ENG-0001
USB Interface Specification

For Wasatch Photonics CCD and CMOS Spectrometers

Revision 1.8

October 18, 2019

Contents

Revision Log

Doc #	Date	Author	Description	Rev.
ENG-1	9/25/14	S. Trani	Initial Release	1.0
	8/11/15	J. Dreitzler	Add second tier commands, model configuration and feature identification	1.1
	10/6/15	J. Dreitzler	Corrected model configuration format Multiple changes to align commands for FX2 and ARM-based products	1.2
	10/20/15	J. Dreitzler	Added additional commands	1.3
	1/25/17	J. Traud	Formatting, cleanup, and preparation for public release.	1.4
	10/2/17	J. Traud	Removed customer specific references	1.5
	10/8/17	J. Traud	Removed legacy and development command references no longer valid in current firmware branch	1.6
	8/24/18	M. Zieg	Update to reflect current firmware	1.7
	10/18/19	M. Zieg R. Dickerson	Added battery commands Added Get/Set CCD Start/Stop Line	1.8

1 General Description

This document describes the USB (Universal Serial Bus) API (Application Programming Interface) for all USB-based Wasatch Photonics Raman, NIR, VIS, VIS-NIR and UV-VIS spectrometers. It does not apply to spectrometers intended for the OCT market such as the Cobra series.

The USB Interface for the Wasatch spectrometer is USB 2.0 compliant. FX2-based spectrometers operate at High Speed (480Mbps), while ARM-based spectrometers currently operate at Full Speed (12Mbps).

Control commands are sent as Vendor Requests (hence the "VR_" suffix on some legacy commands), and the response is read on endpoint 0. The key exception is when reading spectra (with or without triggering), in which the response data is returned on the bulk-output endpoint 2 (and possibly endpoint 6 for larger detectors).

2 USB Connection to device

The spectrometer appears as a USB device with Vendor Identification code (VID) 0x24aa and a Product Identification code (PID) that corresponds to the product configuration (legacy) or to one of three supported values representing Feature Identification Device (FID) models. PIDs used for feature identification can be found in ENG-0034, "FID (Feature Identification Device) Model Configuration," but are primarily 0x1000 (FX2 silicon detector), 0x2000 (FX2 InGaAs detector) and 0x4000 (ARM-based spectrometer).

2.1 USB device drivers

Wasatch Photonics spectrometers communicate primarily via USB. There are many low-level USB drivers available, including the open-source libusb (Linux/MacOS) and libusb-win32 (Windows), Microsoft's WinUSB.sys, Cypress' CyUSB etc. Technically, any USB library can be used to communicate with Wasatch spectrometers by following the documentation of that driver and our USB hardware/firmware API.

However, associating a USB VID/PID with a particular device can be a tricky process (sometimes requiring convoluted procedures to digitally "sign" a driver), and it is typically easiest to use a supported vendor driver combination.

For Microsoft Windows, Wasatch recommends the libusb-win32 library (https://sourceforge.net/p/libusb-win32), which is based on libusb-0.1. For Linux and MacOS, Wasatch recommends libusb-0.1. Wasatch has no current plans to switch to libusb-1.0 (or its associated "libusb Windows"), but may do so if/when performance and required features dictate.

The open-source libusb-win32 library officially supports Windows XP, Windows Vista, Windows 7 (x86/x64) and Windows 10 (x86/x64). It has been tested internally by Wasatch on XP (x86), Vista (x64), Windows 7 (x86/x64) and Windows 10 (x86/x64).

Note that there is a distinction between "low-level" USB drivers like libusb, and "high-level application drivers" like Wasatch.PY or Wasatch.NET. Low-level USB drivers provide byte-level communication between the host PC and the hardware peripheral. Higher-level application drivers run atop the lower-level USB drivers, and automate common operations by "wrapping" complex multi-step procedures into simple function calls (hiding the marshalling/demarshalling of arguments, endian issues etc).

If you are interested in developing your spectroscopy application against our high-level drivers, please see:

https://wasatchphotonics.com/software-drivers/

2.2 Establishing connection through libusb-win32 drivers

Detailed command descriptions of the libusb-win32 library can be found here (at writing):

https://sourceforge.net/p/libusb-win32/wiki/Documentation/

Open-source examples showing how to connect to Wasatch spectrometers through libusb can be found here, and in other published Wasatch Photonics sample code:

- https://github.com/WasatchPhotonics/Wasatch.PY
- https://github.com/WasatchPhotonics/Wasatch.NET

2.2.1 C Example

A standard C routine to open and return a device handle to the spectrometer is as follows:

```
#define MY_VID 0x24aa // Wasatch Photonics
#define MY_PID 0x1000 // will depend on model
usb_dev_handle* open_dev(void) {
  usb_init();
  usb_find_busses();
  usb_find_devices();
  for (struct usb_bus *bus = usb_get_busses(); bus; bus = bus->next)
    for (struct usb_device *dev = bus->devices; dev; dev = dev->next)
          if (   dev->descriptor.idVendor  == MY_VID
              && dev->descriptor.idProduct == MY_PID)
             return usb_open(dev);
  return NULL;
}
```

Two other commands are needed to properly configure the device:

```
usb_set_configuration(dev, 1)   // sets active configuration to 1
usb_claim_interface(dev, 0)     // claims interface 0
```

where *dev* is a device handle returned by open_dev().

At this point standard control message commands described below can be sent to the device. A control message has the following format:

```
int usb_control_msg(usb_dev_handle *dev, int requesttype, int request, int value,
   int index, char *bytes, int size, int timeout);
```

where:

- *dev*: handle returned by usb_open()
- *requesttype*: 0x40 for host to device commands, 0xC0 for device to host requests
- *request*: bRequest field in the setup packet (see commands). bRequest represents the specific command or 'opcode' being sent to the spectrometer.
- *value*: value field in the setup packet. Often used to send parameters of a command.
- *index*: index field in the setup packet. Also, often used to send high-order byte information as parameter of command.
- *bytes*: up to 64-byte data packet associated with control message. Used in some set commands to represent higher order parameter bytes. Unless specified, can be sent as NULL pointer on FX2-based models, although ARM-based spectrometer commands assume it will hold an 8-byte buffer of zeros even if unused.
    - Note: Commands taking UInt40 parameters (such as used for laser modulation) will typically send the *most*-significant of the 5 parameter bytes as the first (data[0]) byte of the data payload, while sending the least-significant bytes in the *value* field, and the middle-significant bytes in the *index* field.
- *size*: size of bytes data packet buffer. Can be zero if a NULL data packet reference is used.
- *timeout*: time before timing out if an error occurs in milliseconds.

For instance, to **set** the integration time to 100ms (see SET_INTEGRATION_TIME below), one would execute:

```
char *buf = {0, 0, 0, 0, 0, 0, 0, 0};
int ret = usb_control_msg(dev,
                          0x40,      // host to device request
                          0xb2,      // “Set Integration Time” bRequest
                           100,      // ms & 0xffff
                             0,      // (ms >> 16) & 0xffff
                           buf,      // payload buffer
                  sizeof(buf),       // size of buffer
                          1000);     // USB timeout in ms
```

The above should return a value in *ret* of 0, which means there was no error. A *ret* value < 0 indicates an error occurred. Please see libusb-0.1 documentation as a reference to the error codes. (Note that opcode response values may vary between FX2 and ARM platforms; see opcode table below.)

To **get** the integration time which the spectrometer is currently using, the following code can be used:

```
char *buf = {0, 0, 0, 0, 0, 0, 0, 0};
int ret = usb_control_msg(dev,
                          0xC0,      // device to host request
                          0xbf,      // “Get Integration Time” bRequest
                             0,      // Doesn’t matter
                             0,      // Doesn’t matter
                           buf,      // output response buffer
                  sizeof(buf),       // size of buf array
                          1000);     // timeout in ms
```

According to the documentation for “Get Integration Time”, 6 bytes will be returned on endpoint 0 and end up in `buf` in LSB order; however, according to the documentation, only the first 3 bytes are used. To parse the integration time, one may use the following code snippet:

```
uint integration_time = buf[0] | (buf[1]<<8) | (buf[2]<<16);
```

3 Command Table

The most recent version of the following table is mastered in ENG-0001 Appendix A (FID USB ICD), but a copy is included for convenience.

Columns are defined as follows:

- **Getter/Setter**: Most commands have both "get" and "set" versions. Some commands can only read data (COMPILATION_OPTIONS) while others can only set (FPGA_RESET).
- **Data Type**: datatype of the primary data value read or written by the command.
  - uint8/12/16/24/40 indicate bit width of the given unsigned integral value. Note that UInt40 parameters, when sent to the spectrometer, are typically split between the USB Control Packet's value, index and payload[0] fields, where the value 0x0123456789 would be sent as value = 0x6789, index = 0x2345 and payload[0] = 0x01.
  - bool indicates only values of 1 or 0 are supported (other values yield undefined behavior)
  - float16 is a proprietary floating-point value in which the MSB represents an integral value (0-255) and the LSB is to be divided by 256 to represent a fractional component. It is not an IEEE 754 half-precision float, nor properly a bfloat16; it is conceptually similar to an unsigned bfloat16.
  - byte[] arrays are passed as literal sequences of bytes
  - string[] are sequences of ASCII characters (similar to byte[], but assumed to be non-null printable characters)
  - enum indicate the argument is technically a uint8 octet, but see the "Enums" column for interpretations of supported zero-indexed value
  - void indicates the opcode does not take or return a primary data value
- **Read Length**: how many bytes of useful information you should get back from the getter.
- **Read-Back Length**: if provided, you should actually read this many bytes back from the USB bus to flush the output buffer; however, only the first "Read Length" bytes will contain useful data.
- **Fake "Get" Buffer Length**: how many bytes you should pass as an "output buffer" when calling the getter, even though getters nominally don't use output buffers. (This deals with some legacy bugs in old firmware.)
- **Set Fake Length From Value**: there are two commands (LASER_MOD_PERIOD and LASER_MOD_PULSE_WIDTH) where, *in addition* to sending the primary value as an argument through wValue, you should also allocate a fake "output buffer" in which the length of the buffer equals the integral value you are trying to set. The contents of this buffer are irrelevant, and may be zero. This addresses a legacy firmware bug which reads the expected "setter" value from the wLength field rather than wValue. This bug is believed fixed in current firmware, and only relates to older FX2 firmware.
- **wValue**: so-called "Second-Tier" commands using opcode **0xff** are individually selected using a specific wValue code. Otherwise, wValue is generally used to hold a parameter being passed to a particular opcode (or perhaps just the LSW thereof).

