User Manual

Register based ASCII Protocol

MAN050 12/2013 V1.2

Contents

1 Preface

1.1 About Photonfocus

The Swiss company Photonfocus is one of the leading specialists in the development of CMOS image sensors and corresponding industrial cameras for machine vision, security & surveillance and automotive markets.

Photonfocus is dedicated to making the latest generation of CMOS technology commercially available. Active Pixel Sensor (APS) and global shutter technologies enable high speed and high dynamic range (120 dB) applications, while avoiding disadvantages like image lag, blooming and smear.

Photonfocus has proven that the image quality of modern CMOS sensors is now appropriate for demanding applications. Photonfocus' product range is complemented by custom design solutions in the area of camera electronics and CMOS image sensors.

Photonfocus is ISO 9001 certified. All products are produced with the latest techniques in order to ensure the highest degree of quality.

1.2 Contact

Photonfocus AG, Bahnhofplatz 10, CH-8853 Lachen SZ, Switzerland

Sales	Phone: +41 55 451 07 45	Email: sales@photonfocus.com
Support	Phone: +41 55 451 01 37	Email: support@photonfocus.com

Table 1.1: Photonfocus Contact

1.3 Sales Offices

Photonfocus products are available through an extensive international distribution network and through our key account managers. Details of the distributor nearest you and contacts to our key account managers can be found at www.photonfocus.com.

1.4 Further information

Photonfocus reserves the right to make changes to its products and documentation without notice. Photonfocus products are neither intended nor certified for use in life support systems or in other critical systems. The use of Photonfocus products in such applications is prohibited.

Photonfocus is a trademark and LinLog® is a registered trademark of Photonfocus AG. CameraLink® and GigE Vision® are a registered mark of the Automated Imaging Association. Product and company names mentioned herein are trademarks or trade names of their respective companies.

Photonfocus can not be held responsible for any technical or typographical errors.

1.5 Legend

In this documentation the reader's attention is drawn to the following icons:

Important note

Alerts and additional information

Attention, critical warning

Notification, user guide

2

Register based ASCII Protocol

2.1 Overview

The register based ASCII protocol can be used to control the camera directly from a DSP, uC or FPGA. The communication runs over a simple UART (RS-232 link) with RX/TX signals without handshake signals.

Please refer to the camera (Board Level or OEM) manual for the description of the logic level voltage of the UART link.

The RS-232 link is configured as 8N1 (8 data byte, no parity, one stop byte), default baud rate is 9600.

2.2 Set camera to ASCII Protocol

The camera can be switched with one simple byte between the PFRemote and the ASCII protocol. The camera will send a ACK (see Table 2.1) if the protocol switch was successful. After successful switching to ASCII protocol wait 200ms before sending other commands. After a power cycles (POR - Power On Reset) the camera starts always in PFRemote protocol mode with default baud rate 9600baud.

send byte	ACK of the camera	new protocol mode
0x01	0x45	PFRemote protocol
0x02	0x02	ASCII protocol
0x02	none	PFRemote protocol (camera does not support ASCII protocol)

Table 2.1: Protocol switching

Example:

Send Command: 0x02

Receive answer: 0x02

2.3 Set Property value

To set a property with a new value 10 bytes have to send. The camera will answer with one byte. The 10 bytes contains 1 start, 4 addr, 4 data and 1 stop byte. The camera answers with one byte.

The MSB of address and data will be send first.

Send Command: <StartWrite> <Addr_3 (MSB)> <Addr_2> <Addr_1> <Addr_0 (LSB)> <Data_3 (MSB)> <Data_2> <Data_1> <Data_0 (LSB)> <EndWrite>

Receive answer: <ACKWrite>

Write command byte	Hex value (ASCII value)
StartWrite	0x57 ('W')
EndWrite	0x77 ('w')
ACKWrite	0x77 ('w')

Table 2.2: Send commands

Example: Addr = 0x0e, Value = 0x0400:

Send Command: 0x57 0x00 0x00 0x00 0x0e 0x00 0x00 0x04 0x00 0x77.

Receive answer: 0x77.

