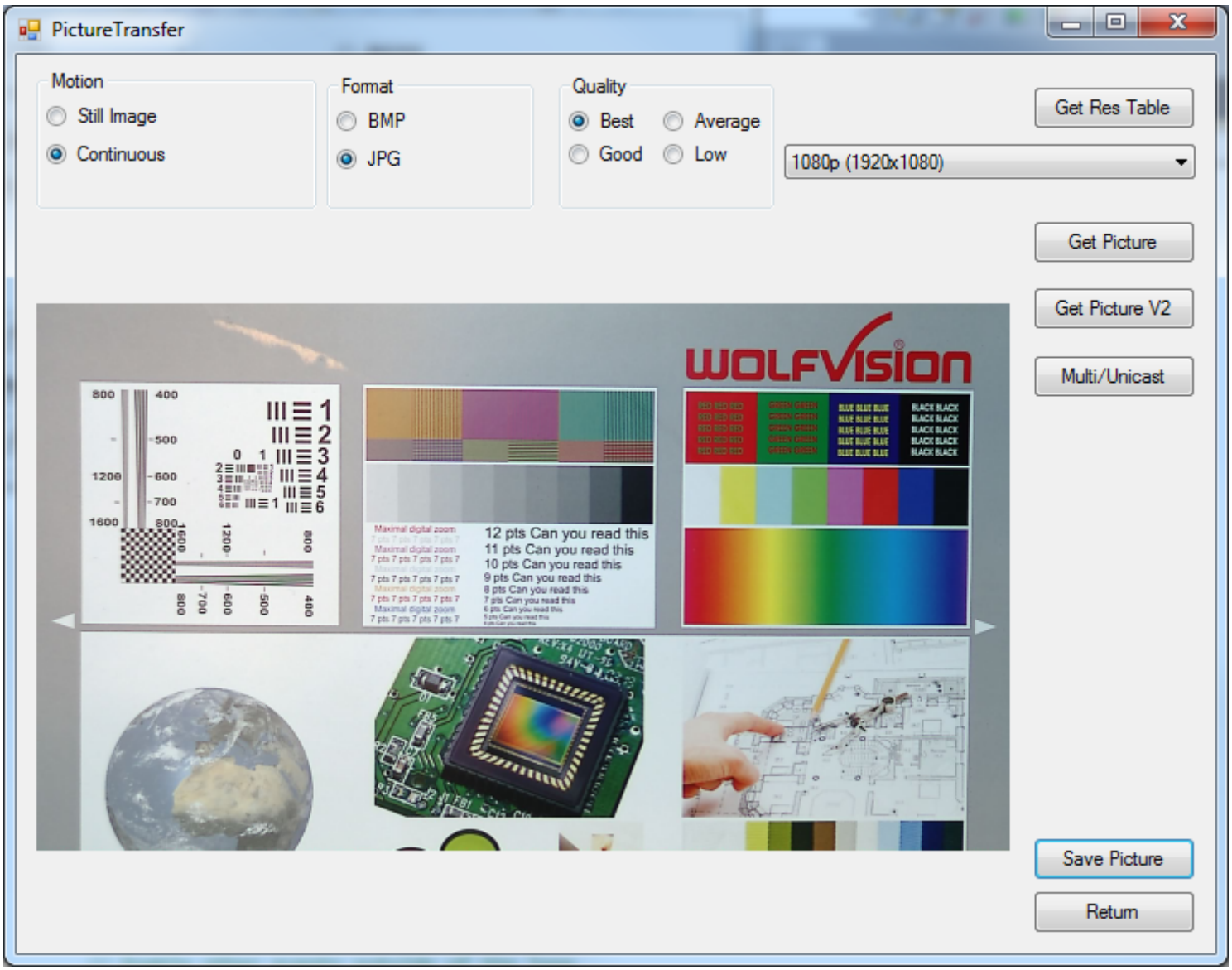

Figure 9.1: Form singlecast picture transfer

There are 2 types of Picture Header commands. The 1st with 1 Information Byte d0 and the second with 5 information bytes d0..d5. The 2nd will be used in the future and is supported on newer WolfVision visualizers. The legacy command is still supported for compatability reasons. The 5 information bytes command extends the original d0 with the ID of the resolution table. The legacy command uses the index of the ID in the resolution table, whereas the new command uses the ID directly.

9.1.1 Picture Header Legacy

The command "Picture Header" defines the picture. The packet for "Picture Header" has following sequence: 00 B8 01 d0.

1. 00: Bit 0 is the direction bit. 0 equals GET
2. B8: The command for picture header
3. 01: Length of data
4. d0: contains the settings of the picture(motion, format, quality, index of ID)

d0:

1. Bit7: Motion: Ethernet: must be 0; USB: 1 or 0, depending on still or continuous image.
2. Bit6: Format: 1=JPG, 0=BMP.

3. Bit5-4: Quality
4. Bit3-0: Index of ID

Listing showing the information that has to be sent in the Picture Header:

```
1          int i;
2          int intpicturesettings = 0;
3          Boolean isBmp;
4
5          // Motion Settings
6          if (rBMotionContinuous.Checked == true)      // when Continuous is used
7          {
8            buttonGetPicture.Text = "STOP";
9            //intpicturesettings |= 0x80;              // dont set the continuous
                flag, this flag is reserved for streaming
10         }
11         else
12           continuousPicture = false;
13
14         // Format Settings
15         if (rBFormatJPG.Checked == true)             // when JPG is used
16         {
17           intpicturesettings |= 0x40;
18           isBmp = false;
19         }
20         else                                         // when BMP is used
21           isBmp = true;
22
23         // Quality Settings
24         if (rBQualityGood.Checked == true)           // when good quality is used
25           intpicturesettings |= 10;
26         else if (rBQualityAverage.Checked == true)   // when average quality is
              used
27           intpicturesettings |= 20;
28         else if (rBQualityLow.Checked == true)       // when low quality is used
29           intpicturesettings |= 30;
30
31         // Index setting
32         i = comboBoxResTable.SelectedIndex;          // index of combobox is
              index of resolution table
33         intpicturesettings |= i & 0x0F;
34
35         // Converts the int number in a Byte
36         Byte picturesettings = Convert.ToByte(intpicturesettings);
```

Listing illustrating how to send the PICTURE HEADER:

```
1      public Byte[] PictureHeader(Byte picturesettings)
2      {
3        Wolfprot2 wp2 = new Wolfprot2(comIf);
4        Boolean iReturn;
5        /*Declaration*/
6        Byte[] bufSnd = new Byte[4];
7        Byte[] bufRecv = new Byte[23];
8
9        /*Assignment*/
10       bufSnd[0] = 0x00;
11       bufSnd[1] = 0xB8;
12       bufSnd[2] = 0x01;
13       bufSnd[3] = picturesettings; // picturesettings get choosen in
            FormPictureSettings
14
```

```
15        /*Send command*/
16        iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17        if (iReturn == true)
18        {
19          MessageBox.Show("No communication!", "Error",
20                          MessageBoxButtons.OK,
21                          MessageBoxIcon.Error);
22          return null;
23        }
24        else if ((bufRecv[0] & 0x80) == 0x80)
25        {
26          wp2.ErrorSummary(bufRecv[2]);
27          return null;
28        }
29
30        return bufRecv;
31      }
```

1. /*Declaration*/ With the information of "command_list.pdf" declare the bytearrays "bufSnd" and "bufRecv" with their defined lengths.
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command*/ Call the command send method. If there is no communication between computer and visualizer or a protocol error, a messagebox shows the error. The result is a block of bytes with information about the picture (width, height, length,...). This received information needs to be filled into "Get Picture Block".

