ZioLink
Communication Protocol Manual

Version 1.2 - 20 October 2023

Avenir Photonics GmbH & Co. KG
Franz-Mayer-Str. 1
93053 Regensburg
Germany

Phone: +49 - 941 - 46 29 72 80
E-Mail: info@avenirphotonics.com
www.avenirphotonics.com

Contents

Introduction

This manual describes the communication protocol ZioLink, which is used to control Aris spectrometers from a host computer. The manual is part of the Avenir Software Development Kit (SDK), which also contains sample code, tools and device drivers.

The ZioLink protocol was designed to be easy to understand yet powerful and flexible to be used in many different setup. If you would like to control an Aris spectrometer from your own software, you have the following options:

- If you are using Windows and the .NET framework, you can take the ZioLinkSpectrometer class (as implemented in the SimpleSpectrometer sample code) together with the other classes in the same folder and use it in your own code.
- You can also directly communicate with the device using this protocol. This should work on any platform in any programming language.

If you are not using Windows and the .NET framework, the first step would be to find out how to send and receive simple messages from and to a USB device. Once you've established the communication to the device, sending commands and receiving replies should be simple and straightforward to implement.

When reading this manual, we recommend you try out the protocol using the tool ProtocolDroid included in the SDK. This should give you a good understanding of the basic concepts. Afterwards, please check the demo codes BasicSpectrometer and SimpleSpectromter to see how to use the procotol to control spectrometers.

For further questions, comments and requests, please do not hesitate to contact our technical support.

The ZioLink protocol

The ZioLink protocol defines communication between a host computer and a device. It is half-duplex with no out-of-band signalling. It does not determine the length of a single transmission or the data integrity, because it assumes that the interface (like USB) has already that. The byte-order is little-endian, as used by almost all computers today.

The ZioLink protocol is quite similar to the earlier NioLink protocol used to control other devices.

All communication is always intiated by the host sending a *request message* to the device consisting of:

1. a 32-bit *command code*
2. additional data (payload) if required by the command code

The device then answers by sending a *response message* consisting of:

1. a 32-bit *return code* (zero for OK or an error code)
2. additional data (payload) if required by the command code

Command Codes

Each command code is of the form 0x0000XYZZ, where:

- X is the *Command Type*
- Y is the *Command Action*
- ZZ is the *Command Index*

Each request from the host can be either a command to the device or a request to send or receive data. This corresponds to the Command Type X, which can be one of the following:

- 0 = *Command* (sometimes followed by a parameter)
- 1 = *Parameter* (a value than can can be read and written)
- 2 = *Device property* (a value that was set during manufacturing and can now only be read)
- 3 = *Measured value* (a read-only value that frequently changes)
- 4 = *Bulk data* (a larger data block than can be read and possibly written)

The Command Action Y then specifies what to do:

- 0 = Get (reads the value from the device or executes the command)
- 1 = Set (writes the value to the device)
- 2 = Minimum (returns the minimum value)
- 3 = Maximum (returns the maximum value)
- 4 = Default (returns the default value)
- 8 = Data type (returns the data type, see below)
- 9 = Name (returns the name of this value as a string)
- A = Unit (returns the unit of this value as a string)
- F = Length (returns the number of available indices for this command type)

Finally the Command Index ZZ specifies the individual command or value.

Examples

The command code 0x0000130A returns the maximum value for the parameter AutoExposureTime:

- 1 = Parameter
- 3 = Maximum

- 0A = AutoExposureTime

The command code 0x00004103 writes User Data to the device:

- 4 = Bulk data
- 1 = Set
- 03 = UserData

Data Types

Parameters, Device Properties and Measured Values can be one of the following data types:

- 0 = Signed 32-bit integer
- 1 = Unsigned 32-bit integer
- 2 = Single-precision float
- 3 = 32-bit version number (0xAABBCCDD means version AA.BB.CC.DD)
- 4 = String (null-terminated ASCII encoded)
- 5 = Boolean
- 6 = Bit field

All data types except strings are 32-bit values. The data type of an individual value can be retrieved with the Command Action 8 (= Data Type).

Return Codes

Each request from the host is answered by the device with a return code of the form 0x0000YYZZ:

- YY is zero for OK or an error code
- ZZ may contain a sub-code for the error specifying further details

The most common return codes are:

- 0x0000 = OK (operation was successfull)
- 0x0100 = Unknown command code
- 0x0200 = Invalid argument
- 0x0300 = Missing argument
- 0x0400 = Invalid operation (not possible due to current state of device)
- 0x0500 = Not supported (not available with this device)
- 0x0600 = Invalid passcode
- 0x0700 = Communication error
- 0x0800 = Internal error
- 0x0A00 = Cannot read internal memory

The USB Interface

When using the ZioLink protocol over a USB interface, each message corresponds to a single USB transfer (consisting of one or more USB packets). Data is sent using the transfer type Bulk via the default endpoints 0x81 (in) and 0x01 (out).