- **Getter Endianness**: some commands return their data LSB-first (little endian), while others return data MSB-first (big endian). Many return only a single byte, such that endianness does not come into play. Some commands may express a different endianness on different chip architectures.
- **Enums**: see the "enum" Data Type, above.
- **Supports**: some commands only work on models with InGaAs detectors, FX2 or ARM microcontrollers, Zynq chips, the SiG model line etc.
- **Requires Laser**: laser-related commands should not be executed on models without internal lasers, or undefined behavior may occur.
- **Notes**: additional comments about individual commands.

Table 1 FID Opcodes

Name	Getter	Setter	Data Type	Read Len	Read Back Len	Fake Get Buf Len	Set Fake Len From Value	wValue	Getter Endian	Enums	Supports	Req. Laser	Notes
ACQUIRE_SPECTRUM	0xad		void			8			NA				response read back on bulk endpoints 2 and 6
ACTUAL_FRAMES	0xe4		uint16	2					MSB				
ACTUAL_INTEGRATION_TIME	0xdf		uint24	3	6				LSB				response of 0xffffff indicates error
BATTERY_STATE	0xff		mask	3				0x13	MSB		(SiG)		
BATTERY_REG	0xff		uint16	2				0x14			(SiG)		
CCD_START_LINE (GET)	0xff	0xf2	Uint16	2	2				LSB		(SiG)		
CCD_STOP_LINE (GET)	0xff	0xf3	Uint16	2	2				LSB		(SiG)		
CCD_START_LINE (SET)	0xff	0xf4	Uint16	2					LSB		(SiG)		
CCD_STOP_LINE (SET)	0xff	0xf5	Uint16	2					LSB		(SiG)		
CF_SELECT	0xec	0xeb	bool	1					NA		(InGaAs)		on InGaAs indicates high-gain mode enabled
COMPILATION_OPTIONS	0xff					8		4	LSB				

Name	Getter	Setter	Data Type	Read Len	Read Back Len	Fake Get Buf Len	Set Fake Len From Value	wValue	Getter Endian	Enums	Supports	Req. Laser	Notes
DETECTOR_GAIN	0xc5	0xb7	float16	2					LSB				half-precision float (MSB is integer, LSB is fraction)
DETECTOR_OFFSET	0xc4	0xb6	uint16	2					LSB				
DETECTOR_SENSING_THRESHOLD	0xd1	0xd0	uint16	2					LSB				
DETECTOR_TEC_ENABLE	0xda	0xd6	bool	1					NA				
DETECTOR_TEC_SETPOINT	0xd9	0xd8	uint16	2					LSB				
DETECTOR_TEMPERATURE	0xd7		uint16	2					MSB				Raw 12-bit ADC output from the TEC
DETECTOR_THRESHOLD_SENSING_MODE	0xcf	0xce	bool	1					NA				
DFU_MODE		0xfe	void						NA		(ARM)		Dynamic Firmware Upgrade; configures STM32 to accept firmware updates via DfuSe Demonstrator
FIRMWARE_VERSION	0xc0		byte[]	4					LSB				bytes read-out backwards (0xaa bb cc dd] means version dd.cc.bb.aa)
FPGA_FIRMWARE_VERSION	0xb4		string	7					MSB				
HORIZONTAL_BINNING	0xbc	0xb8	enum	1					NA	(NONE, TWO_PIXEL, FOUR_PIXEL)	(ARM)		Uncertain device support

Name	Getter	Setter	Data Type	Read Len	Read Back Len	Fake Get Buf Len	Set Fake Len From Value	wValue	Getter Endian	Enums	Supports	Req. Laser	Notes
INTEGRATION_TIME	0xbf	0xb2	uint24	3					LSB				MS or US per OPT_INT_TIME_RES; sent as 32-bit word (LSW wValue, MSW wIndex, big-endian within each)
LASER_ENABLE	0xe2	0xbe	bool	1					NA			TRUE	
LASER_INTERLOCK	0xef		bool	1					NA		(FX2)	TRUE	
LASER_MOD_DURATION	0xc3	0xb9	uint40	5					LSB			TRUE	
LASER_MOD_ENABLE	0xe3	0xbd	bool	1		8			NA				
LASER_MOD_PERIOD	0xcb	0xc7	uint40	5			TRUE		MSB			TRUE	
LASER_MOD_PULSE_DELAY	0xca	0xc6	uint40	5					LSB			TRUE	
LASER_MOD_PULSE_WIDTH	0xdc	0xdb	uint40	5			TRUE		LSB			TRUE	
LASER_RAMPING_MODE	0xea	0xe9	bool	1					NA		(ARM)	TRUE	
LASER_TEC_SETPOINT	0xe8	0xe7	uint12	1		8			NA		(ARM)	TRUE	setter takes value in range (63, 127)
LASER_TEMPERATURE	0xd5		uint16	2					LSB			TRUE	
LINE_LENGTH	0xff		uint16	2				0x03	LSB				
LINK_LASER_MOD_TO_INTEG_TIME	0xde	0xdd	bool	1					NA			TRUE	

Name	Getter	Setter	Data Type	Read Len	Read Back Len	Fake Get Buf Len	Set Fake Len From Value	wValue	Getter Endian	Enums	Supports	Req. Laser	Notes
MODEL_CONFIG	0xff	0xa2	byte[]	64				0x01	MSB				On read, pass desired page index (0-7) via wIndex. Note that read opcode is 2nd-tier, write opcode is not. On write, wValue should be 0x3c00 + 64 * (zero-indexed page index); buf should be 64 bytes
OPT_ACTUAL_INTEGRATION_TIME	0xff		bool	1		8		0x0b	NA				
OPT_AREA_SCAN	0xff		bool	1		8		0x0a	NA				
OPT_CF_SELECT	0xff		bool	1		8		0x07	NA				Indicates support for “High-Gain Mode” on InGaAs
OPT_DATA_HEADER_TAB	0xff		enum	1		8		0x06	NA	(NONE, OCEAN OPTICS, WASATCH)			Unclear hardware support
OPT_HORIZONTAL_BINNING	0xff		bool	1				0x0c	NA				Unclear hardware support
OPT_INTEGRATION_TIME_RESOLUTION	0xff		enum	1				0x05	NA	(ONE MS, TEN MS, SWITCHABLE)			Unclear hardware support
OPT_LASER_TYPE	0xff		enum	1				0x08	NA	(NONE, INTERNAL, EXTERNAL)			
OPT_LASER_CONTROL	0xff		enum	1				0x09	NA	(MODULATION, TRANSITION POINTS, RAMPING)			Unclear hardware support

Name	Getter	Setter	Data Type	Read Len	Read Back Len	Fake Get Buf Len	Set Fake Len From Value	wValue	Getter Endian	Enums	Supports	Req. Laser	Notes
RESET_FPGA		0xb5	void						NA				attempts to perform a runtime reset (power cycle) of the FPGA
SELECTED_ADC	0xee	0xed	enum	1					NA	(PRIMARY, SECONDARY)			Typically laser thermistor or photodiode if present
TRIGGER_DELAY	0xab	0xaa	uint24	3					LSB		(ARM)		Delay is in 0.5us, supports 24-bit unsigned value (about 8.3sec max)
TRIGGER_OUTPUT	0xe1	0xe0	enum	1					NA	(LASER MODULATION, INTEGRATION ACTIVE PULSE)			
TRIGGER_SOURCE	0xd3	0xd2	enum	1					NA	(USB, EXTERNAL)			
SET_CONTINUOUS_ACQUISITION	0xcc	0xc8	bool	1					NA				When using external triggering, perform multiple acquisitions with a single inbound trigger event
SET_CONTINUOUS_FRAMES	0xcd	0xc9	uint8	1					NA				When using continuous CCD acquisitions with external triggering, how many spectra are being acquired per trigger event.

4 Command Detail

Following are the valid commands for the USB interface. All commands are sent as Vendor Requests. The data packet for "set" commands can be a NULL pointer, except for the following commands:

- SET_LASER_MOD_DURATION
- SET_LASER_MOD_PULSE_DELAY
- SET_LASER_MOD_PERIOD
- SET_LASER_MOD_PULSE_WIDTH

In the above commands, a reference to at least one byte must be sent. If these high-bits (see command format) are not used (often the case), then the value of this byte reference should be zero. Alternatively, the data packet has a maximum length of 8 bytes. As a default, a reference to all 8 bytes set to zero can be sent.