9.1.2 Picture Header V2

The new Picture Header command uses the ID rather than the index of the ID. This makes transferring pictures easier. The new IDs are 4 bytes in length. The packet for "Picture Header" has following sequence: 00 B8 05 d0 d1 d2 d3 d4. d1 to d4 is the ID. The original ID index in d0 is no longer used and its value does not matter. It is recommended to be set to 0.

d0:

1. Bit7: Motion: dont care. Set to 0.
2. Bit6: Format: 1=JPG, 0=BMP.
3. Bit5-4: Quality
4. Bit3-0: dont care. Set to 0.

d1-d4: ID

Listing showing the information that has to be sent with the new picture header command:

```
1         int i;
2         int intpicturesettings = 0;
3         int iId;
4         ResTable clResTable = new ResTable();
5
6         // Motion Settings
7         if (rBMotionContinuous.Checked == true)       // when Continuous is used
8         {
9           buttonGetPictureV2.Text = "STOP";
10          //intpicturesettings |= 0x80;               // dont set the continuous
                flag, reserved for streaming
11        }
12        else
13          continuousPicture = false;
14
15        // Format Settings
16        if (rBFormatJPG.Checked == true)              // when JPG is used
```

```
17          {
18            intpicturesettings |= 0x40;
19          }
20
21          // Quality Settings
22          if (rBQualityGood.Checked == true)           // when good quality is used
23            intpicturesettings |= 10;
24          else if (rBQualityAverage.Checked == true)   // when average quality is
                used
25            intpicturesettings |= 20;
26          else if (rBQualityLow.Checked == true)       // when low quality is used
27            intpicturesettings |= 30;
28
29          // Index setting
30          i = comboBoxResTable.SelectedIndex;          // index of combobox
31          clResTable = (ResTable)rt[i];
32          iId = clResTable.iId;
33
34          // Converts the int number in a Byte
35          Byte picturesettings = Convert.ToByte(intpicturesettings);
```

The following listing illustrates how to send the extended PICTURE HEADER:

```
1       public Byte[] PictureHeaderV2(Byte picturesettings, int iId)
2       {
3         Wolfprot2 wp2 = new Wolfprot2(comIf);
4         Boolean iReturn;
5         /*Declaration*/
6         Byte[] bufSnd = new Byte[8];
7         Byte[] bufRecv = new Byte[23];
8
9         /*Assignment*/
10        bufSnd[0] = 0x00;
11        bufSnd[1] = 0xB8;
12        bufSnd[2] = 0x05;
13        bufSnd[3] = picturesettings;
14        bufSnd[4] = Convert.ToByte((iId & 0xFF000000) >> 24);
15        bufSnd[5] = Convert.ToByte((iId & 0x00FF0000) >> 16);
16        bufSnd[6] = Convert.ToByte((iId & 0x0000FF00) >> 8);
17        bufSnd[7] = Convert.ToByte(iId & 0x000000FF);
18
19        /*Send command*/
20        iReturn = wp2.execCmd(bufSnd, ref bufRecv);
21        if (iReturn == true)
22        {
23          MessageBox.Show("No communication!", "Error",
24                          MessageBoxButtons.OK,
25                          MessageBoxIcon.Error);
26          return null;
27        }
28        else if ((bufRecv[0] & 0x80) == 0x80)
29        {
30          wp2.ErrorSummary(bufRecv[2]);
31          return null;
32        }
33
34        return bufRecv;
35      }
```

9.1.3 Get Picture Block

This method allows a pictureblock to be received. The information received by sending out "Picture Header" has to be passed on to "Get Picture Block". The whole request is 25 bytes long and the reply is a block of bytes that contains a complete picture.
Note: For PF1 platform devices this command is supported for both USB and ETH and the command 0xB9 is no longer supported. Also, pictures on PF1 platforms are sent complete, that means including the header. Also the count is now in bytes and not in lines.

The following table lists the parameters that have to be sent out depending on the interface to receive a complete picture in one block:

Platform	Format	Interface	Command	Offset	Max Block-length	DMA Size	Start Index	Reply
EYE12/FB4	JPG	ETH	BA	0	Length	Length	7	HHCLLLL[Pic]
EYE12/FB4	JPG	USB	B9	0	Length	Length	0	[Pic]HCLD
EYE12/FB4	BMP	ETH	BA	0	Length+54	Length+54	7	HHCLLLL[Pic]
EYE12/FB4	BMP	USB	B9	0	Length	Length	0	[Pic!]HCLD
PF1	JPG	ETH	BA	0	Length	Length	7	HHCLLLL[Pic]
PF1	JPG	USB	BA	0	Length	Length	7	HHCLLLL[Pic]
PF1	BMP	ETH	BA	0	Length	Length	7	HHCLLLL[Pic]
PF1	BMP	USB	BA	0	Length	Length	7	HHCLLLL[Pic]

Table 9.1: Picture parameters

Legend:

1. Length; length received in GET PICTURE HEADER.
2. Length + 54; length of BMP header added.
3. Pic!; Picture without header and lines reversed. The BMP header must be added manually after reception of the block, also the lines of a picture must be reordered.
4. HHCLLLL; 2 byte header, 1 byte command, 4 byte length at the beginning of the block.
5. HCLD; 1 byte header, 1 byte command, 1 byte length, 1 byte data at the end of the block.

Listing illustrating the new method how to receive a complete PICTURE BLOCK in singlecast mode:

```
1     public Byte[] GetPictureBlockV2(Byte[] pic_header, int iId)
2     {
3       Wolfprot2 wp2 = new Wolfprot2(comIf);
4       Boolean iReturn;
5
6       /*Declaration*/
7       Byte[] bufSnd = new Byte[25];
8       Byte[] bufRecv = new Byte[6220800 + 54 + 7]; //bmp + header
9
10      /*Assignment*/
11      bufSnd[0] = 0x00;
12      bufSnd[1] = 0xBA;
13      bufSnd[2] = 0x16;
14
15      // Flags
16      bufSnd[3] = pic_header[7];
17
18      // Mode Byte
19      bufSnd[4] = pic_header[8];
20
21      // Offset
22      bufSnd[5] = 0x00;
23      bufSnd[6] = 0x00;
```

```
24        bufSnd[7] = 0x00;
25        bufSnd[8] = 0x00;
26
27        // Max blocklength
28        int blocklength;
29
30        blocklength = Convert.ToInt32(pic_header[15] << 24)
31                    + Convert.ToInt32(pic_header[16] << 16)
32                    + Convert.ToInt32(pic_header[17] << 8)
33                    + Convert.ToInt32(pic_header[18]);
34        if (blocklength == 0)
35          return null;
36
37
38        bufSnd[9] = Convert.ToByte((blocklength & 0xFF000000) >> 24);
39        bufSnd[10] = Convert.ToByte((blocklength & 0x00FF0000) >> 16);
40        bufSnd[11] = Convert.ToByte((blocklength & 0x0000FF00) >> 8);
41        bufSnd[12] = Convert.ToByte(blocklength & 0x000000FF);
42
43        // Width
44        bufSnd[13] = pic_header[3];
45        bufSnd[14] = pic_header[4];
46
47        // Height
48        bufSnd[15] = pic_header[5];
49        bufSnd[16] = pic_header[6];
50
51        // Total Size
52        bufSnd[17] = pic_header[15];
53        bufSnd[18] = pic_header[16];
54        bufSnd[19] = pic_header[17];
55        bufSnd[20] = pic_header[18];
56
57        // Memory Page
58        bufSnd[21] = pic_header[19];
59        bufSnd[22] = pic_header[20];
60
61        // Out Offset
62        bufSnd[23] = pic_header[21];
63        bufSnd[24] = pic_header[22];
64
65        /*Send command 0xBA*/
66        iReturn = wp2.execCmdFast(bufSnd, ref bufRecv, blocklength + 7);
67        //iReturn = wp2.execCmd(bufSnd, ref bufRecv);
68
69        if (iReturn == true)
70        {
71          MessageBox.Show("No communication!", "Error",
72                          MessageBoxButtons.OK,
73                          MessageBoxIcon.Error);
74          return null;
75        }
76        else if ((bufRecv[0] & 0x80) == 0x80)
77        {
78          wp2.ErrorSummary(bufRecv[2]);
79          return null;
80        }
81
82        return bufRecv;
83      }
```

1. /*Declaration*/ Declare buffer big enough to hold a complete picture
2. /*Assignment*/ Fill the bufSnd byte array with the command.
3. /*Send command 0xBA or 0xB9*/ Receive the picture

9.2 Multicast and Unicast

Multicast and Unicast streaming are faster than Singlecast. With these types of streaming, the picture settings stay the same resulting in less picture encoding effort. The difference between Multicast and Unicast is that in Unicast mode a start packet needs to be sent, and for finishing a stop packet. These two packets are sent with the "TCP" protocol at port 50915. Multicast and Unicast use the "UDP" protocol, because it has no handshake. Packets are sent on a port in the range between "8800" and "9000". Multicast IP range is from "224.0.0.0" and "239.255.255.255". The 224.x.x.x and 239.x.x.x ranges should not be used because they are reserved for special purposes.
Note: PF1 platform visualizers support Unicast in a different form. The start and stop packages have been obsoleted and the visualizer decides Unicast or Multicast mode by the selected IP address in the OSD menu.