The USB Vendor ID for all devices by Avenir Photonics is 0x0x354F. Product IDs for spectrometers by Avenir Photonics are of the form 0x01XX.

The Aris spectrometer communicates with „Full speed" according to the USB 2.0 specifications.

In Windows we recommend you use the WinUSB device driver supplied by Microsoft. A suitable .inf file for installing the driver is included in the SDK.

The spectrometer can also work as a WCID ("Windows Compatible ID") device. This means that the device provides additional information to Windows that (in theory) should eliminate the need for an .inf file. However, we found that if driver installation is interrupted, the installation may remain incomplete. For example, this may happen if the user attaches the device, but the removes it quickly before the driver installation is finished. In this case, it may remain unfinished even if you connect the device again. Then you would have to delete the corresponding entries in the Windows Registry in order to recover. Therefore (unfortunately) we have to recommend that you continue to use the "traditional" driver installation with .inf files.

In Unix-based systems (including Mac OS) you can communicate with the device over USB using the libusb library, which is included in all major Unix systems. With libusb you don't need a special driver installation file. Just connect the device and address it using the Vendor and Product ID.

For all other platforms please see the documentation of the hardware interface layer in the corresponding operating system.

Controlling Spectrometers

The following chapter describes the command codes to control a spectrometer. Don't be intimidated by the size of these lists. The Aris spectrometer has lots of features built into its firmware and electronics, but the procedure for a regular measurement is rather simple.

The procedure for a simple spectrum measurement would be:

- **Send a Reset command** to the device (0x0000)
- **Set the exposure time, averaging and auto exposure parameters** as desired. These values are generally stored persistently in the device, so you only need to set them if the conditions have changes since the last measurement.
- **Send a StartExposure command** (0x0004, followed by the number of exposures)
- **Read the device status** (0x3000) until it becomes "Idle" again
- **Read the spectrum** with GetSpectrum (0x4000)
- When finished: **send the StandBy command** (0x0001) if you'd like to keep any changes of parameters

To obtain the wavelengths for each spectrum value, please use the GetWavelengths command (0x4001). Alternatively you can also read the wavelengths coefficients and calculate the wavlengths with a 3rd order polynomial.

You generally don't need to apply offset or linearity corrections to the measured spectra. This is already done inside the device. Dark and reference spectra or a sensitivity calibration are also applied within the device if enabled by the corresponding parameters.

Please see the BasicSpectrometer demo code for a actual implementation of this procedure. For an example of a more sophisticated software please check the SimpleSpectrometer demo.

Specific Command Indices

These are the command indices for the Aris spectrometer. Combine them with a command type and action (see previous chapter) to form a command code.

To find out more about the values listed below, please open the device connection in the application software "Inspective" and choose "Device Settings" from the Device menu. This displays a list of these values including data type, unit of measurement, and minimum, maximum and default values.

Commands

Cmd_Reset = 0x00,
Cmd_StandBy = 0x01,
Cmd_DeviceReset = 0x02,
Cmd_ParameterReset = 0x03,
Cmd_StartExposure = 0x04,
Cmd_CancelExposure = 0x05,
Cmd_RestoreFactoryDarkSpectrum = 0x06,
Cmd_RestoreFactorySensitivityCal = 0x07,

Parameters	Notes
Param_ExposureTime = 0x00,	in µs
Param_Averaging = 0x01,	number of single spectra to be averaged
Param_AuxVoltageEnable = 0x02,	supply voltage output on the aux connector

Param_AuxConfiguration = 0x03,	
Param_TriggerInConfiguration = 0x04,	(see description below)
Param_TriggerInDelay = 0x05,	in µs
Param_TriggerInEnable = 0x06,	for external triggering
Param_BaudRate = 0x07,	for the UART interface
Param_TurnOffLEDs = 0x08,	1 = turns off indicator LEDs on device
Param_AutoExposureEnable = 0x09,	
Param_AutoExposureTime = 0x0A,	
Param_TriggerOutConfiguration = 0x0B,	
Param_TriggerOutDelay = 0x0C,	
Param_SensorGain = 0x0D,	0 = low, 1 = high gain (if supported by sensor)
Param_SubtractDark = 0x0E,	
Param_IntensityUnit = 0x0F,	(see description below)
Param_WavelengthUnit = 0x10,	so far only 0 = "nm" supported
Param_AnalogOutput = 0x11,	sets the voltage level on the Analog Output pin
Param_PrescanAfterTimeChange = 0x12,	
Param_SaveChangesPersistently = 0x13	(see description below)
Param_DataFormat = 0x14	(see description below)