In this section's tables, note the following:

- bmRequestType = "bit mask", as bmRequestType marshalls together the following 3 values:

7	6	5	4	3	2	1	0

  Bit 7: Data Phase Direction (0 = HOST → DEVICE, 1 = DEVICE → HOST)
  Bits 6-5: Type (2 = Vendor)
  Bits 4-0: Recipient (0 = Device)

  All bmRequestType values will be 0x40 (setters, commands) or 0xC0 (getters, requests)

- Uint40 values (40-bit unsigned ints) are passed by sending the LSW (Least Significant Word) as wValue, the next 16 bits as wIndex, and the MSB (Most Significant Byte) in the first byte of the payload record. For legacy reasons, the payload record itself should normally be 8 bytes in length; the value of the other 7 bytes does not matter and can be zero.
- Uint24 (24-bit unsigned ints) are passed by sending the LSW in wValue, and the MSB in the least-significant 8 bits of wIndex (the upper half of wIndex is ignored and can be left zero).

4.1 Acquire Image (ACQUIRE_SPECTRUM)

Commands the detector to begin a new acquisition using the currently-applied integration time, gain/offset etc. The response from this command is not available until at least "integration time" milliseconds after it is received by the spectrometer (potentially a minute or more for long measurements).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xAD	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

Unlike most other spectrometer commands, the response for this command is not returned on the same "Control Endpoint 0" used to issue the command. Spectrometers with linear-array detectors of 1024 or

fewer pixels will find all pixels returned as uint16 (LSB-first) values on bulk-output endpoint 2 (0x82). That is, 1024-pixel detectors will return 2048 bytes, while 512-pixel detectors will return 1024 bytes, etc.

Larger linear-array detectors with up to 2048 pixels will output the remaining spectrum on bulk-output endpoint 6 (0x86).

The SiG series of 2D sensors outputs all pixels on endpoint 2 (0x82), regardless of pixel count or configured Region of Interest.

4.2 Set Integration Time (SET_INTEGRATION_TIME)

Sets the integration time which will be used in subsequent acquisitions. On most spectrometers, the unit is in milliseconds; check FPGA compilation options (READ_COMPILATION_OPTIONS) to confirm integration time resolution. On older spectrometers, the default integration time will be zero (shortest-possible acquisition supported by the detector), while newer models can specify a startup integration time using the "STARTUP_INTEGRATION_TIME" field in the EEPROM (see ENG-0034).

Regardless of unit, the integration time is passed as a 24-bit value, meaning it is split across the wValue and wIndex USB control fields.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xB2	Request
2	wValue	2	Integration Time LSW	Least-significant 16 bits of value
4	wIndex	2	Integration Time MSW	Most-signifiant 8 bits of value
6	wLength	2	0	Payload size

Integration time can be expressed as:

Integration Time = wValue [0] + wValue [1]<<8 + wIndex [0]<<16

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.3 Get FPGA Firmware Revision (GET_FPGA_FIRMWARE_VERSION)

Reads the revision of the FPGA firmware code.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xB4	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

The FPGA revision will appear as seven bytes of ASCII codes on endpoint 0. Returns value < 0 if unsuccessful.

4.4 Set Detector Signal Offset (SET_DETECTOR_OFFSET)

Sets the offset added to the detector pixel values. Defaults to 0 on reset, on older spectrometers. On newer models, the startup value can be configured in the STARTUP_OFFSET_EVEN and STARTUP_OFFSET_ODD EEPROM fields (see ENG-0034).

The "correct, expected" behavior is to interpret the parameter as a signed Int16 which is added to the measured spectrum, with underflow protection enabled. Some spectrometers will add the offset to the spectrum; others will subtract it. Some spectrometers interpret the parameter as a signed Int16; others treat it as a UInt16. Some spectrometers handle rollover (going below 0, or above 65535) properly by clamping to the supported UInt16 pixel range; others may exhibit odd spectral response. Actual behavior depends on configured firmware load, especially in the FPGA.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xB6	Request
2	wValue	2	Offset (15:0)	Value
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.5 Set Detector Signal Gain (SET_DETECTOR_GAIN)

Sets the gain by which detector pixel values are multiplied. Note that the value is passed as an unsigned bfloat16 16-bit float. Defaults to 1.9 on reset, on older spectrometers. On newer models, the startup gain can be set via the DETECTOR_GAIN_EVEN and DETECTOR_GAIN_ODD EEPROM fields (see ENG-0034).

To marshall or demarshall a 16-bit float, refer to the following bit values:

```
Bit 15: 128                    Bit 7: 1/2
Bit 14:  64                    Bit 6: 1/4
Bit 13:  32                    Bit 5: 1/8
Bit 12:  16                    Bit 4: 1/16
Bit 11:   8                    Bit 3: 1/32
Bit 10:   4                    Bit 2: 1/64
Bit  9:   2                    Bit 1: 1/128
Bit  8:   1                    Bit 0: 1/256
```

Therefore, the value 0x1234 would be parsed as follows:

```
MSB 0x12 = decimal 18
LSB 0x34 = b0011 0100 = 1/8 + 1/16 + 1/64 = 13/64 = decimal 0.203125
           7654 3210 (bit position)
```

Value = 18.203125

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xB7	Request
2	wValue	2	Gain (15:0, see format above)	Value
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.6 Set Laser Modulation Duration (SET_LASER_MOD_DURATION)

When modulating the laser, this command selects the time the laser is being modulated during the integration time (in µs).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xB9	Request
2	wValue	2	Duration (15:0)	Duration LSW
4	wIndex	2	Duration (31:16)	Duration bits 16-31
6	wLength	2	1	Payload size
8	Data	1	Duration (39:32)	Duration MSB (mandatory, even if 0)

So Laser Modulation Duration can be expressed as:

Duration (µs) = wValue[0] + wValue[1]<<8 + wIndex[0]<<16 + wIndex[1]<<24 + data[0]<<32

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.7 Set Laser Modulation (SET_LASER_MOD_ENABLE)

The command enables or disables laser modulation, the standard way to control laser output power as a percentage of full power (unmodulated output). On reset laser modulation is disabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xBD	Request

Offset	Field	Size	Value	Description
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.8 Enable/Disable Laser or External Trigger Signal (SET_LASER_ENABLE)

The command enables or disables the internal laser; or when dealing with an external trigger, controls the external trigger signal. On reset the laser is disabled, so external trigger signaling must be preceded by enabling this line.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xBE	Request
2	wValue	2	0 = disable 1 = enable	Value
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.9 Get Integration Time (GET_INTEGRATION_TIME)

The command reads the integration time. The return value is generally in milliseconds; see OPT_INTEGRATION_TIME_RESOLUTION to confirm units for your spectrometer.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xBF	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Integration time (ms) = data[0] + data[1]<<8 + data[0]<<16

Response

The integration time shows up as 6 bytes on endpoint 0, but only the first 3 bytes are used (LSB first); you may ignore the last 3 bytes.

4.10 Get Microcontroller Firmware Version (GET_FIRMWARE_VERSION)

The command reads the firmware version of the FX2 or ARM microcontroller.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xC0	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

```
char version[16];
sprintf(version, "%u.%u.%u.%u", data[3], data[2], data[1], data[0]);
```

Response

For PIDs 0x1000 or higher, the response is four bytes (older units may return 2 bytes). The response is LSB-first, so if you read [0xaa, 0xbb, 0xcc, 0xdd] that indicates firmware version dd.cc.bb.aa.

4.11 Get Laser Modulation Duration (GET_LASER_MOD_DURATION)

The command reads the laser modulation duration, as previously set by the user. Defaults to 0.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xC3	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

```
Duration (µs) = data[0] + data[1]<<8 + data[2]<<16 + data[3]<<24 + data[4]<<32
```

Response

The laser modulation time is returned as five bytes on endpoint 0 (LSB first) in µs.

4.12 Get Detector Signal Offset (GET_DETECTOR_OFFSET)

Reads the offset added to the CCD pixel values.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xC4	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns detector offset (two bytes) on endpoint 0 if command was successful. ARM-based products returns value < 0 if unsuccessful. FX2-based products return 1 if successful and 0 if unsuccessful.

4.13 Get Detector Signal Gain (GET_DETECTOR_GAIN)

Reads the detector signal gain, as an unsigned bfloat16 (see SET_DETECTOR_GAIN).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xC5	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

Returns CCD signal gain (two bytes) on endpoint 0 if command was successful. Returns value < 0 if unsuccessful.

4.14 Set Laser Modulation Pulse Delay (SET_LASER_MOD_PULSE_DELAY)

When modulating laser, this command selects the delay from the start of integration to the beginning of laser modulation (in µs).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC6	Request
2	wValue	2	Delay (15:0)	Delay LSW
4	wIndex	2	Delay (31:16)	Delay bits 16-31
6	wLength	2	1	Payload length
8	Data	1	Delay (39:32)	Delay MSB (mandatory, even if 0)

Delay (µs) = wValue[0] + wValue[1]<<8 + wIndex[0]<<16 + wIndex[1]<<24 + data[0]<<32

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.15 Set Laser Modulation Period (SET_LASER_MOD_PERIOD)

When modulating laser, this command sets the period of the laser modulation. The unit and resolution is 1us.