9.2.1 Form MultiUnicast

The class "MultiUnicast" contains the commands to change the settings of the visualizer (streaming mode, port, and ip[1]) and also the "start" and "stop" command for Unicast.

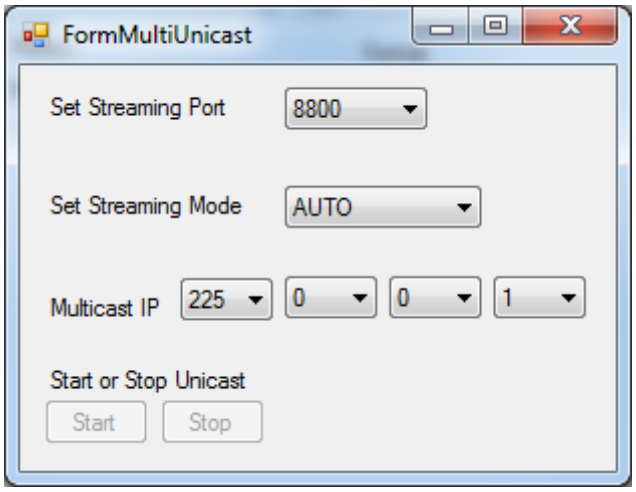

Figure 9.2: No Multi or Unicast stream

[1] only for Multicast

9.2.2 VLC Player

Besides the Connectivity software from WolfVision a third party media player can be used to see the Multi- or Unicast stream from the visualizer. The media player that supports all modes of the visualizer is the VLC media player. There is the possibilty to include the VLC media player as a plugin into the c# program. The easier way is to start the VLC media player externally and set the required settings for a network stream. If a third party media player is used than the visualizers streaming mode must be set to CONTINUOUS in the OSD menu.

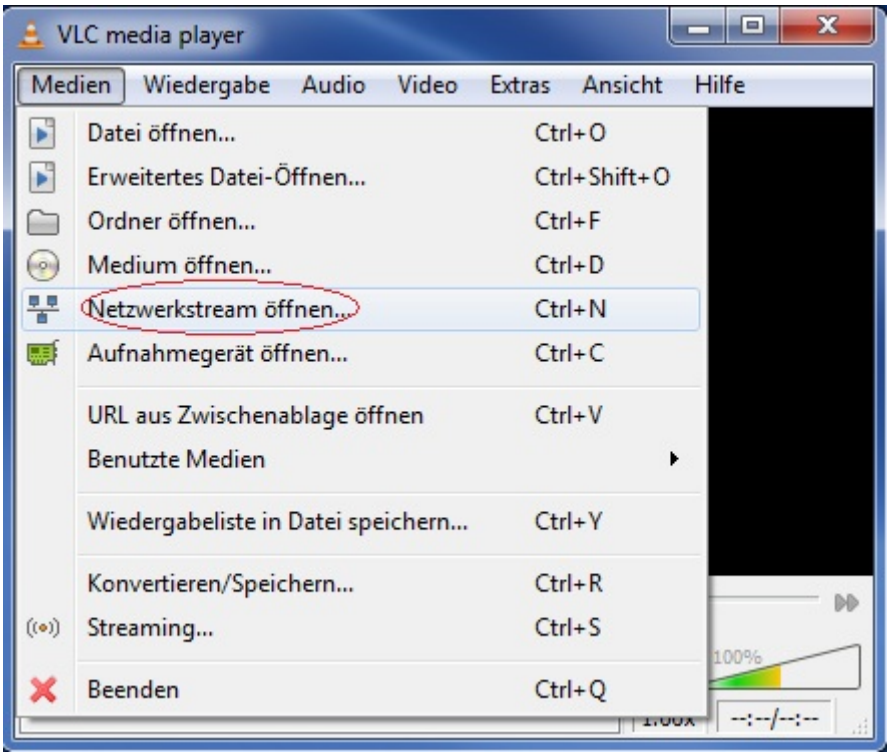

Figure 9.3: Open network stream

The connection method is different, according to the streaming mode (Multi- or Unicast).

Multicast

In case of "Multicast", you have to set "UDP" protocol, Multicast IP and port.

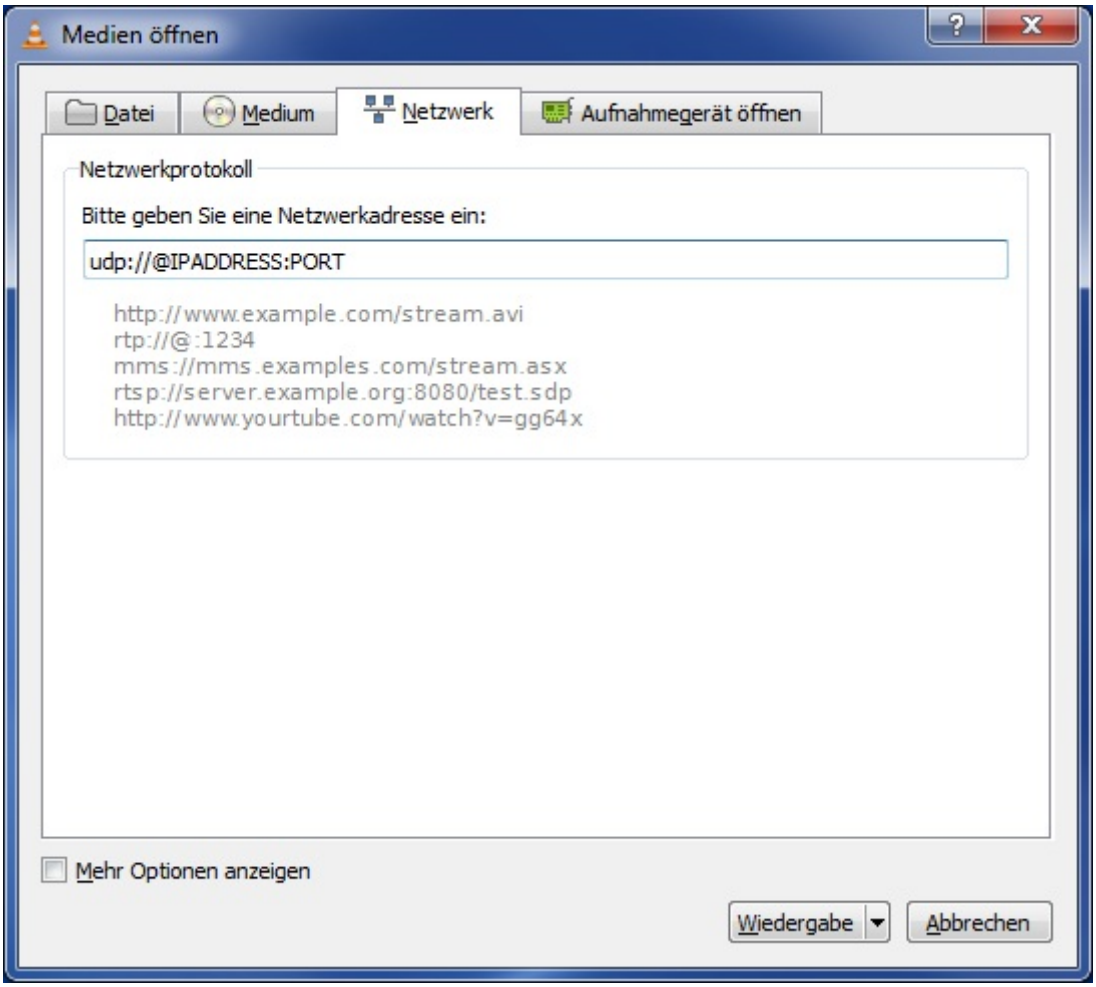

Figure 9.4: Include Multicast

Unicast

In case of "Unicast", you have to set "UDP" protocol and port.