Device Properties

DevProp_DeviceID = 0x00,
DevProp_SerialNumber = 0x01,
DevProp_Manufacturer = 0x02,
DevProp_ModelName = 0x03,
DevProp_HardwareVersion = 0x04,
DevProp_SoftwareVersion = 0x05,
DevProp_SaturationLevel = 0x06,
DevProp_PixelCount = 0x07,
DevProp_DataCount = 0x08,
DevProp_FirstOffsetPixel = 0x09,
DevProp_NumOffsetPixels = 0x0A,
DevProp_FirstDarkPixel = 0x0B,
DevProp_NumDarkPixels = 0x0C,
DevProp_FirstSpectrumPixel = 0x0D,
DevProp_PixelsPerBin = 0x0E,
DevProp_MirrorSpectrum = 0x0F,
DevProp_SensorType = 0x10,
DevProp_OpticalConfiguration = 0x11,
DevProp_ElectronicsType = 0x12,
DevProp_CalibrDataNumSpectra = 0x13,
DevProp_UserDataMaxSize = 0x14,
DevProp_SamplingPoint = 0x15,
DevProp_ReadoutSettings = 0x16,
DevProp_ReadoutNoise = 0x17,
DevProp_BadPixels0 = 0x18,

DevProp_BadPixels1 = 0x19,
DevProp_BadPixels2 = 0x1A,
DevProp_BadPixels3 = 0x1B,
DevProp_WavelengthCoeff0 = 0x1C,
DevProp_WavelengthCoeff1 = 0x1D,
DevProp_WavelengthCoeff2 = 0x1E,
DevProp_WavelengthCoeff3 = 0x1F,
DevProp_WavelengthTemp = 0x20,
DevProp_WavelengthTempCoeff = 0x21,
DevProp_OffsetLowGain = 0x22,
DevProp_OffsetHighGain = 0x23,
DevProp_OffsetLowGainTempCoeff = 0x24,
DevProp_OffsetHighGainTempCoeff = 0x25,
DevProp_OffsetTemp = 0x26,
DevProp_NonlinNumCoeff = 0x27,
DevProp_NonlinCoeff0 = 0x28,
DevProp_NonlinCoeff1 = 0x29,
DevProp_NonlinCoeff2 = 0x2A,
DevProp_NonlinCoeff3 = 0x2B,
DevProp_NonlinCoeff4 = 0x2C,
DevProp_NonlinCoeff5 = 0x2D,
DevProp_NonlinCoeff6 = 0x2E,
DevProp_NonlinCoeff7 = 0x2F,
DevProp_DarkTempCoeff0 = 0x30,
DevProp_DarkTempCoeff1 = 0x31,
DevProp_CommunicationProtocol = 0x32

Measured Values

Notes

MeasVal_Status = 0x00, (see description below)
MeasVal_SensorTemperature = 0x01, internal device temperature in °C
MeasVal_IOPortStatus = 0x02,
MeasVal_SysTick = 0x03, elapsed ms since device was powered on
MeasVal_RemainingExposures = 0x04,
MeasVal_BufferCount = 0x05, current number of spectra in device buffer
MeasVal_CpuTemperature = 0x06,
MeasVal_BufferSize = 0x07, size of the device spectrum buffer
MeasVal_SupplyVoltage = 0x08,
MeasVal_AuxVoltageOK = 0x09,
MeasVal_AnalogInput = 0x0A, reads the voltage on the Analog Input pin
MeasVal_Time = 0x0B,
MeasVal_Date = 0x0C,
MeasVal_CalibrSpectraAvailable = 0x0D,
MeasVal_SensitivityCalUnit = 0x0E,

Bulk Data

BulkData_Spectrum = 0x00, (see description below)

```
BulkData_Wavelengths = 0x01,        the wavelengths for each pixel as float
BulkData_CalibrationData = 0x02,
BulkData_UserData = 0x03,
BulkData_AuxInterface = 0x04,
BulkData_DateTime = 0x05,
BulkData_OffsetSpectrum = 0x06,
BulkData_FactoryDark = 0x07,
BulkData_FactorySensCal = 0x08,
BulkData_DarkSpectrum = 0x09,
BulkData_SensitivityCal = 0x0A,
BulkData_RefSpectrum = 0x0B,
BulkData_ReflecStandard = 0x0C
```

Data Types, Structures and Enumerations

Some of these values and command codes require special data structures that are described below.

Status values

If you send a GetStatus request (0x3000) the device sends a 32-bit response value (after the return code). Byte 0 of this response is one of the following values:

```
StatVal_Idle = 0x00,
StatVal_WaitingForTrigger = 0x01,
StatVal_TakingSpectrum = 0x02,
StatVal_PoweredOff = 0x08,
StatVal_SleepMode = 0x09,
StatVal_InterfaceNotConnected = 0x0A
```

Bytes 1 and 2 of the response value contain the number of available spectra that have been measured, processed and are now ready to read them out from the internal buffer.