Typical values for modulation period are 100µs or 1000µs, sometimes referred to as "low-resolution" and "high-resolution" laser power, as the first allows the laser power to be easily set as integral percentages of full power (e.g. 63%), and the second allows percentages to be set to 0.1% precision (e.g. 63.7%).

Note that laser output power is not linear across the range of modulation. To reliably command the laser to a desired output power in mW, a laser power calibration is required. That calibration will only be valid across a portion of the laser's output power range. Such calibrations are also specific to the modulation period used when the calibration was generated; a calibration generated with a period of 100µs will not give the expected results when applied at a period of 1000µs.

When laser power ramping is enabled, this value determines the length of the ramp.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC7	Request
2	wValue	2	Period (15:0)	Period LSB
4	wIndex	2	Period (31:16)	Period bits 16-31
6	wLength	2	1	Payload size
8	Data	1	Period (39:32)	Period MSB (mandatory, even if 0)

So Laser Pulse Period can be expressed as:

Period (µs) = wValue[0] + wValue[1]<<8 + wIndex[0]<<16 + wIndex[1]<<24 + data[0]<<32

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.16 Get Laser Modulation Pulse Delay (GET_LASER_MOD_PULSE_DELAY)

The command reads the laser modulation pulse delay.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xCA	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Delay (µs) = data[0] + data[1]<<8 + data[2]<<16 + data[3]<<24 + data[4]<<32

Response

Response shows up as five bytes on endpoint 0 (LSB first).

4.17 Get Laser Modulation Period (GET_LASER_MOD_PERIOD)

The command reads the laser modulation period.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xCB	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Period (µs) = data[0] + data[1]<<8 + data[2]<<16 + data[3]<<24 + data[4]<<32

Response

Response shows up as five bytes on endpoint 0 (LSB first).

4.18 Set Detector Data Threshold Sensing (SET_DETECTOR_THRESHOLD_SENSING_MODE)

The command enables or disables data threshold sensing of the detector data. When enabled, each pixel of the incoming detector data is compared to a user-defined threshold. If any pixel exceeds the threshold, a bit in the FPGA status register is set. When disabled, the status bit is always low. On reset, detector data threshold sensing is disabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xCE	Request
2	wValue	2	0 = disable 1 = enable	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.19 Get Detector Data Threshold Sensing Mode (GET_DETECTOR_THRESHOLD_SENSING_MODE)

The command reads the state of the CCD Data Threshold Sensing mode.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xCF	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (threshold sensing disabled) or 1 (threshold sensing enabled) on endpoint 0.

4.20 Set Detector Data Sensing Threshold Level (SET_DETECTOR_SENSING_THRESHOLD)

Sets the threshold for sensing the detector data. If any value in the detector data frame exceeds this setpoint, then the POLL_DATA command will return a value of 2 rather than 1. Defaults to 0 on reset.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xD0	Request
2	wValue	2	Threshold	Value (uint16)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.21 Get Detector Data Sensing Threshold (GET_DETECTOR_SENSING_THRESHOLD)

The command reads the detector data sensing threshold.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD1	Request
2	wValue	2	0xXXXX	Value (n/a)
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

Response shows up as two bytes on endpoint 0 (LSB first).

4.22 Set Trigger Source (SET_TRIGGER_SOURCE)

Sets trigger source for acquiring data. Defaults to 0 (USB software command trigger) on reset.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xD2	Request
2	wValue	2	0 = USB SW command (default) 1 = external HW trigger	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.23 Get Trigger Source (GET_TRIGGER_SOURCE)

The command reads the trigger source for data acquisition.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD3	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 1 byte on Endpoint 0. 0 indicates USB software command trigger, or 1 for external HW trigger. Returns value < 0 if unsuccessful.

4.24 Detector Data Polling (POLL_DATA)

The command returns a value indicating whether detector data is available and whether any of the pixels crossed the data-sensing threshold.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD4	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response as 4 bytes on Endpoint 0.

Byte Number	Size	Value
1	1	0 = no data is available 1 = data is available (none crossed the sensing threshold) 2 = data is available (some crossed the sensing threshold)
2-4	3	Reserved for future use

4.25 Get Laser Temperature (GET_LASER_TEMPERATURE, aka GET_ADC)

In normal operation, reads the Analog-to-Digital Converter (ADC) wired to the thermistor mounted to the laser cavity housing. However, technically the command can be used to read other ADCs in the system, which is why it is also known as GET_ADC. See the SELECT_ADC command for information about how to point the GET_ADC command to other targets besides the internal laser thermistor.

The returned value is a "raw" (unitless) 12-bit ADC response scaled to the thermistor voltage, and can be converted to °C using the following equation. Note that all Wasatch Photonics lasers are monitored with the same model thermistor (TDK B57862S0103F040), hence the coefficients do not need to be calibrated per-model or per-unit.

```
// curve-fit per the datasheet
// @see https://media.digikey.com/pdf/Data%20Sheets/Epcos%20PDFs/B57862.pdf
voltage    = 2.5 * raw / 4096
resistance = 21450 * voltage / (2.5 - voltage)
if resistance > 0:
   logVal     = math.log(resistance / 10000)
   insideMain = logVal + 3977 / (25 + 273)
   degC       = 3977 / insideMain - 273
```

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD5	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Two bytes of temperature reading (12-bit value with the top four bits zeroed out) on endpoint 0. Note this value is little-endian, unlike the similar GET_DETECTOR_TEMPERATURE.

4.26 Enable/Disable Detector Thermo-Electric Cooler (SET_DETECTOR_TEC_ENABLE)

The command enables or disables the Thermo-Electric Cooler (TEC) on the detector. When enabled the TEC chip on the TEC Board will attempt to maintain the detector at the defined setpoint. On reset, the TEC is disabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xD6	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.27 Get Detector Temperature (GET_DETECTOR_TEMPERATURE)

Gets a reading scaled to the voltage of the thermistor mounted next to the detector.

Note that while this command reads an ADC, similar to the GET_LASER_TEMPERATURE command, it is hard-coded to read a specific ADC. While the GET_LASER_TEMPERATURE command is in effect reading an arbitrary and selectable ADC (which simply happens to default to the laser thermistor), the GET_DETECTOR_TEMPERATURE command is locked to the detector thermistor and cannot be changed.

The response is a "raw" (unitless) ADC reading. To convert to °C, use the 3rd-order "ADC to DegC" polynomial coefficients in your spectrometer's EEPROM (see ENG-0034). The resulting equation would be:

$$\text{Temperature °C} = C_0 + C_1(\text{raw}) + C_2(\text{raw}^2)$$

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD7	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Two bytes of temperature reading (12-bit value with the top four bits zeroed out) on endpoint 0. Note this value is big-endian, unlike GET_LASER_TEMPERATURE.

4.28 Read Detector TEC Enable (GET_DETECTOR_TEC_ENABLE)

The command reads whether the Thermo-Electric Cooler (TEC) controlling the detector temperature is enabled and running.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xDA	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (detector TEC disabled) or 1 (detector TEC enabled) on endpoint 0.

4.29 Get Internal Frame Count (GET_ACTUAL_FRAMES)

This command returns the internal frame/capture count. It is incremented based on internal (USB-command) captures and external trigger-based captures. It is reset to 0 upon power up, or on rollover as the UInt16 maximum value is exceeded.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device →Host
1	bRequest	1	0xE4	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

4.30 Set Detector TEC Setpoint / Set DAC (SET_DETECTOR_TEC_SETPOINT)

This command has two modes.

In the general case, it writes a 12-bit value (0-4095) to a Digital-to-Analog Converter (DAC), which scales the value to generate an output voltage of 0-5VDC. In normal operation, this command is used to specify the setpoint around which the Thermo-Electric Cooler (TEC) attempts to stabilize the temperature of the detector.

However, some spectrometers have more than one DAC, and this command can be used to set any of them. In particular, some spectrometers control an external laser (NOT the "internal" laser used with Wasatch -L Raman systems), in which this command can also be used to control a secondary DAC that determines the output power of that external laser.

The target DAC is specified using the wIndex parameter. The DAC parameter is always a 12-bit value, and the upper four bits are unused and may be left zero.

Note that when commanding the TEC setpoint, the unit of the value set represents approximately 1/4096 (0.0002) of one volt — this is not set directly in °C. To convert from the intended setpoint in °C to the equivalent DAC value, use the "degCtoDAC" coefficients in the spectrometer's EEPROM (see ENG-0034).

Given the goal temperature T in °C, and the 3 coefficients C_0 through C_2, the resulting equation would be evaluated like this:

$$\text{Uint16 raw\_dac\_value} = \text{round}(C_0 + C_1T + C_2T_2)$$

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xD8	Request
2	wValue	2	12-bit setpoint value (top four bits are don't care)	Value
4	wIndex	2	0 = detector TEC setpoint 1 = secondary DAC (e.g. external laser power, etc)	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.31 Get Detector TEC Setpoint / Get DAC (GET_DETECTOR_TEC_SETPOINT, aka GET_DAC)

Reads one of the two DAC settings as a 12-bit value (0 to 4095) corresponding to a voltage between 0V and ~5V. If the index value is 0, the value read is from the detector TEC setpoint DAC; if the index value is 1, the value read is from the secondary DAC (which may be configured to control the power of an external laser or other peripherals depending on your system configuration).

As with SET_DETECTOR_TEC_SETPOINT above, the returned value will not be in °C, and should simply return the most-recently set DAC value that you assigned.