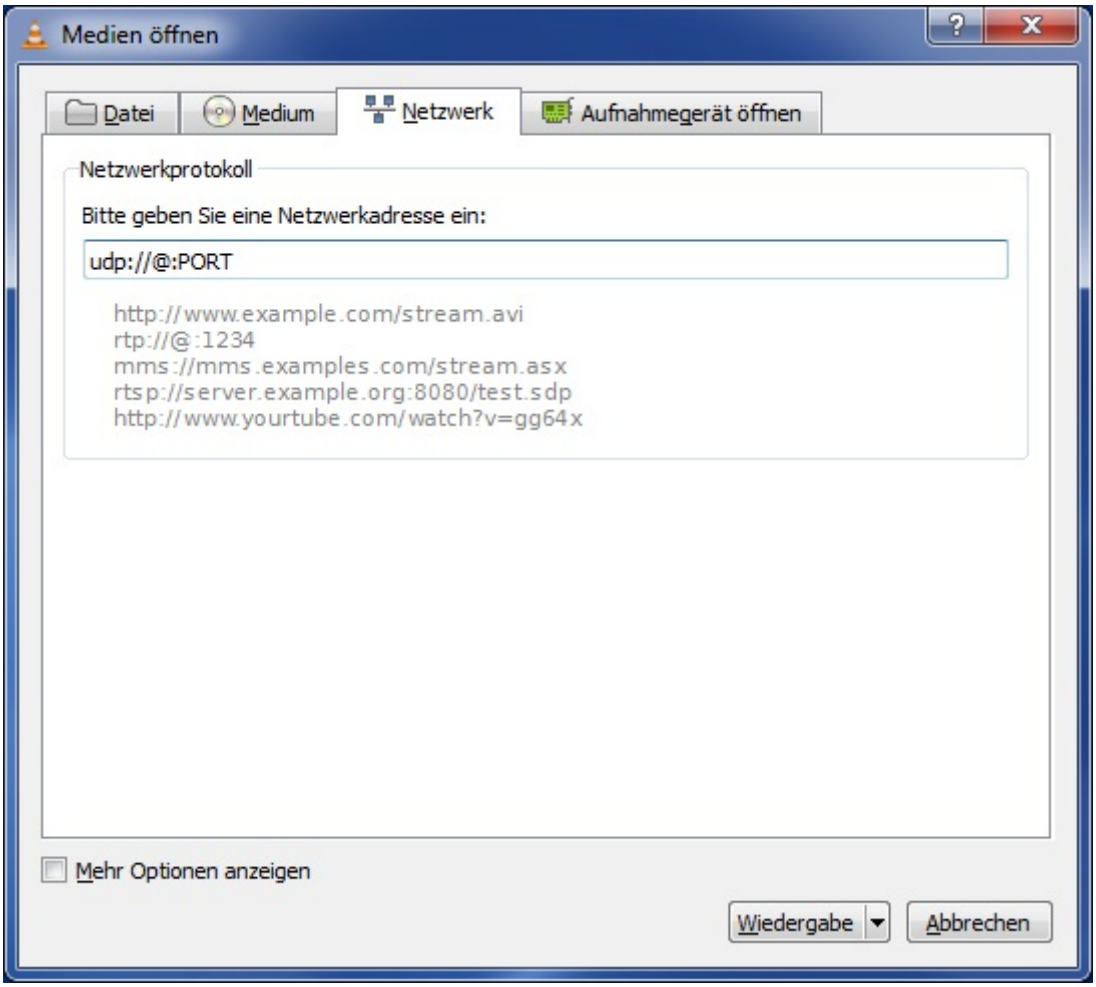

Figure 9.5: Include Unicast

Chapter 10

Authentication

Authentication is only available for Ethernet. Authentication is recommended if the visualizer is connected via Internet. When using authentication it is important that the connection is kept the same for the entire session. In order to enable authentication it is necessary to switch the visualizer to Ethernet Mode "AUTH" in the OSD menu.
On the Authentication example form there are 2 textboxes for entering username and password and 2 buttons for login and logout. The 2 supported usernames are "admin" and "guest". "admin" has CTRL, IMG and FW access level, whereas "guest" has IMG only access level. In a regular application after login the necessary commands would be issued. The session should end with the logout action.

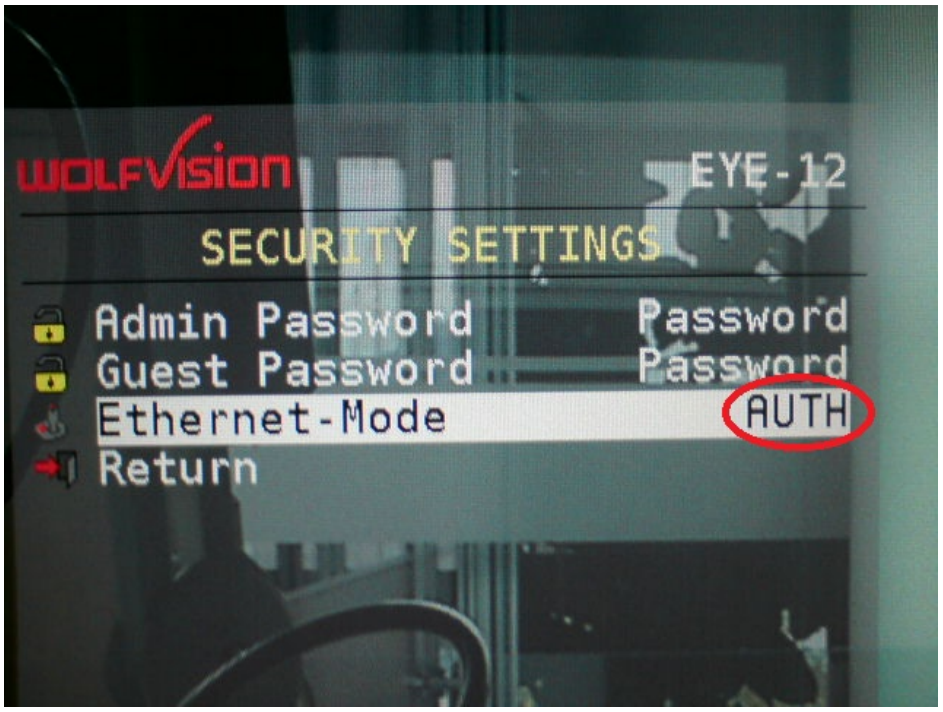

Figure 10.1: Ethernet settings

For security reasons, the password is sent encoded as a " md5 hash code". The hash code contains 4 parameters:

- session ID[1]
- username
- password
- IP address

Listing illustrating how to collect the data for LOGIN:

```
1     private void buttonLogin_Click(object sender, EventArgs e)
2     {
```

[1]is a 4 byte random number sent from the visualizer

```
3         // Enables to convert string into byte array
4         System.Text.Encoding enc = System.Text.Encoding.ASCII;
5
6         /*Collect username,password and IP*/
7         Byte[] user = enc.GetBytes(tbUsername.Text);
8         Byte[] passw = enc.GetBytes(tbPassword.Text);
9
10        Byte[] ip;
11        System.Net.IPAddress Ip;
12        System.Net.IPAddress.TryParse(handbook.sendIP(), out Ip);
13        ip = Ip.GetAddressBytes();
14
15        /*Padding*/
16        Byte[] username = new Byte[16];
17        // Because the username must have a length of 16 Bytes,
18        // the missing Bytes have to be filled up with "0"
19        if (user.Length < tbUsername.MaxLength)
20        {
21
22          for (int i = 0; i < username.Length; i++)
23          {
24            // Fill up the username with the converted Bytes of the
25            // string from the textbox "tbUsername"
26            if (i < user.Length)
27              username[i] = user[i];
28            // Fill up the rest of Bytes with "0"
29            else
30              username[i] = 0x00;
31          }
32        }
33
34        Byte[] password = new Byte[10];
35        // Because the password must have a length of 10 Bytes,
36        // the missing Bytes have to be filled up with "0"
37        if (passw.Length < tbPassword.MaxLength)
38        {
39          for (int i = 0; i < password.Length; i++)
40          {
41            // Fill up the password with the converted Bytes of the
42            // string from the textbox "tbPassword"
43            if (i < passw.Length)
44              password[i] = passw[i];
45            // Fill up the rest of Bytes with "0"
46            else
47              password[i] = 0x00;
48          }
49        }
50        /*Send commands*/
51        Authentication auth = new Authentication(comIf);
52        if (connOpen())
53          return;
54        if (auth.Login(username, password, ip))  //only in case of an error close
              the connection, otherwise leave it open to be able to send out more
              commands.
55          connClose();
56      }
```