2.4 Get Property value

To get a property value 6 bytes have to send. The camera will answer 6 bytes too. The 6 bytes contains: 1 start, 4 addr and 1 stop byte. The camera answers with: 1 start, 4 data and 1 stop byte.

The MSB of address and data will be send first.

Send Command: <StartRead> <Addr_3 (MSB)> <Addr_2> <Addr_1> <Addr_0 (LSB)> <EndRead>

Receive answer: <ACKRead> <Data_3 (MSB)> <Data_2> <Data_1> <Data_0 (LSB)> <ACKRead>

Read command byte	Hex value (ASCII value)
StartRead	0x52 ('R')
EndRead	0x72 ('r')
ACKRead	0x72 ('r')

Table 2.3: Receive commands

Example: Addr = 0x0e.

Send Command: 0x52 0x00 0x00 0x00 0x0e 0x72.

Receive answer: 0x72 0x00 0x00 0x04 0x00 0x72. -> Value: 0x00000400 = 0x400 = 1024.

2.5 Address of Property

All camera properties have a register address. This address is fix over all camera types from Photonfocus. Please check the `pfASCII_PropertyList.h` file and Section 2.9.

Addr	Property name	Type
0x00000001	ExposureTime	PF_FLOAT
0x0000000c	Window.X	PF_INT
0x0000000e	Window.W	PF_INT
...	...	...

Table 2.4: Details of the pfASCII_PropertyList.h file

2.6 Value of Property (data bytes)

The 4 bytes data can be interpreted as Integer or IEEE 754 float. Properties with types PF_INT, PF_BOOL and PF_MODE interpret the data bytes as Integer. PF_FLOAT as IEEE 754.

Data bytes	PF_INT, PF_BOOL and PF_MODE	PF_FLOAT
0x00000000	0	0.0
0x0000000c	12	1.7E-44
0x3F800000	1065353216	1.0
0x40000000	1073741824	2.0
0x414B26E9	1095444201	12.697

Table 2.5: Interpretation of data bytes

2.6.1 Integer

For interger properties (PF_INT, PF_BOOL and PF_MODE) the 4byte value needs to interpret as 32bit signed integer.

2.6.2 IEEE 754 Single Precision

PF_FLOAT properties need to interpret the 4 byte data as IEEE 754. IEEE floating point numbers have three basic components: the sign, the exponent, and the mantissa. Table 2.6 and Fig. 2.1 shows the layout for single (32-bit) precision floating-point values. The number of bits for each field are shown (bit ranges are in square brackets).

	Sign	**Exponent**	**Mantissa**
Single Precision	1 [31]	8 [30-23]	23 [22-00]

Table 2.6: Storage Layout

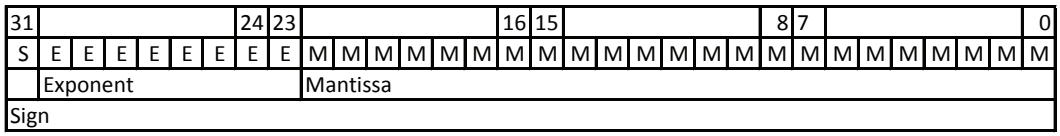

Figure 2.1: Storage Layout (bits)

IEEE 754 float: $\text{value} = \text{Sign} * 2^{\text{Exponent}} * \text{Mantissa}$

For more information, please check the IEEE 754 standard.

2.7 Set Property example

Set property `Window.W` to 1024. (Address of `Window.W` is 0x0e)

```
unsigned long propertyAddr;
long propertyValue;
unsigned char buf[16];

//set camera to ASCII mode
buf[0] = 0x02;
WriteRS232(buf, 1);
buf[0] = 0x00;
ReadRS232(buf, 1);
if(buf[0] != 0x02){
    //camera does not support ASCII protocol
    return -1;
}

//wait 200ms
Sleep(200);


//write Window.W to 1024
propertyAddr = 0x0e;
propertyValue = 1024;

buf[0] = 'W';
buf[1] = propAddr >> 24;
buf[2] = propAddr >> 16;
buf[3] = propAddr >> 8;
buf[4] = propAddr;
buf[5] = propertyValue >> 24;
buf[6] = propertyValue >> 16;
buf[7] = propertyValue >> 8;
buf[8] = propertyValue;
buf[9] = 'w';
error = WriteRS232(buf, 10);

//read ACK, must be 0x77 = 'w' (1 byte)
ReadRS232(buf, 1);
if(buf[0] != 0x77){
    //camera ACK error
    return -1;
}
```