Note that the TEC is essentially a write-only component (via setpoint), and the thermistor is a read-only component. To measure the actual detector temperature, please see GET_DETECTOR_TEMPERATURE to read the detector thermistor and convert back to °C.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD9	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0 = detector TEC setpoint 1 = secondary DAC	Index
6	wLength	2	0	Payload size

Response

Two bytes of DAC reading (12-bit value with the top four bits zeroed out) on endpoint 0 (LSB first).

4.32 Set Laser Modulation Pulse Width (SET_LASER_MOD_PULSE_WIDTH)

When modulating laser, this command sets the pulse width of the laser modulation in 1µs increments.

This is the standard way to control laser output power, by setting a "duty cycle" and modulating (or pulsing) the nominally "continuous wave" laser.

The laser modulation pulse width must be lower than the configured laser modulation pulse period. If the period is set to 100µs, and the width is set to 33µs, then the laser will be "on" for 33 out of every 100µs, and will be running at approximately 1/3 full power.

There are several factors which contribute to that "approximately":

- Laser output power is not guaranteed linear across the full range of modulation
- Modulated output power varies with the modulation period; a duty cycle of 33/100µs can yield different output power than 333/1000µs, especially as measured by a high-speed photodiode which may have its own timing configuration and potential wave interference
- Some lasers may have a "minimum output lasing power," and will not fire at all below a set level; this minimum power may be tunable via a hardware potentiometer (for instance), and may be itself related to the modulation period

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xDB	Request
2	wValue	2	Pulse Width (15:0)	Value (LSW)
4	wIndex	2	Pulse Width (31:16)	Index (middle bits)
6	wLength	2	1	Payload size
8	Data	1	Pulse Width (39:17)	First byte is MSB (then 7 zeros)

Width (µs) = wValue[0] + wValue[1]<<8 + wIndex[0]<<16 + wIndex[1]<<24 + data[0]<<32

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.33 Get Laser Modulation Pulse Width (GET_LASER_MOD_PULSE_WIDTH)

The command reads the laser modulation pulse width.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xDC	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response shows up as five bytes on endpoint 0 (LSB first).

4.34 Link Laser Modulation to Integration Time (SET_LINK_LASER_MOD_TO_INTEGRATION_TIME)

The command links the state of whether the laser is actively being modulated or not (when permitted by other related settings) to the integration time.

When this linkage is enabled AND laser modulation is enabled, the laser is modulated only while integrations are taking place, and the laser reverts to unmodulated (full power) when integration is complete.

When the link is disabled AND laser modulation is enabled, the laser modulates continuously, regardless of whether the detector is acquiring or not.

Regardless of linkage setting, if SET_LASER_MODULATION is disabled, the laser will not be modulated. Likewise, if SET_LASER_ENABLE is disabled, the laser will not fire (and therefore will not be modulated).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xDD	Request
2	wValue	2	0 = don't link modulation to integration 1 = link modulation to integration	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.35 Read Status of Link Laser Modulation to Integration Time (GET_LINK_LASER_MOD_TO_INTEGRATION_TIME)

The command reads whether the laser modulation is linked to the integration time.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xDE	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (modulation not linked to integration) or 1 (modulation linked to integration) on endpoint 0. Returns value < 0 if unsuccessful.

4.36 Get Actual Integration Time (GET_ACTUAL_INTEGRATION_TIME)

The command reads the actual integration time. The actual integration time is returned in microseconds (µs), unlike commanded integration time (typically ms), and irrespective of the compiled integration time resolution.

Actual integration time differs from the commanded integration time under two conditions:

1. In standard USB-based acquisition, the actual integration time will add the clock-out (readout) time.
2. Under conditions of external triggering, the integration window will be extended to include the output laser pulse time, thus extending the commanded integration time.

Stated differently, this is the time required to read the detector pixel values into the FPGA and reset the sensor such that it could theoretically start the next integration, if Wasatch spectrometers had a free-running mode.

Use the READ_COMPILATION_OPTIONS command to determine if this function is available.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xDF	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Actual Integration Time (µs) = data[0] + data[1]<<8 + data[2]<<16

Response

The integration time shows up as six bytes on endpoint 0 but only the first three bytes are used (LSB first). Returns 0xFFFFFF if command failed.

4.37 Select Trigger Output (SET_TRIGGER_OUTPUT)

The command selects which signal ("laser firing" or "integration active") the spectrometer will output on the external trigger pin.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xE0	Request
2	wValue	2	0 = output laser modulation status 1 = output integration status	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.38 Get Trigger Output (GET_TRIGGER_OUTPUT)

The command returns whether the output trigger pin is configured to output a "laser firing" or "integration active" signal.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xE1	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (laser modulation output) or 1 (integration active pulse) on endpoint 0.

4.39 Get Laser Enable/Disable Status (GET_LASER_ENABLE)

The command returns status indicating the laser is enabled or disabled. Note that when laser modulation is enabled, the laser is considered to be "firing" continuously when the laser is enabled, even though technically it may be "pulsing" at some frequency to achieve lower power levels.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xE2	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 0 (laser disabled) or 1 (laser enabled) in one byte on endpoint 0.

4.40 Get Laser Modulation (GET_LASER_MOD)

The command returns status indicating laser modulation is enabled or disabled. Laser modulation is used to "pulse" the laser at a high frequency (typically 100Hz or 1Khz), such that the duty cycle (the fraction of each pulse period in which the laser is actually firing) can be scaled to yield variable output power levels. By definition, the laser's "full power" is achieved with laser modulation disabled, such that the beam is continuous and without interruption.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xE3	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 0 (no modulation / full power) or 1 (laser modulation enabled) in one byte on endpoint 0. Returns value < 0 if unsuccessful.

4.41 Set Laser TEC Setpoint (SET_LASER_TEC_SETPOINT)

Sets the setpoint in the laser's Thermo-Electric Cooler (TEC) used to maintain a constant laser temperature.

Note that this is a unitless measure, and is not °C. No coefficients are provided to convert this value to an actual temperature. Engineering documentation suggests this value should not be set outside the 7-bit range (0, 127) inclusive.

This value is used exclusively during manufacturing calibration to balance a hardware potentiometer, and is not recommended for end-user manipulation. The supported temperature maintained in the laser is prescribed by the laser manufacturer and is not tunable or adjustable by customers.

Essentially, this command allows manufacturing technicians to nudge the laser temperature up and down slightly, or cycle it over a range, while tuning a board-level potentiometer to a stable point well away from temperatures associated with "mode-hops" in single-mode lasers.

On reset the laser TEC setpoint is 63, the midpoint of the runtime-configurable range.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xE7	Request
2	wValue	2	Set Point (15:0)	Value (63-127)
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.42 Get Laser TEC Setpoint (GET_LASER_TEC_SETPOINT)

The command reads the laser temperature set point. See "SET_LASER_TEC_SETPOINT" above for warnings about this value.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xE8	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload

Response

The laser temperature set point shows up as 6 bytes on endpoint 0 but only the first byte is used (and only the first 7 bits of that).

4.43 Enable/Disable Laser Power Ramping (SET_LASER_RAMPING_MODE)

The command enables or disables ramping of the laser power when the laser is being turned on or off. When ramping the laser, the configured Laser Modulation Period is used as the period of each successive step in the laser ramp.

The generated laser output power ramp is linear, and will steadily increment or decrement the applied laser modulation pulse width from the "start state" (whether the laser started "off" or "on"), to the "end state" (opposite of the start state).

By way of example, suppose that LASER_MOD_PERIOD is set to 100 (typical), LASER_MOD_PERIOD (*n*) is set to 5 (~5% laser power), LASER_RAMPING_MODE is enabled, and LASER_ENABLE is changed from 0 (disabled) to 1 (enabled). The laser will gradually "ramp up" and later "ramp down" over a period of 400µs ((*n* – 1)(LASER_MOD_PERIOD)) in each direction:

- *(laser is off)*
- User sets LASER_ENABLE → 1
- Pass 1: for 100µs (LASER_MOD_PERIOD), laser is on 1µs and off 99µs
- Pass 2: for 100µs (LASER_MOD_PERIOD), laser is on 2µs and off 98µs
- Pass 3: for 100µs (LASER_MOD_PERIOD), laser is on 3µs and off 97µs
- Pass 4: for 100µs (LASER_MOD_PERIOD), laser is on 4µs and off 96µs
- *(laser is now firing continuously at 5% power)*
- User sets LASER_ENABLE → 0
- Pass 1: for 100µs (LASER_MOD_PERIOD), laser is on 4µs and off 96µs
- Pass 2: for 100µs (LASER_MOD_PERIOD), laser is on 3µs and off 97µs
- Pass 3: for 100µs (LASER_MOD_PERIOD), laser is on 2µs and off 98µs
- Pass 4: for 100µs (LASER_MOD_PERIOD), laser is on 1µs and off 99µs
- *(laser is now off)*

On reset laser ramping is disabled. Not available in FX2.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xE9	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 0 if command was successful. Returns value < 0 if unsuccessful.

4.44 Get Laser Power Ramping Mode (GET_LASER_RAMPING_MODE)

The command reads the state of the Laser Power Ramping mode. Not available in FX2.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xEA	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (laser power ramping disabled) or 1 (laser power ramping enabled) on endpoint 0.

4.45 Get Interlock (GET_INTERLOCK)

The command returns the status of the laser interlock.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xEF	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (interlock circuit open, laser cannot fire) or 1 (interlock circuit open, laser can fire) on endpoint 0. Not supported on ARM.