1. /*Collect username,password and IP*/ Convert strings to byte arrays
2. /*Padding*/ Pad with 0s
3. /*Send commands*/ Send Get Sessin ID and Login command

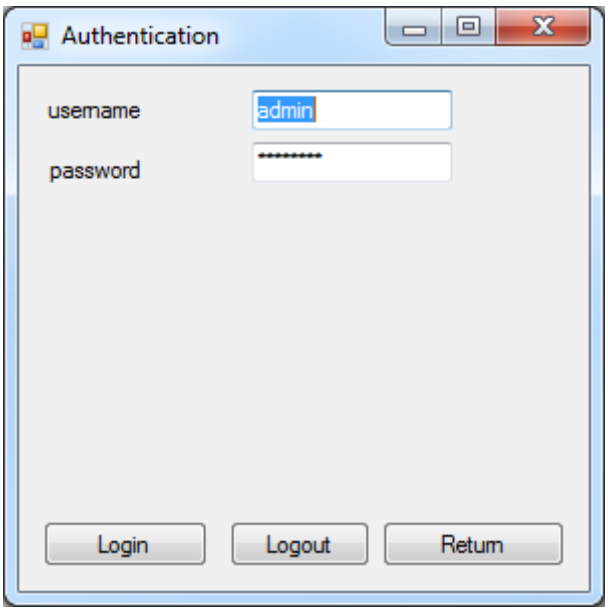

Figure 10.2: Form authentication

For login the first action is to ask for a session ID. Then the encrypted password and username is sent to the visualizer encoded with this session ID. The visualizer computes its own hashcode and compares it with the sent hashcode. If they match login will be successful.
CAUTION: When authentication is used all commands must be sent in a single session, that means the connection has to be opened, then commands are issued. At the end of the session the connection has to be closed. If the connection is closed inbetween, authentication is lost and error code "AUTH REQUIRED" is received.

10.1 Get Session ID

The session ID is a 4 byte random number. You need it for the md5 hash code to encode the password.

Listing illustrating how to receive the SESSION ID:

```
1     public Byte[] Request_SessionID()
2     {
3       /*Declaration*/
4       Wolfprot2 wp2 = new Wolfprot2(comIf);
5       Byte[] bufSnd = new Byte[3];
6       Byte[] bufRecv = new Byte[7];
7       Byte[] sessionID = new Byte[4];
8       Boolean iReturn;
9
10      /*Assignment*/
11      bufSnd[0] = 0x00;
12      bufSnd[1] = 0xD3;
13      bufSnd[2] = 0x00;
14
15      /*Send command*/
16      iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17      if (iReturn == true)
18      {
19        MessageBox.Show("No communication!", "Error",
20                        MessageBoxButtons.OK,
21                        MessageBoxIcon.Error);
22        return null;
23      }
24      else if ((bufRecv[0] & 0x80) == 0x80)
25      {
26        wp2.ErrorSummary(bufRecv[2]);
```

```
27          return null;
28        }
29
30        else
31        {
32          // Take the session ID out of the
33          // received bytearray
34          for (int i = 0; i < sessionID.Length; i++)
35          {
36            sessionID[i] = bufRecv[i + 3];
37          }
38        }
39
40        return sessionID;
41      }
```

1. /*Declaration*/ Define the necessary Wolfprot variables
2. /*Assignment*/ Fill in the command
3. /*Send command*/ Command with error handling

Be aware again that the connection must not be closed after sending this command, because closing and reopening the connection would result in a different session ID.

10.2 Login

With the session ID, the username, the password and IP address the MD5 hash code is calculated.

Listing illustrating how to send the LOGIN:

```
1       public Boolean Login(Byte[] username, Byte[] password, Byte[] ip)
2       {
3         /*Declaration*/
4         Wolfprot2 wp2 = new Wolfprot2(comIf);
5         Byte[] bufSnd = new Byte[39];
6         Byte[] bufRecv = new Byte[3];
7         Byte[] sessionID;
8         Byte[] md5Out;
9         Boolean iReturn;
10
11        /*Send command 0xD3*/
12        sessionID = Request_SessionID();
13        if (sessionID == null)
14          return true;
15
16        /*Calc MD5*/
17        ArrayList aList = new ArrayList();
18
19        // Create the array list for the MD5 Hash Code
20        aList.AddRange(sessionID);
21        aList.AddRange(username);
22        aList.AddRange(password);
23        aList.AddRange(ip);
24
25        // Create the Hash Code
26        System.Security.Cryptography.MD5CryptoServiceProvider x = new
              System.Security.Cryptography.MD5CryptoServiceProvider();
27        md5Out = x.ComputeHash((Byte[])aList.ToArray(typeof(Byte)));
28
29        // Clear Arraylist
```

```
30        aList.Clear();
31        // Initialising the command
32        aList.AddRange(new Byte[] { 0x01, 0xD4, 0x24 });
33        // Initialising the md5 to the command
34        aList.AddRange(md5Out);
35        // Initialising the username to the command
36        aList.AddRange(username);
37        // Initialising the IP Adress to the command
38        aList.AddRange(ip);
39
40        bufSnd = (Byte[])aList.ToArray(typeof(Byte));
41
42        /*Send command 0xD4*/
43        iReturn = wp2.execCmd(bufSnd, ref bufRecv);
44        if (iReturn == true)
45        {
46          MessageBox.Show("No communication!", "Error",
47                          MessageBoxButtons.OK,
48                          MessageBoxIcon.Error);
49          return true;
50        }
51        else if ((bufRecv[0] & 0x80) == 0x80)
52        {
53          wp2.ErrorSummary(bufRecv[2]);
54          return true;
55        }
56        else
57          MessageBox.Show("Login successful");
58
59        return false;
60      }
```

1. /*Declaration*/ Define the necessary Wolfprot variables
2. /*Send command 0xD3*/ Get the session ID
3. /*Calc MD5*/ Calculate the hash code
4. /*Send command 0xD4*/ Command with error handling

10.3 Logout

To logout you have first to login successfully, or there will be an error message that you have no communication. Closing the interface will have the same effect as sending logout.

Listing illustrating how to send the LOGOUT:

```
1       public Boolean Logout()
2       {
3         /*Declaration*/
4         Wolfprot2 wp2 = new Wolfprot2(comIf);
5         Byte[] bufSnd = new Byte[4];
6         Byte[] bufRecv = new Byte[3];
7         Boolean iReturn;
8
9         /*Assignment*/
10        bufSnd[0] = 0x01;
11        bufSnd[1] = 0xD5;
12        bufSnd[2] = 0x01;
13        bufSnd[3] = 0x00;
14
15        /*Send command*/
```

```
16        iReturn = wp2.execCmd(bufSnd, ref bufRecv);
17        if (iReturn == true)
18        {
19          MessageBox.Show("No communication!", "Error",
20                          MessageBoxButtons.OK,
21                          MessageBoxIcon.Error);
22          return true;
23        }
24        else if ((bufRecv[0] & 0x80) == 0x80)
25        {
26          wp2.ErrorSummary(bufRecv[2]);
27          return true;
28        }
29        else
30          MessageBox.Show("Logout successful");
31
32        return false;
33      }
```

1. /*Declaration*/ Define the necessary Wolfprot variables
2. /*Assignment*/ Fill in the command
3. /*Send command*/ Command with error handling

After sending the logout command the interface can be closed.

