ENG-0001 USB FID ICD

Interface Control Document (ICD) for Feature Identification Device (FID) USB Spectrometers

Revision 18

Apr 16, 2026

Contents

Revision Log 6
1 General Description 8
2 USB Connection to device 9
2.1 USB device drivers 10
2.2 libusb-win32 drivers 11
2.2.1 C Example 11
3 Command Matrix 14
3.1 Command Table 16
3.2 Developmental / R&D Opcodes 21
3.3 Deprecated / Reserved Opcodes 23
4 Command Detail 26
4.1 Metadata 27
4.1.1 Get Microcontroller Firmware Version (GET_FIRMWARE_VERSION) 27
4.1.2 Get Microcontroller Serial Number (GET_MICROCONTROLLER_SERIAL_NUMBER, GET_CPU_UNIQUE_ID, etc) 27
4.1.3 Get Bluetooth Firmware Revision (GET_BLE_FIRMWARE_VERSION) 28
4.1.4 Get FPGA Firmware Revision (GET_FPGA_FIRMWARE_VERSION) 28
4.1.5 Get FPGA Compilation Options (READ_COMPILATION_OPTIONS) 28
4.2 EEPROM Control 30
4.2.1 Get Model Info (GET_MODEL_CONFIG) 30
4.2.2 Set Model Info (SET_MODEL_CONFIG) 30
4.3 Spectral Acquisition and Frame Header 32
4.3.1 Acquire Spectrum (ACQUIRE) 32
4.3.2 Acquire Auto-Raman (ACQUIRE_AUTO_RAMAN) 33
Parameter Block Structure 33
4.3.3 Get Auto-Raman Params (GET_AUTO_RAMAN_PARAMS) 35
4.3.4 Get Auto-Raman Status (GET_AUTO_RAMAN_STATUS) 35
4.3.5 Set Frame Header (SET_FRAME_HEADER) 36
4.3.6 Get Frame Header (GET_FRAME_HEADER) 38
4.3.7 Get Image Sensor State (GET_IMAGE_SENSOR_STATE) 38
4.3.8 POLL_DATA / ACQUISITION_STATUS 39
4.4 Integration Time Control 40
4.4.1 Set Integration Time (SET_INTEGRATION_TIME) 40
4.4.2 Get Integration Time (GET_INTEGRATION_TIME) 40
4.5 Detector Gain and Offset Control 43

4.5.1 Set Detector Offset (SET_DETECTOR_OFFSET) 43
4.5.2 Set Detector Gain (SET_DETECTOR_GAIN) 43
4.5.3 Get Detector Offset (GET_DETECTOR_OFFSET) 45
4.5.4 Get Detector Gain (GET_DETECTOR_GAIN) 45
4.6 Scan Averaging 46
4.6.1 Set Scan Averaging (SET_SCANS_TO_AVERAGE) 46
4.6.2 Get Scan Averaging (GET_SCANS_TO_AVERAGE) 46
4.7 Laser Control 47
4.7.1 Laser Interlock Overview 47
FX2-based Spectrometers 47
XS Spectrometers 47
4.7.2 Set Laser Enable (SET_LASER_ENABLE) 48
4.7.3 Get Laser Enable (GET_LASER_ENABLE) 48
4.7.4 Get Laser Temperature (GET_LASER_TEMPERATURE, aka GET_ADC) 49
4.7.5 Set Laser TEC Setpoint (SET_LASER_TEC_SETPOINT) 50
4.7.6 Get Laser TEC Setpoint (GET_LASER_TEC_SETPOINT) 50
4.7.7 Set Laser TEC Mode (SET_LASER_TEC_MODE) 51
4.7.8 Get Laser TEC Mode (GET_LASER_TEC_MODE) 51
4.7.9 Get Laser Interlock (GET_LASER_INTERLOCK, aka CAN_LASER_FIRE) 52
4.7.10 Get Laser Is Firing (GET_LASER_IS_FIRING) 52
4.7.11 Set Laser Power Attenuator (SET_LASER_POWER_ATTENUATOR) 53
4.7.12 Get Laser Power Attenuator (GET_LASER_POWER_ATTENUATOR) 53
4.7.13 Set Laser Warning Delay (SET_LASER_WARNING_DELAY_SEC) 54
4.7.14 Get Laser Warning Delay (GET_LASER_WARNING_DELAY_SEC) 54
4.7.15 Set Laser Watchdog (SET_LASER_WATCHDOG) 54
4.7.16 Get Laser Watchdog (GET_LASER_WATCHDOG) 55
4.7.17 Set Laser TEC Watchdog (SET_LASER_TEC_WATCHDOG) 55
4.8 Modulation Control 57
4.8.1 Set Modulation Pulse Period (SET_MOD_PULSE_PERIOD) 57
4.8.2 Get Modulation Pulse Period (GET_MOD_PULSE_PERIOD) 58
4.8.3 Set Modulation Pulse Width (SET_MOD_PULSE_WIDTH) 58
4.8.4 Get Modulation Pulse Width (GET_MOD_PULSE_WIDTH) 59
4.8.5 Set Modulation Enable (SET_MOD_ENABLE) 59
4.8.6 Get Modulation Enable (GET_MOD_ENABLE) 59
4.8.7 Set Modulation Linked to Integration (SET_MOD_LINKED_TO_INTEGRATION) 60
4.8.8 Get Modulation Linked to Integration Time (GET_MOD_LINKED_TO_INTEGRATION) 61
4.9 Detector Temperature Control 62