4.46 Select ADC (SET_SELECTED_ADC)

The command selects the active Analog-to-Digital Converter (ADC) for the next "GET_ADC" command. At reset, the selected ADC is the one coupled to the laser thermistor, which is why the "GET_ADC" command is typically called "GET_LASER_TEMPERATURE". However, many spectrometers have a secondary ADC which can be coupled to different components in OEM system designs (for instance, a second laser, or a photodiode).

When calling SELECT_ADC, 0 will reset to the default target (typically the internal laser temperature thermistor), while a value of 1 will indicate the secondary ADC if one is present.

Note that after changing the ADC selector, it is recommended to perform a "throwaway read" (a redundant call to GET_ADC) to fully synchronize the read cycle and ensure the next read operation does not contain "mingled data" from both components.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xED	Request
2	wValue	2	0 = primary ADC (laser thermistor) 1 = secondary ADC (if present)	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.47 Get Selected ADC (GET_SELECTED_ADC)

The command returns the index of the selected ADC (see SET_SELECTED_ADC).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xEE	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (primary ADC selected) or 1 (secondary ADC selected) on endpoint 0.

4.48 Select Horizontal Binning (SET_HORIZONTAL_BINNING)

The command selects the type of horizontal binning performed on the sensor data, if any. 0 selects no binning. 1 selects two-pixel binning. 2 selects four-pixel binning. On reset no binning is selected. Not available in FX2.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xB8	Request
2	wValue	2	0 = no binning 1 = 2-pixel binning 2 = 4-pixel binning	Binning mode
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 0 if command was successful. Returns value < 0 if unsuccessful.

4.49 Get Horizontal Binning (GET_HORIZONTAL_BINNING)

The command returns the selected horizontal binning mode. Not available in FX2.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xBC	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Response is 0 (no binning), 1 (2-pixel binning) or 2 (4-pixel binning) on endpoint 0.

4.50 Set Trigger Delay (SET_TRIGGER_DELAY)

Sets the delay from the trigger to the start of integration time. Resolution of the trigger delay is 0.5 µs. On reset the trigger delay is set to 0. Not available in FX2.

Example: a configured trigger delay of 50 will cause each acquisition to begin 25µs after receipt of the trigger.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xAA	Request
2	wValue	2	Trigger Delay LSW	Value (bits 0-15)
4	wIndex	2	Trigger Delay MSB	Value (bits 16-23)
6	wLength	2	0	Payload size

Response

Returns 0 if command was successful. Returns value < 0 if unsuccessful.

4.51 Get Trigger Delay (GET_TRIGGER_DELAY)

The command reads the trigger delay. Not available in FX2.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xAB	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

The trigger delay shows up as 6 bytes on endpoint 0 but only the first 3 bytes are used (LSB first) – ignore response on last 3 bytes. Trigger delay (x .5 us) = $B_0 + B_1 << 8 + B_2 << 16$.

4.52 Get Model Info (GET_MODEL_CONFIG)

The command reads the model configuration information stored on the spectrometer's EEPROM. This data includes the serial number, model, wavelength calibration, temperature coefficients and many other key attributes defining the runtime abilities, limitations and defaults of the spectrometer.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x0001	Value
4	wIndex	2	Page index (0-5)	See ENG-0034 for EEPROM page indices and their contents
6	wLength	2	64	Payload size

Response

This command returns 64 bytes (the size of a single EEPROM page). Refer to ENG-0034 for appropriate parsing and demarshalling of a particular page's contents, and the number of supported pages.

4.53 Set Model Info (SET_MODEL_CONFIG)

The command stores the model configuration information, dynamically overwriting the internal EEPROM.

This is one of the most dangerous commands in the API, because improperly-formatted EEPROM pages, or fields containing out-of-range values, could corrupt, "brick" (render uncommunicative), physically damage your spectrometer (by overheating / overclocking components) or even risk human injury (if laser settings are impacted).

Users are highly advised not to alter the contents of their EEPROM without using purpose-built, approved and tested tools released by Wasatch Photonics. Any unsupported alterations to the EEPROM will void the spectrometer warranty and may require factory RMA.

Note that commands to write the EEPROM differ between FX2 and ARM architectures.

Format (FX2)

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xa2	bRequest
2	wValue	2	0x3c00 + (64 x page_index)	See ENG-0034 for EEPROM page contents and marshalling instructions
4	wIndex	2	0	wIndex (not used)
6	wLength	2	64	Payload size

Example: to write EEPROM page 3 (the fourth zero-indexed page of 64 bytes):

wValue = 0x3c00 + (64 x 3) = 0x3cc0

Format (ARM)

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xff	Second-tier bRequest
2	wValue	2	0x02	opcode
4	wIndex	2	0-7	EEPROM page index
6	wLength	2	64	Payload size

Response

Returns 1 if command was successful. Returns value 0 if unsuccessful.

4.54 Get Sensor Line Length (GET_LINE_LENGTH)

Gets the size of the sensor line (number of pixels in the returned spectrum). This value should agree with the configured EEPROM field active_horizontal_pixels.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x03	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	2	Payload size

Response

Returns the size of a sensor line (2 bytes, LSB first).

4.55 Get FPGA Compilation Options (READ_COMPILATION_OPTIONS)

Gets the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x04	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	2	Payload size

Response

Two bytes of FPGA compilation options. See ENG-0034 documentation for table showing field structure within the returned 16-bit mask.

4.56 Integration Time Resolution (GET_OPT_INTEGRATION_TIME_RESOLUTION)

Gets the integration time resolution from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x05	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte:

- 0: 1 ms
- 1: 10 ms
- 2: Switchable between 1 ms and 10 ms

4.57 Data Header or Tag (GET_OPT_DATA_HEADER_TAG)

Gets the data header or tag option from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x06	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte containing:

- 0 = no header or tag
- 1 = Ocean Optics "OceanBinaryProtocol" (OBP) header and tag
- 2 = Wasatch tag

4.58 CF Select Available (GET_OPT_CF_SELECT)

Indicates whether the FPGA was compiled with a selectable "CF Select" (Clock Frequency) switch. This option is currently only used on InGaAs detectors to toggle "High-Gain Mode" by adjusting the voltage which determines the photonic conversion efficiency.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x07	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte:

- 0 = CF Select disabled (default)
- 1 = CF Select enabled (InGaAs High Gain Mode)

4.59 Laser Type Available (GET_OPT_LASER_TYPE)

Gets the type of laser from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x08	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte:

- 0 = No laser
- 1 = Internal laser
- 2 = External laser

4.60 Laser Control Type Available (GET_OPT_LASER_CONTROL)

Gets the type of laser control from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x09	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte:

- 0 = Laser modulation
- 1 = Laser transition points
- 2 = Laser ramping

4.61 Area Scan Available (GET_OPT_AREA_SCAN)

Gets the area scan availability from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x0A	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte (0 = Not available, 1 = Available)

4.62 Actual Integration Time Available (GET_OPT_ACTUAL_INTEGRATION_TIME)

Returns whether the FPGA was compiled with options to support actual integration time.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x0B	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte (0 = Not available, 1 = Available)

4.63 Horizontal Binning Available (GET_OPT_HORIZONTAL_BINNING)

Gets the horizontal binning availability from the FPGA compilation options register.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x0C	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

One byte (0 = Not available, 1 = Available)

4.64 Enable/Disable Continuous Acquisition (SET_CONTINUOUS_ACQUISITION)

This command enables or disables continuous read mode from the CCD. When continuous acquisition is enabled AND external triggering is enabled, a single trigger event will trigger multiple spectral acquisitions. The number of acquisitions to perform is set by the SET_TRIGGER_FRAMES command, below.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC8	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.65 Get Continuous Acquisition (GET_CONTINUOUS_ACQUISITION)

This command reads whether continuous acquisition mode is enabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xCC	Request
2	wValue	2	0xXXXX	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 0 if disabled, 1 if enabled.

4.66 Set Continuous Frame Count (SET_CONTINUOUS_FRAMES)

Sets number of frames to read after a trigger pulse is received while “Continuous Acquisition” is enabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC9	Request
2	wValue	2	0x00XX	Number of frames (0-255)
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

ARM-based products return 0 if command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.67 Get Continuous Frame Count (GET_CONTINUOUS_FRAMES)

Reads number of frames to read after a trigger pulse is received when in “Continuous Acquisition” mode.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xCD	Request
2	wValue	2	0xXXXX	n/a
4	wIndex	2	0xXXXX	n/a
6	wLength	2	0	Payload size

Response

Returns number of frames as one byte (0-255).

4.68 Get Battery state (GET_BATTERY_STATE)

Gets the battery percentage from the gas gauge. This command is only supported on SiG spectrometers with internal batteries.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x13	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Three bytes:

- Byte 0: fractional battery charge level (divide by 256 to get value from 0.000 to 0.996 — this is not the complete charge percentage)
- Byte 1: integral battery charge level (valid values 0-100)
- Byte 2: charging state (0 if falling e.g. discharging, non-zero if rising e.g. charging)

For instance, the response array [0x12, 0x34, 0x01] would indicate the battery was at 52.07% (0x34 + 0x12/256) and that the battery was currently charging.

4.69 Get battery register (GET_BATTERY_REG)

Gets any of the available registers in the gas gauge per the gas gauge datasheet (https://datasheets.maximintegrated.com/en/ds/MAX17055.pdf). For instance, if you wanted to read the battery's Design Capacity, you would set wIndex to 0x0018.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Second tier command
2	wValue	2	0x14	Value
4	wIndex	2	0xXXXX	Register to be read (big-endian)
6	wLength	2	0	Payload size

Response

Two bytes. Interpretation depending on register.