Chapter 11

XML File

The visualizer holds an XML file for the creation of the OSD menu. It is possible to download this file from the visualizer. In the demo application the file can be stored to disc. The form only contains a button "read XML file"

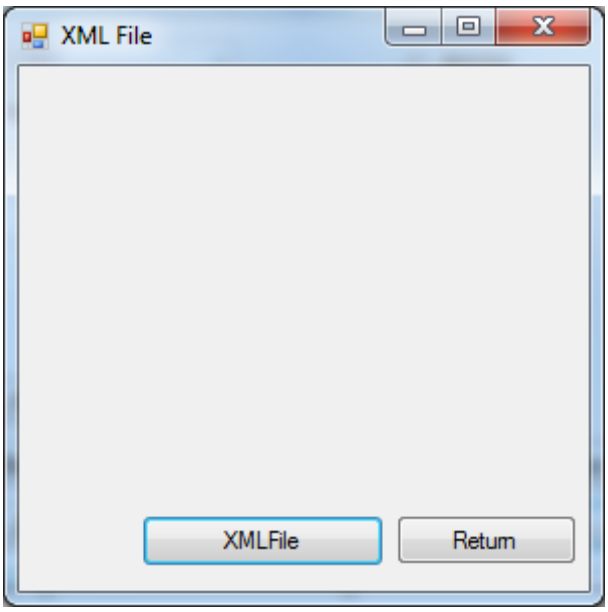

Figure 11.1: Form XML file

Listing showing the command for reading the XML file:

```
1      public Boolean readXMLFile(string path)
2      {
3        Wolfprot2 wp2 = new Wolfprot2(comIf);
4        Byte[] bufSnd = new Byte[3];
5        Byte[] bufRecv = new Byte[10000000]; //10MB
6        Boolean iReturn;
7
8        bufSnd[0] = 0x00;
9        bufSnd[1] = 0xCA;
10       bufSnd[2] = 0x00;
11
12       iReturn = wp2.execCmd(bufSnd, ref bufRecv);
13       if (iReturn == true)
14       {
15         MessageBox.Show("No communication!", "Error",
16                         MessageBoxButtons.OK,
17                         MessageBoxIcon.Error);
18         return true;
19       }
20       else
```

```
21        {
22          if ((bufRecv[0] & 0x80) == 0x80)
23          {
24            wp2.ErrorSummary(bufRecv[2]);
25            return true;
26          }
27          else
28          {
29            // Read the Bytes for the length of the XML file
30            // from bufRecv
31            Byte[] bytelength = new Byte[4];
32
33            for (int i = 0; i < bytelength.Length; i++)
34            {
35              bytelength[i] = bufRecv[i + 3];
36            }
37
38            // Convert the byte array to a integer
39            uint XMLFilelength = Convert.ToUInt32(bytelength[0] << 24)
40                                 + Convert.ToUInt32(bytelength[1] << 16)
41                                 + Convert.ToUInt32(bytelength[2] << 8)
42                                 + Convert.ToUInt32(bytelength[3] << 0);
43
44            Byte[] XMLfile = new Byte[XMLFilelength];
45
46            // Read the Bytes of the XML file from bufRecv
47            for (int i = 0; i < XMLfile.Length; i++)
48            {
49              XMLfile[i] = bufRecv[i + 7];
50            }
51
52            ByteArrayToFile(path + "\\File.xml", XMLfile);
53          }
54        }
55        return false;
56      }
```

Chapter 12

Firmware Update

WolfVision visualizers support "Firmware Update" of the devices. At present there are 2 different firmware file formats used. EYE12/FB4 1st generation equipment use files that have the ending w19, and PF1 2nd generation equipment use files with the ending wgz. The commands for sending both file formats are the same to keep the commands compatible. The order the commands are sent though has changed. Therefore the examples are split into 2 different state machines. One for loading FB4 and EYE12 firmware files and one for loading PF1 firmware files.

12.1 FB4 and EYE12 Algorithm

FB4 and EYE12 loading algorithm:

1. Check Buildnumber
2. (Downgrade)
3. Erase Flash
4. Poll Erase Flash
5. Upload Data Start
6. Upload Data
7. Upload Data Stop

Listing showing the state machine for FB4 and EYE12 based hardware:

```
1     public void StateMachineFb4Eye12()
2     {
3       int iUpDowngrade = 0;
4
5       // Enable the commands
6       FirmwareUpdate fWUpdate = new FirmwareUpdate(comIf);
7
8       /*step0*/
9       if (step == 0)
10      {
11        if (FWUpOrDowngradeControl(ref iUpDowngrade))
12        {
13          timerFWUpdate.Stop();
14          connClose();
15        }
16        switch (iUpDowngrade)
17        {
18          case 0:
19            step = 0;
20            timerFWUpdate.Stop();
21            connClose();
22            break;
```

```
23            case -1:  //Downgrade
24              step++;
25              break;
26            case 1:   //Upgrade
27              step += 2;
28              break;
29          }
30        }
31        // Enable downgrade if needed
32        /*step1*/
33        else if (step == 1)
34        {
35          // Increment step to go to next command
36          step++;
37
38          // Initialize the method to erase the flash
39          // and stop the timer in case of an error
40          if (fWUpdate.FWDowngrade())
41          {
42            timerFWUpdate.Stop();
43            connClose();
44          }
45        }
46        // Start the erase flash
47        /*step2*/
48        else if (step == 2)
49        {
50          // Increment step to go to next command
51          step++;
52
53          // Initialize the method to erase the flash
54          // and stop the timer in case of an error
55          if (fWUpdate.UploadEraseFlash())
56          {
57            timerFWUpdate.Stop();
58            connClose();
59          }
60        }
61        // Get the state of erasing flash
62        // to know when it's finished
63        /*step3*/
64        else if (step == 3)
65        {
66          // Change the text of the label
67          labelUpdateState.Text = "Update: Erasing ... "
68                                  + fWUpdate.GetUploadEraseFlash().ToString() + "%";
69
70          /* Change the value of the progress bar
71           * and initialize the method for getting the
72           * state of erasing flash if the received is
73           * not an error number (200 choosen)
74           */
75          if (fWUpdate.GetUploadEraseFlash() > 100)
76          {
77            timerFWUpdate.Stop();
78            connClose();
79          }
80          else
81            progressBar1.Value = fWUpdate.GetUploadEraseFlash();
82
83          if (progressBar1.Value == 100)
```

```
84          {
85            // Increment step to go to next command
86            step++;
87
88            // Change the text of the label
89            labelUpdateState.Text = "Update: Erasing ... "
90                                    + fWUpdate.GetUploadEraseFlash().ToString() +
                                        "%";
91          }
92        }
93        // Start the upload of the new firmware
94        /*step4*/
95        else if (step == 4)
96        {
97          // Increment step to go to the next command
98          step++;
99
100         Byte[] fWDataLength = new Byte[4];
101         fWDataLength[0] = Convert.ToByte(fWData.Length >> 24);
102         fWDataLength[1] = Convert.ToByte((fWData.Length & 0x00FF0000) >> 16);
103         fWDataLength[2] = Convert.ToByte((fWData.Length & 0x0000FF00) >> 8);
104         fWDataLength[3] = Convert.ToByte(fWData.Length & 0x000000FF);
105
106         // Initialize the method to start new firmware
107         if (fWUpdate.UploadDataStart(fWDataLength))
108         {
109           timerFWUpdate.Stop();
110           connClose();
111         }
112       }
113       // Upload the new firmware
114       /*step5*/
115       else if (step == 5)
116       {
117         // Increment step to go to the next command
118         step++;
119         /* Initialize the method to generate the needed information
120          * to upload the new firmware
121          */
122         if (Upload("Programming"))
123         {
124           timerFWUpdate.Stop();
125           connClose();
126         }
127       }
128       // Stop/finish the firmware update
129       /*step6*/
130       else if (step == 6)
131       {
132         // Change the text of the label
133         labelUpdateState.Text = "Update: Finished";
134
135         // Stop the timer
136         timerFWUpdate.Stop();
137
138         // Reset the step
139         step = 0;
140
141         // Initialize the method to stop the firmware update
142         fWUpdate.UploadStop();
143         /*Close interface*/
```

```
144           connClose();
145           // Show a message box to inform that the
146           // firmware update has finished
147           MessageBox.Show("Update finished." +
148                           "\nThe visualizer restarts in 30 - 150 seconds.");
149         }
150       }
```

1. /*step0*/ Decide if upgrade or downgrade by reading the buildnumber (0x1C).
2. /*step1*/ Send the downgrade command (0xB4) in case buildnumber of file is smaller than buildnumber of visualizer.
3. /*step2*/ Send the Erase Flash command (0xB0).
4. /*step3*/ Poll until flash is erased (0xB0).
5. /*step4*/ Send the Firmware Upload Data Start command (0xB1).
6. /*step5*/ Send Firmware Upload Data blocks (0xB2).
7. /*step6*/ Send Firmware Upload Data Stop command (0xB3).