2.8 Get Property example

Get value of property `ExposureTime`. (Address of `ExposureTime` is 0x01)

```
unsigned long propertyAddr;
float propertyValue;
unsigned char buf[16];
unsigned char data[4];

//set camera to ASCII mode
buf[0] = 0x02;
WriteRS232(buf, 1);
buf[0] = 0x00;
ReadRS232(buf, 1);
if(buf[0] != 0x02){
    //camera does not support ASCII protocol
    return -1;
}

//wait 200ms
Sleep(200);


//read ExposureTime
propertyAddr = 0x01;

buf[0] = 'R';
buf[1] = propAddr >> 24;
buf[2] = propAddr >> 16;
buf[3] = propAddr >> 8;
buf[4] = propAddr;
buf[5] = 'r';
error = WriteRS232(buf, 6);

//read value (6 bytes)
ReadRS232(buf, 6);

//check if first byte is 'r'
if(buf[0] != 0x72){
    //camera ACK error
    return -1;
}

//little endian <-> big endian
data[0] = buf[4];
data[1] = buf[3];
data[2] = buf[2];
data[3] = buf[1];

propertyValue = ((float*)data)[0];
```

2.9 pfASCII_PropertyList.h

All camera properties have a register address. These addresses are the same for all camera types from Photonfocus. The `pfASCII_PropertyList.h` shows the address of the properties over all cameras, not all properties are supported from a camera type. The header file `pfASCII_PropertyList.h` could be found in the `PFRemote/SDK/include` folder

```
#define INVALID_ADDR 0xffffffff

typedef enum{
    PF_INT,         //32bit signed integer
    PF_FLOAT,       //IEEE 754 32bit float, single precision
    PF_BOOL,        //boolean value (1: true, 0: false), handled as PF_INT
    PF_MODE,        //mode value, handled as PF_INT
    PF_COMMAND,     //command, handled as PF_INT
    PF_STRING,      //not yet supported
    PF_BUFFER,      //not yet supported
}pfASCII_PropertyType;

char *pfASCII_PropertyTypes[] = {
    (char*)"Integer",
    (char*)"Float",
    (char*)"Boolean",
    (char*)"Mode",
    (char*)"Command",
    (char*)"String",
    (char*)"Buffer",
};

typedef struct{
    unsigned long         addr;
    char                  *name;
    pfASCII_PropertyType  type;
}pfASCII_PropertyList_t;

pfASCII_PropertyList_t pfASCII_PropertyList[] = {
/*    0, addr=0001 */  { 0x00000001, "ExposureTime", PF_FLOAT },
/*    1, addr=0003 */  { 0x00000003, "FrameTime", PF_FLOAT },
/*    2, addr=0012 */  { 0x0000000c, "Window.X", PF_INT },
/*    3, addr=0013 */  { 0x0000000d, "Window.Y", PF_INT },
/*    4, addr=0014 */  { 0x0000000e, "Window.W", PF_INT },
/*    5, addr=0015 */  { 0x0000000f, "Window.H", PF_INT },
/*    6, addr=0016 */  { 0x00000010, "Window.Max", PF_COMMAND },
...
...                    { INVALID_ADDR , NULL, 0}
};
```

2.10 Camera properties

A camera type does not support all properties of the `pfASCII_PropertyList.h`. Each camera type has his own property list file in the following folder: `PFRemote/doc/CameraProperties`. These files show which properties are supported at the given camera type.

A

Revision History

Revision	Date	Changes
1.0	October 2011	First release
1.1	February 2013	Add chapter "Camera properties"
1.2	December 2013	Hex value of StartWrite command corrected