Spectrum

If you read a spectrum from the device with the GetSpectrum request (0x4000), the devices sends a spectrum header followed by the actual spectrum values.

The spectrum header contains meta data about the spectrum as follows:

```
typedef struct
{
    unsigned int ExposureTime;     //in µs
    unsigned int Averaging
    unsigned int Time;             //elapsed 0.1 ms since midnight (refers to start of exposure)
    unsigned int Date;             //elapsed days since 1 January 1970
    float LoadLevel;               //0.0 = no signal, 1.0 = saturation level
    float SensorTemperature;       //in °C
    uint16_t PixelCount;
    uint8_t DataFormat;            //see description below
    uint8_t VersionNumber;         //reserved for future use, now 0x00
```

```
    uint16_t ExposureSettings;      //ReadoutSettings plus flags for auto exposure, gain and
                                    //buffer overflow
    uint16_t AppliedProcessing;     //nibble 0: IntensityUnit, nibble 1: DarkSpectrum,
                                    //nibble 2: reserved, nibble 3: processing error flags
    char Name[32];                  //up to 3 null-terminatedstrings:
                                    //Spectrum name, intensity unit, sample
} SpectrumHeader; //size: 64 bytes
```

The spectrum header is followed by the actual spectrum. By default this is an array of 32-bit single-precision floating point numbers. The length of the spectrum is specified by the PixelCount value, which you can obtain with the command code 0x2007 (Get Device Property: PixelCount) or from the spectrum header (see above).

The spectrum data structure (consisting of spectrum header and float array) is also used when transmitting calibration spectra from and to the device. These include the dark spectrum, the reference spectrum and the sensitivity calibration.

TriggerInConfiguration

This 32-bit value is a byte-wise combination of the following settings:

- Byte 0: Trigger Mode
  0 = Free-Running (trigger on end of exposure)
  1 = Free-Running (trigger of start of exposure)
  2 = Hardware Trigger
- Byte 1: External Trigger Edge
  0 = Rising edge
  1 = falling edge
- Byte 2 and 3: reserved for future use

IntensityUnit

This parameter selects the unit of the y-axis of the spectrum. It can be used to choose between different measurement modes and corresponds to the "Intensity" box on the user interface of the application software "Inspective". It can have one of the following values:

```
IntensityUnit_Counts = 0x0,
IntensityUnit_CountsPerMs = 0x1,
IntensityUnit_Transmittance = 0x2,
IntensityUnit_Reflectance = 0x3,
IntensityUnit_Absorbance = 0x4,
IntensityUnit_Calibrated = 0x5,
IntensityUnit_Evaluated = 0x6
```

DataFormat

By default all spectra are returned as 32-bit floating point numbers. To speed up data transfer, shorter formats can be choosen:

```
DataFormat_32bitFloat = 0,              //32-bit float values (default)
DataFormat_16bitInteger = 1,            //offset and slope as two float values,
                                        //then spectrum values as 16-bit integer values
DataFormat_16bitInteger2Binning = 2     //similar to DataFormat_16bitInteger, but odd and
                                        //even pixels are combined into one pixel
```

For an example of how these data formats are to be used, please see the FastReadout demo code.

SaveChangesPersistently

The device includes a flash memory to store all parameters persistently. This means that if you change some parameters and then close the software and disconnect the device, all parameters will be saved into flash memory. The next time you use the device, all parameters will be set to the most recent values.

The parameters are only saved to the flash memory when you send the "Cmd_StandBy" or "Cmd_Reset" command. Therefore the "Cmd_StandBy" command should always be sent by the host software when the device connection or the software is closed.

If you do not want to save any parameter changes persistently in the device flash memory, set the "SaveChangesPersistently" parameter to 0 (= false). Then whenever the device is powered on, the parameters are always reset to the same values. This may be helpful if you need to ensure that the device always starts up with the same parameter settings.

Special Procedures

External Triggering

Before using external triggering from your own software, please make yourself familiar with the terminology and operation for triggering. You can find more information about this in the user manual and try it out using the application software "Inspective".

To configure external triggering from your own software:

1. Set the TriggerInConfiguration parameter (command code 0x1104) to the desired value for trigger mode and trigger edge (see the description of these values abve). In most cases this would be the value 0x0002 for "Hardware Trigger" and "Rising Edge".
2. Set the TriggerInDelay(command code 0x1105) as desired. To start the exposure immediately after the trigger event, set this parameter to 0.
3. Set the TriggerInEnable (command code 0x1106) to 1 (= true).
4. Start the exposure (as usual) by sending the StartExposure command (0x0004).
5. The spectrometer then waits for a trigger event to occur on the "Trigger In" pin before starting the exposure.