12.2 PF1 Algorithm

PF1 loading algorithm:

1. Check Buildnumber
2. Upload Data Start
3. Upload Data
4. Upload Data Stop
5. Erase Flash
6. Poll Erase Flash

Listing showing the state machine for PF1 based hardware:

```
1       public void StateMachinePf1()
2       {
3         int iUpDowngrade = 0;
4         // Enable the commands
5         FirmwareUpdate fWUpdate = new FirmwareUpdate(comIf);
6
7         // Initialize control if down or upgrade
8         /*step0*/
9         if (step == 0)
10        {
11          if (FWUpOrDowngradeControl(ref iUpDowngrade))
12          {
13            timerFWUpdate.Stop();
14            connClose();
15          }
16          switch (iUpDowngrade)
17          {
18            case 0:
19              step = 0;
20              timerFWUpdate.Stop();
21              connClose();
22              break;
23            case -1:  // Downgrade there is no downgrade command in PF1
24            case 1:   // Upgrade
25              step++;
```

```
26              break;
27            }
28          }
29          /*step1*/
30          else if (step == 1)
31          {
32            // Increment step to go to the next command
33            step++;
34
35            Byte[] fWDataLength = new Byte[4];
36            fWDataLength[0] = Convert.ToByte(fWData.Length >> 24);
37            fWDataLength[1] = Convert.ToByte((fWData.Length & 0x00FF0000) >> 16);
38            fWDataLength[2] = Convert.ToByte((fWData.Length & 0x0000FF00) >> 8);
39            fWDataLength[3] = Convert.ToByte(fWData.Length & 0x000000FF);
40
41            // Initialize the method to start new firmware
42            if (fWUpdate.UploadDataStart(fWDataLength))
43            {
44              timerFWUpdate.Stop();
45              connClose();
46            }
47          }
48          // Upload the new firmware
49          /*step2*/
50          else if (step == 2)
51          {
52            // Increment step to go to the next command
53            step++;
54            /* Initialize the method to generate the needed information
55             * to upload the new firmware
56             */
57            if (Upload("Loading"))
58            {
59              timerFWUpdate.Stop();
60              connClose();
61            }
62          }
63          // Stop/finish the firmware update
64          /*step3*/
65          else if (step == 3)
66          {
67            step++;
68
69            // Initialize the method to stop the firmware update
70            if (fWUpdate.UploadStop())
71            {
72              timerFWUpdate.Stop();
73              connClose();
74            }
75          }
76          // Start the programming
77          /*step4*/
78          else if (step == 4)
79          {
80            // Increment step to go to next command
81            step++;
82
83            // Initialize the method to erase the flash
84            // and stop the timer in case of an error
85            if (fWUpdate.UploadEraseFlash())
86            {
```

```
87            timerFWUpdate.Stop();
88            connClose();
89          }
90        }
91        // Get the state of programming
92        /*step5*/
93        else if (step == 5)
94        {
95          // Change the text of the label
96          labelUpdateState.Text = "Update: Programming ... "
97                                  + fWUpdate.GetUploadEraseFlash().ToString() + "%";
98
99          /* Change the value of the progress bar
100          * and initialize the method for getting the
101          * state of erasing flash if the received is
102          * not an error number (200 choosen)
103          */
104         if (fWUpdate.GetUploadEraseFlash() > 100)
105         {
106           timerFWUpdate.Stop();
107           connClose();
108         }
109         else
110           progressBar1.Value = fWUpdate.GetUploadEraseFlash();
111
112         if (progressBar1.Value == 100)
113         {
114           // Increment step to go to next command
115           step++;
116
117           // Change the text of the label
118           labelUpdateState.Text = "Update: Programming ... "
119                                   + fWUpdate.GetUploadEraseFlash().ToString() +
                                        "%";
120         }
121       }
122       // Start the upload of the new firmware
123       // stop/finish the firmware update
124       /*step6*/
125       else if (step == 6)
126       {
127         // Change the text of the label
128         labelUpdateState.Text = "Update: Finished";
129
130         // Stop the timer
131         timerFWUpdate.Stop();
132
133         // Reset the step
134         step = 0;
135
136         /*Close interface*/
137         connClose();
138         // Show a message box to inform that the
139         // firmware update has finished
140         MessageBox.Show("Update finished." +
141                         "\nThe visualizer restarts in 30 - 150 seconds.");
142       }
143     }
```

1. /*step0*/ Decide if upgrade or downgrade by reading the buildnumber (0x1C).

2. /*step1*/ Send the Firmware Upload Data Start command (0xB1).
3. /*step2*/ Send Firmware Upload Data blocks (0xB2).
4. /*step3*/ Send Firmware Upload Data Stop command (0xB3).
5. /*step4*/ Send the Erase Flash command (0xB0), this triggers programming of flash.
6. /*step5*/ Poll until flash is programmed (0xB0).
7. /*step6*/ Finished.

Chapter 13

Height Adjustment

Only "Ceiling" visualizers have height adjustment. These commands are used to get a correctly illuminated, focused picture. There are 2 modes:

1. Automatic Height Adjustment
2. Manual Height Adjustment

For the configuration you need an external monitor to observe the result. The example provides a button for automatic height adjustment and a button for manual height adjustment.

13.1 Automatic Height Adjustment

The automatic height adjustment is a single command. The visualizer is doing all necessary steps on its own. It is possible for constructing a GUI to get different status from the visualizer.

13.2 Manual Height Adjustment

The manual height adjustment needs user interaction. Zoom focus and light focus are adjusted manually. This sometimes gives better results than the automatic mode. Various buttons send commands to move the zoom focus and the light focus. The provided example GUI guides the user through the adjustment.

Chapter 14

Any Command

The "Any Command" form makes it convenient to try out different commands. The commands are entered into the textfield as they are listed in the command_list.pdf document. The input will be converted from ASCII to binary and sent to the visualizer. Spaces are allowed to group bytes. The binary result from the visualizer will be converted to ASCII and shown in the text area.

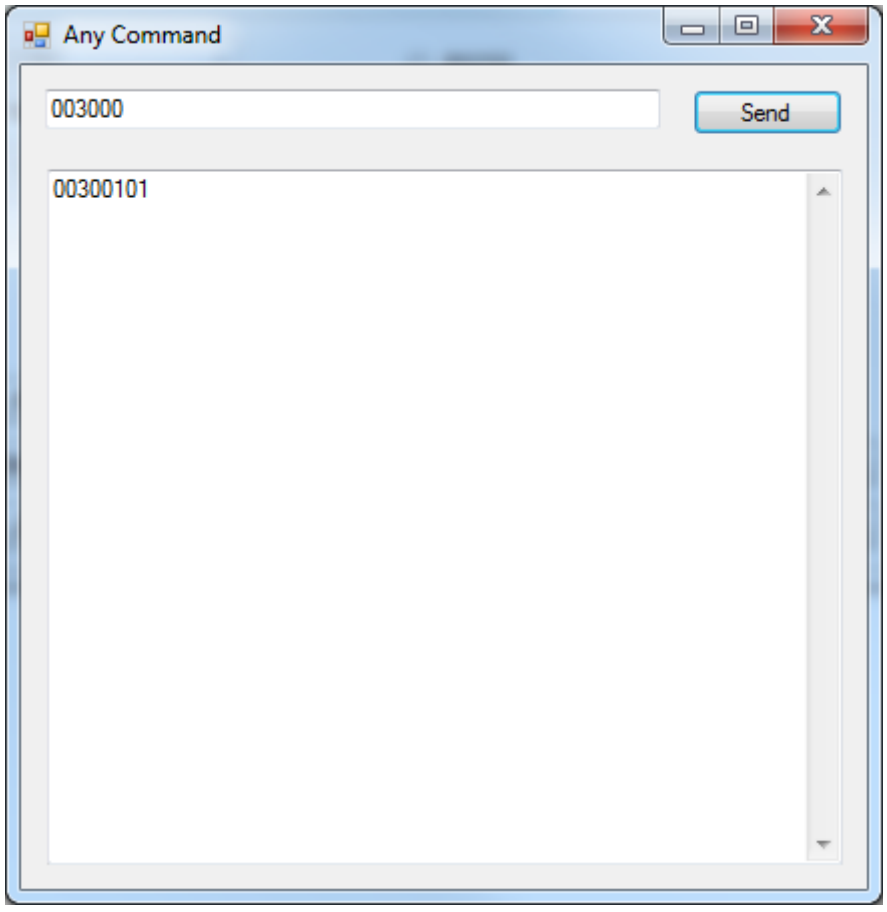

Figure 14.1: Form any command

Chapter 15

Device Discovery Protocol

In a network, often more than one visualizer is connected. It's useful to be able to search all devices. This form is opened when you click the button "Search Devices" in the mainform. The new form includes a button for scan and a button to return to the mainform. The list view shows the received devices. Note: Commands in this section do not use the Wolfprot package format, because it is necessary to send the search command as a broadcast message. There are 2 versions of the devices scan