4.70 Get CCD Start Line (GET_CCD_START_LINE)

Gets the first horizontal line (detector row) used in the vertical binning process of the sensor. Any rows above this (with a lower index, considering row 0 to be the top of the detector) will not be vertically binned into the output spectrum.

This command is only available on SiG spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0xF2	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 16-bit Unsigned, 0-Based line number that the spectrum binning starts at.

4.71 Get CCD Stop Line (GET_CCD_STOP_LINE)

Gets the line where the binning process stops for the sensor. This line is not included in the binning process. Lines at this index, and numerically higher index values, will not be vertically binned into the output spectrum.

This command is only available on SiG spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0xF3	Value
4	wIndex	2	0xXXXX	Index
6	wLength	2	0	Payload size

Response

Returns 16-bit Unsigned, 0-Based line number that the spectrum binning ends at.

4.72 Set CCD Start Line (SET_CCD_START_LINE)

Gets the first line used in the binning process of the sensor, where 0 is considered the first horizontal row at the top of the detector.

This command is only available on SiG spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0xF4	Value
4	wIndex	2	0xXXXX	16-Bit Unsigned, 0-Based Line Number
6	wLength	2	0	Payload size

Response

Returns 0 if command was successful, non-zero if unsuccessful.

4.73 Set CCD Stop Line (SET_CCD_STOP_LINE)

Sets the line where the binning process stops for the sensor. This line is not included in the binning process.

This command is only available on SiG spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0xF5	Value
4	wIndex	2	0xXXXX	16-Bit Unsigned, 0-Based Line Number
6	wLength	2	0	Payload size

Response

Returns 0 if command was successful, non-zero if unsuccessful.

5 Acquisition Workflows

Working source code examples of most USB commands can be found for a number of common programming languages:

- Python: https://github.com/WasatchPhotonics/Python-USB-WP-Raman-Examples (single-command unit-tests)
- Python: https://github.com/WasatchPhotonics/Wasatch.PY (application-level driver and demo)
- C#: https://github.com/WasatchPhotonics/Wasatch.NET (application-level driver and demo)
- Delphi: https://github.com/WasatchPhotonics/Wasatch.Delphi (application demo)
- LabVIEW: https://github.com/WasatchPhotonics/Wasatch.LV (application demo)
- MATLAB: https://github.com/WasatchPhotonics/Wasatch.MATLAB (application demo)

In addition, detailed engineering-level walkthroughs of the all-important spectral acquisition, laser control and external triggering procedures are discussed below.

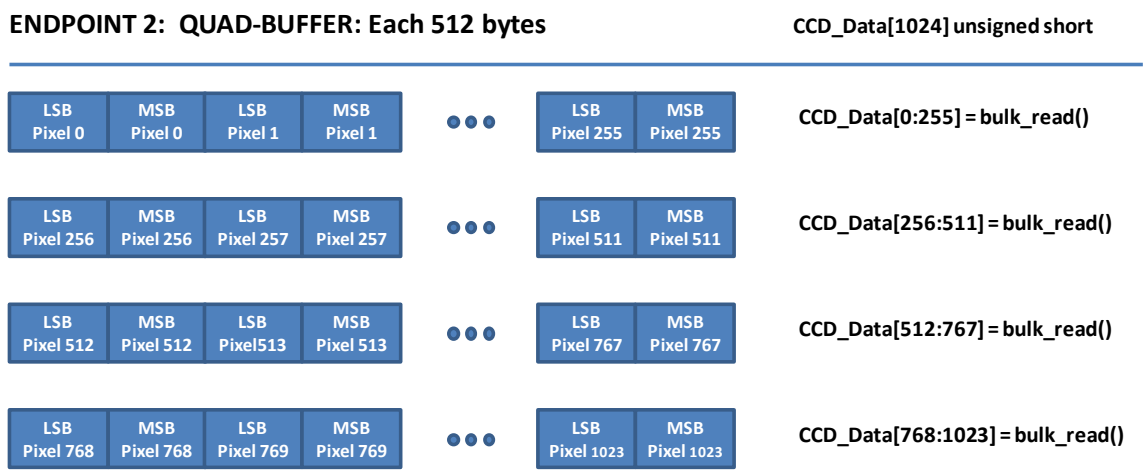
5.1 Spectral Acquisition

Data acquisition can be triggered by a USB command, “Acquire Image” for all product configurations. Data acquisition can also be started by an external trigger event (positive TTL transition). The external trigger is injected on pin 2 of USB Board connector J6.

When properly configured, either the USB command or an external trigger event causes an integration to occur and the resultant data to appear. The data for standard 1024-pixel spectrometers appears on **Endpoint 2** as 1024 words (LSB first). NIR spectrometers may have fewer than 1024 pixels, and other spectrometers may have more than 1024 pixels. On spectrometers using linear-array detectors with more than 1024 pixels, the pixels above zero-indexed 1023 will appear on **Endpoint 6**. (Spectrometers with 2D image sensors such as the SiG output all pixels on **Endpoint 2**.)

Use the GET_LINE_LENGTH command to confirm the number of pixels on your detector; all currently shipping spectrometers use a 16-bit unsigned integer to store intensity, so the number of bytes to read will be 2 * line_length.

Whatever number of bytes are to be read from a given endpoint, the spectrometer supports reading them either in one large read or a series of smaller reads. For instance, working applications have been deployed which perform a bulk-read of 2048 bytes, or a series of four 512-byte blocks. The following diagram shows four sequential reads of 512-bytes each.



5.2 Software-commanded USB Acquisition (internal laser @ 100% power)

A simple USB-command initiated capture event command workflow with manual control of the output trigger (laser) would typically follow these steps (these assume you've found and gotten a handle to an open USB device – see Establishing connection through libusb-win32 drivers):

In order to disable modulating the trigger "output" signal, the following two commands should be executed:

1. SET_LINK_LASER_MOD_TO_INTEGRATION_TIME = 0
2. SET_LASER_MOD = 0

The following will set up the detector for a USB-based acquisition with a user-defined integration time (in milliseconds) on a spectrometer with 1024 pixels:

3. SET_INTEGRATION_TIME = 100 // or other value in ms
4. SET_TRIGGER_SOURCE = 0 // USB command initiated trigger

To set external trigger signal "high" (enable external laser) and initiate a USB-based acquisition:

5. SET_LASER_ENABLE = 1
6. ACQUIRE_SPECTRUM
7. Either
    a. sleep(100) // wait for the integration time in milliseconds, or
    b. while POLL_DATA == 0: sleep(1)
8. Either
    a. perform four bulk reads on Endpoint 2 of 512 bytes each, or
    b. perform a single read of 2048 bytes
9. SET_LASER_ENABLE = 0 // if measurement is complete (no scan averaging etc)
10. demarshall received buffers into 1024 16-bit words (LSB first)

5.3 Software-commanded USB Acquisition (external laser @ 50% power)

A simple USB-command initiated capture event command workflow with user-defined parameter control of the output trigger (laser) would typically follow these steps (these assume you've found and gotten a handle to an open USB device – see Establishing connection through libusb-win32 drivers):

In order to enable modulating the trigger "out" signal, the following three commands should be executed:

1. LINK_LASER_MOD_TO_INTEGRATION_TIME = 1
2. SET_LASER_MOD = 1
3. SET_LASER_ENABLE = 1

The following will set up the detector for a USB-based acquisition with a user-defined integration time (in milliseconds):

4. SET_INTEGRATION_TIME = 100 // or other time in ms
5. SET_TRIGGER_SOURCE = 0 // USB command initiated trigger

SPECIFIC EXAMPLE: To initiate a 5ms pulse at 50% power starting 1.5ms after receiving a USB-initiated acquisition, set the following values:

6. SET_LASER_MODULATION_PULSE_DELAY = 1500 // 1.5ms in µs
7. SET_LASER_MODULATION_PULSE_WIDTH = 2500 // 50% of PULSE_PERIOD

8. SET_LASER_MODULATION_PULSE_PERIOD = 5000
9. SET_LASER_MODULATION_PULSE_DURATION = 5000 // For single pulse, PERIOD=DURATION

Resume normal acquisition processing:

10. ACQUIRE_SPECTRUM
11. Either
    a. sleep(100) // wait for the integration time in milliseconds, or
    b. while POLL_DATA == 0: sleep(1)
12. Either
    c. perform four bulk reads on Endpoint 2 of 512 bytes each, or
    d. perform a single read of 2048 bytes
13. SET_LASER_ENABLE = 0 // if measurement is complete (no scan averaging etc)
14. demarshall received buffers into 1024 16-bit words (LSB first)

5.4 Externally Triggered Acquisition

Triggering is supported on Wasatch Photonics ARM spectrometers. See ENG-0085 for details on the "OEM Connector" on the ARM USB Board (PN 110378). In short, a 3.3V (LVTTL) signal may be applied to pin 10 of connector J8, using pins 5, 9 or 15 for GND.