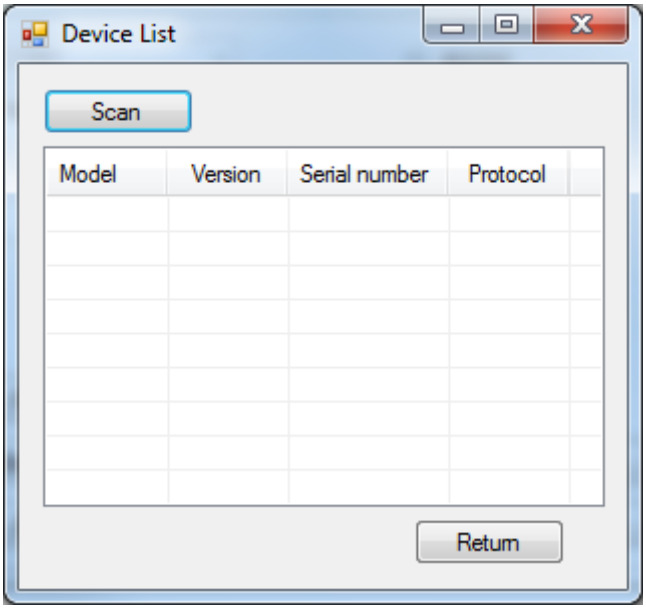

Figure 15.1: Form scan devices

protocol. V1 sends out "req_vz_list" on port 50913 and receives the reply described in the document "command_list.pdf". V2 sends out "2_req_vz_list" and receives more information than V1.

Listing showing the algorithm for scanning devices with device discovery protocol V1:

```
1       public string[] ScanDevices()
2       {
3         // Define the default port
4         int defaultPort = 50913;
5         // Create the string to send
6         string strRequest = "req_vz_list";
7
8         // Get the host name
9         string hostName = Dns.GetHostName();
10
11        // Get the current ip address
12        IPAddress[] localAddressArray = Dns.GetHostAddresses(hostName);
13
14        // localAddress array contains the IP of IPV6 and IPV4
15        // the desired protocol has to be choosen
```

```
16        IPAddress localAddress = IPAddress.Any;
17        bool ipv4 = false;
18        int x = 0;
19        while (!ipv4)
20        {
21          // Split address into a string array
22          // the string array is longer than 1 byte, if the address is the wanted
              one
23          // split with the char "." for IPv4 and ":" for IPv6)
24          string[] stringlocalAdress = localAddressArray[x].ToString().Split('.');
25          if (stringlocalAdress.Length > 1)
26          {
27            localAddress = localAddressArray[x];
28            ipv4 = true;
29          }
30          x++;
31        }
32
33        // Create a new socket
34        Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
35                                   ProtocolType.Udp);
36
37        // Bind the socket to default IP address and port.
38        socket.Bind(new IPEndPoint(localAddress, defaultPort));
39
40        // Set the receive timeout
41        socket.SetSocketOption(SocketOptionLevel.Socket,
              SocketOptionName.ReceiveTimeout, 5);
42
43        // Set socketoption for broadcast
44        socket.SetSocketOption(SocketOptionLevel.Socket,
              SocketOptionName.Broadcast, 1);
45
46        // Create a broadcast
47        IPAddress ipaddr = IPAddress.Broadcast;
48
49        // Send the broadcast into network
50        socket.SendTo(ASCIIEncoding.ASCII.GetBytes(strRequest), new
              IPEndPoint(ipaddr, defaultPort));
51
52        Byte[] buf = new Byte[1000];
53
54        // Get all devices
55        string[] receivedBuf = new string[100];
56        // A counter for the value of devices
57        int count = 0;
58        bool continueReceive = true;
59        do
60        {
61          // Try to get a device, goes to catch if got none
62          try
63          {
64            // Receive a device
65            socket.Receive(buf);
66
67            // enable to check if the reply packet is ok
68            Byte[] byteBackup = new Byte[12];
69            for (int i = 0; i < byteBackup.Length; i++)
70            {
71              byteBackup[i] = buf[i];
72            }
```

```
73            string stringBackup = Encoding.ASCII.GetString(byteBackup);
74            // Safetyquery to be sure that the a visualizer
75            // answers to the broadcast
76            if (stringBackup == strRequest && buf.Length >= byteBackup.Length)
77            {
78              // Convert the byte array into a string
79              receivedBuf[count] = System.Text.Encoding.ASCII.GetString(buf);
80              // Clear: the byte array buf
81              for (int i = 0; i < buf.Length; i++)
82              {
83                buf[i] = 0;
84              }
85
86              // Increment integer for the next device, a max number of devices of
                    100 is assumed
87              count++;
88              if (count >= receivedBuf.Length)
89              {
90                continueReceive = false;
91                MessageBox.Show("Reached maximal count of devices", "Attention",
92                                MessageBoxButtons.OK);
93              }
94            }
95          }
96          catch
97          { continueReceive = false; }
98        }
99        while (continueReceive);
100
101       string[] receivedDevices = new string[count];
102       for (int j = 0; j < receivedDevices.Length; j++)
103       {
104         string[] splittedRecvBuf = receivedBuf[j].Split('?');
105         receivedDevices[j] = splittedRecvBuf[1];
106       }
107
108       // Close the socket
109       socket.Close();
110       return receivedDevices;
111     }
```

Chapter 16

Contact

Manufacturer / Worldwide Distribution

WolfVision GmbH
Oberes Ried 14
A-6833 Klaus / AUSTRIA
Tel: ++43-5523-52250
Fax: ++43-5523-52249
E-Mail: wolfvision@wolfvision.com
Internet Homepage: www.wolfvision.com
E-Mail to Technical Support: support@wolfvision.com

US Distribution East
WolfVision Inc.
2055 Sugarloaf Circle, Suite 125
Duluth, GA 30097 / USA
Tel: 770 931-6802 or 877-873-WOLF 9653
Fax: 770 931-6906
E-Mail: support@wolfvision.us

US Distribution West
WolfVision Inc.
1601 Bayshore Highway, Suite 168
Burlingame, CA 94010 / USA
Tel: 650 648-0002 or 800 356-WOLF 9653
Fax: 650 648-0009
E-Mail: support@wolfvision.us

Asian Distribution
WolfVision Pte Ltd.
81 Ubi Ave 4, #06-27,
UB One Singapore 408830
Tel: ++65 - 6636 1268
Fax: ++65 - 6636 1269
E-mail: info@wolfvisionasia.com

Middle East Distribution
WolfVision Middle East
Suite 401, Jumeirah Terrace, Jumeirah 1
Dubai, United Arab Emirates
Tel: +971 04 354 2233
Fax: +971 04 354 2244
E-Mail: middle.east@wolfvision.net

Canadian Distribution
WolfVision Canada Inc.
7-5380 Canotek Road
Ottawa, ON K1J 1H7 Canada
Tel: 613 741-9898 or 877 513-2002
Fax: 613 741-3747
E-Mail: wolfvision.canada@wolfvision.com

Japan Distribution
WolfVision Co Ltd.
Advance Bldg. 2F, 8-1-16 Nishi Shinjuku, Shinjuku
Tokyo 160-0023, Japan
Tel: 81 3 3360 3231
Fax: 81 3 3360 3236
E-mail: wolfvision.japan@wolfvision.com

United Kingdom Distribution
WolfVision UK Limited
Trident One, Styal Road
Manchester M22 5XB, United Kingdom
Tel: ++44-161-435-6081
Fax: ++44-161-435-6100
E-Mail: wolfvision.uk@wolfvision.com