An externally triggered capture event would typically follow these steps (these assume you've found and gotten a handle to an open USB device – See Establishing connection through libusb-win32 drivers):

To ensure the laser won't fire until the triggered acquisition begins (i.e., the laser doesn't sit there firing for minutes or hours until the trigger signal arrives), "link" the laser modulation to the integration time:

1. LINK_LASER_MOD_TO_INTEGRATION_TIME = 1
2. SET_LASER_MOD = 1
3. SET_LASER_ENABLE = 1 // per linkage, shouldn't actually fire until "get_spectrum"

The following will configure the detector for an externally triggered acquisition:

4. SET_TRIGGER_SOURCE = 1 // external triggering

To set up pulsed output operation (external laser), it is important to understand how integration time and external out trigger (aka "ext_light_source_enable") are related. The external trigger settings have priority over any integration time setting. That is, when an input trigger is sensed by the instrument, the instrument will delay PULSE_DELAY microseconds (minimum value = 1000 microseconds), turn on the laser for PERIOD=DURATION microseconds and then turn off the laser, while the integration time (if shorter) will be extended to match the output pulse of the laser.

Therefore, if the laser integration time is 4 ms, but the output pulse is 5000 µs long, then the currently executing integration time window will be extended to ensure that the laser pulse falls completely within a single integration time window. This means that it is advisable to set the integration time < output pulse-width so that minimal superfluous integration will occur.

5.4.1 SPECIFIC EXAMPLE

To initiate a 5 ms pulse, starting 1.5 ms after receiving an externally triggered acquisition, set the following values:

1. SET_INTEGRATION_TIME = 4 // (ms) Integration time < pulse delay + period.

2. SET_LASER_MODULATION_PULSE_DELAY = 1500 // (µs)
3. SET_LASER_MODULATION_PULSE_WIDTH = 5000 // (µs) (full power)
4. SET_LASER_MODULATION_PULSE_PERIOD = 5000 // (µs)
5. SET_LASER_MODULATION_PULSE_DURATION = 5000 // (µs) (single pulse)

Then, it is advisable to check the current frame number. After *n* "known external trigger events" have occurred, one can check this to verify that the spectrometer correctly acquired exactly *n* spectra.

6. n = GET_ACTUAL_FRAMES

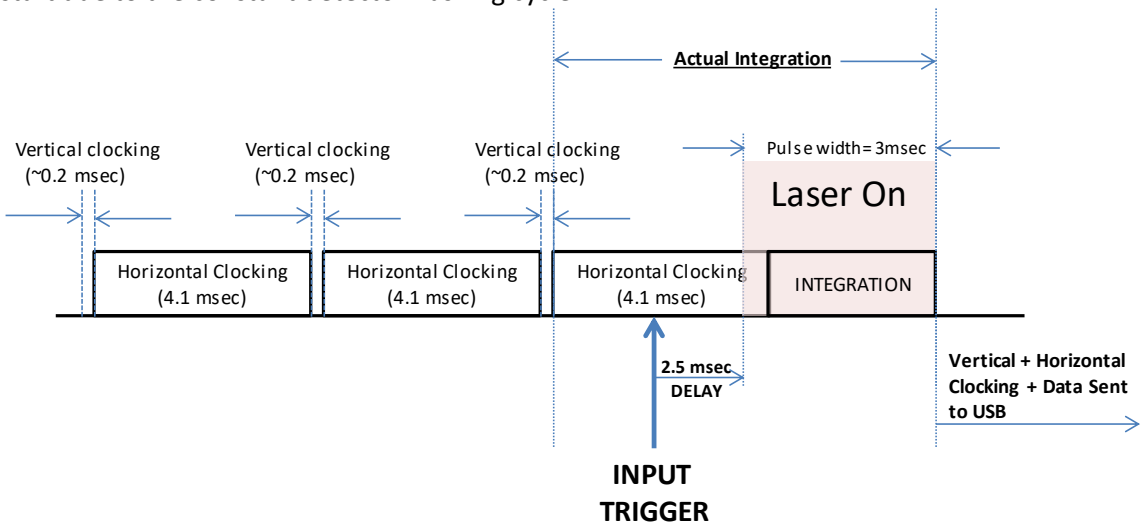
The acquisition state machine should proceed as follows:

1. POLL_DATA continuously or at least 4X-5X the minimum input inter-pulse period (the expected period between incoming external trigger signals). If POLL_DATA > 0 then data is sitting in USB buffer Endpoint 2.
2. Four bulk reads on Endpoint 2 of 512 bytes each (or one read of 2048 bytes)
3. Assemble (left to right) into 1024 16-bit words (LSB first)

6 Detector Timing and External Laser Triggering

The firmware allows a periodic signal within the laser "duration" window. That is, modulation width and modulation period can be set (where width <= period) smaller than duration, and multiple pulses will "fit" inside the "modulation duration" window.

While one can always set an integration window width, one can't control when an integration will truly start due to the constant detector flushing cycle.



In the example above, the input trigger occurred during a horizontal clocking cycle. Since the vertical pixels are untouched, this can be counted as integration. In the above scenario, the firmware latches the input trigger, delays 2.5 ms, turns on the laser for 3 ms. So the actual integration would be ~7 ms, even if the user said the integration was to last < 7 ms. The above clocking scenario can handle up to 100 frames/sec. A single laser pulse is achieved by setting duration = period. More details on external triggering control are below.

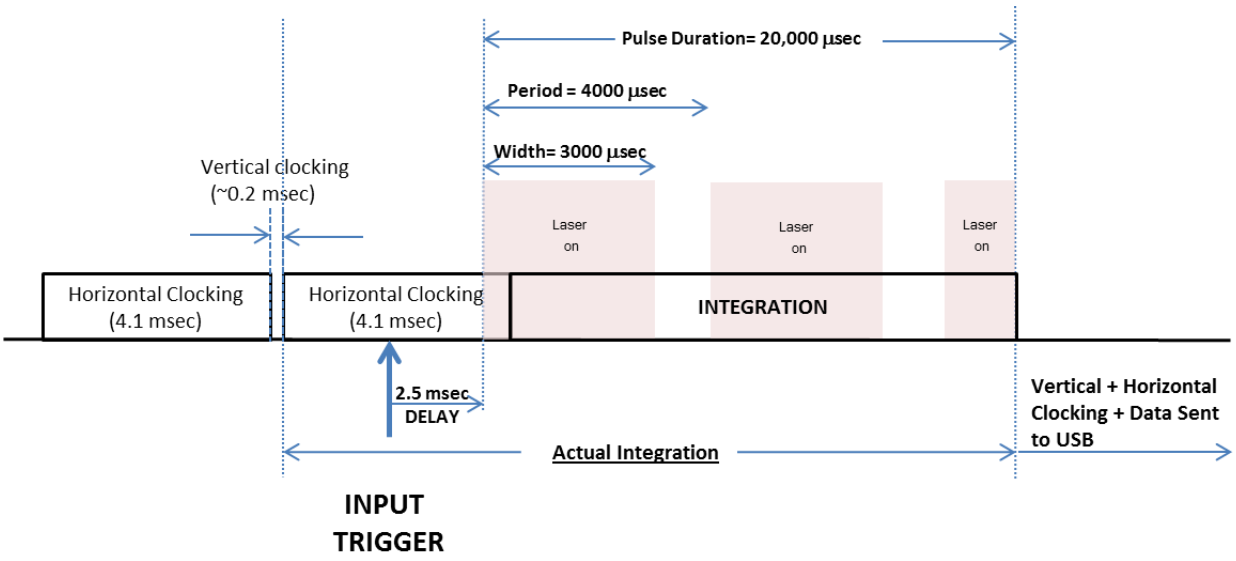
Stated differently, Wasatch spectrometers internally run a constant "free-running mode," and just throw away most of the spectra they collect. A conceptual difference between Wasatch electronics and some commercial alternatives is that some spectrometer designs cache the result of every spectrum,

and when "get_spectrum" is called, return the last-collected cache (if one exists), then deletes the cache (so the same spectrum can't be read twice).

In contrast, Wasatch spectrometers never return an "old" spectrum *completed* before the "acquire spectrum" command is received, but they may — as in this example — return a spectrum whose integration had already *started* before the get_spectrum command was received.

The following section details the parameters related to laser (external) triggering. The timing of the external triggering is absolute with respect to an input trigger. The system with microsecond accuracy will output a pulse based on the rising edge of an input trigger with characteristics defined by duration, period, width and delay.

The diagram below illustrates the concept:



Four parameters define the above external trigger (laser) behavior. All parameters are independent of integration time and are enabled using the `SET_LASER_MOD = 1` and `SET_LASER_ENABLE = 1` commands illustrated in the example in Section 4.3.

Pulse Delay: Defines the delay, in microseconds, from the input trigger, after which the leading edge of the output trigger will begin.

Pulse Period: After setting the pulse delay, the pulse period "clock" is enabled, which establishes the full period (in microseconds) of an ongoing series or train of laser pulses. If laser modulation is enabled, then the laser will be firing for some percentage of each period (as much as 100%), and disabled for the remainder of the period — this is controlled via pulse width, below.

Pulse Width: The pulse width (on-time) of the pulse period cycle in microseconds. Note that a pulse period is defined by an *on-time* first, then *off-time* cycle. Pulse width is used to control the output laser power. If the pulse width == pulse period, the laser is operating at full power (equivalent to disabling laser modulation). If the pulse width is less than the pulse period, then the output laser power will equal the fraction (width / period). Behavior when the width exceeds the period is undefined.

Pulse Duration: The total pulse cycle duration. The laser will be shut off at time defined by duration (measured from the end of the delay time) whatever phase the laser pulse cycle is currently in. Therefore, the “last” pulse in a pulse train can be shortened below the defined pulse width.

As described in the first “single pulse” example above, the duration and period should be set equal to each other.