4.9.1 Set Detector Thermo-Electric Cooler Enable (SET_DETECTOR_TEC_ENABLE) 62
4.9.2 Get Detector TEC Enable (GET_DETECTOR_TEC_ENABLE) 62
4.9.3 Get Detector Temperature (GET_DETECTOR_TEMPERATURE) 62
4.9.4 Set Detector TEC Setpoint / Set DAC (SET_DETECTOR_TEC_SETPOINT) 63
4.9.5 Get Detector TEC Setpoint / Get DAC (GET_DETECTOR_TEC_SETPOINT, aka GET_DAC) 64
4.9.6 Select ADC (SET_SELECTED_ADC) 64
4.9.7 Get Selected ADC (GET_SELECTED_ADC) 65
4.10 Trigger Control 66
4.10.1 Set Trigger Source (SET_TRIGGER_SOURCE) 66
4.10.2 Get Trigger Source (GET_TRIGGER_SOURCE) 66
4.10.3 Set Trigger Delay (SET_TRIGGER_DELAY) 67
4.10.4 Get Trigger Delay (GET_TRIGGER_DELAY) 67
4.11 High-Gain Mode 68
4.11.1 Set High-Gain Mode Enable (SET_HIGH_GAIN_MODE_ENABLE) 68
4.11.2 Get High Gain Mode Enable (GET_HIGH_GAIN_MODE_ENABLE) 68
4.12 Battery Control 69
4.12.1 Get Battery state (GET_BATTERY_STATE) 69
4.12.2 Set Battery Charger Enable (SET_BATTERY_CHARGER_ENABLE) 69
4.13 Area Scan and Detector ROI 70
4.13.1 Set Area Scan Enable (SET_AREA_SCAN_ENABLE) 70
4.13.2 Set Detector Start Line (SET_DETECTOR_START_LINE) 70
4.13.3 Get Detector Start Line (GET_DETECTOR_START_LINE) 71
4.13.4 Set Detector Stop Line (SET_DETECTOR_STOP_LINE) 71
4.13.5 Get Detector Stop Line (GET_DETECTOR_STOP_LINE) 71
4.14 Ambient Temperature 73
4.14.1 Get Ambient Temperature ARM (GET_AMBIENT_TEMPERATURE_ARM) 73
4.14.2 Get Ambient Temperature LM57B (GET_AMBIENT_TEMPERATURE_LM57B) 73
4.15 Board State 75
4.15.1 Get Ambient Temperature ARM (GET_AMBIENT_TEMPERATURE_ARM) 75
4.15.2 Set DFU Mode (SET_DFU_MODE) 75
4.15.3 Get USB Adapter Info (GET_USB_ADAPTER_INFO, GET_POWER_CONNECTION_STATE) 75
4.15.4 Reset Field-Programmable Gate Array (RESET_FPGA) 76
4.16 Power Management 77
4.16.1 Power Off (VR_POWER_OFF) 77
4.16.2 Reboot Device (RESET_UNIT) 77

4.16.3 Set Detector Timeout (SET_DETECTOR_TIMEOUT_SEC) 77
4.16.4 Get Detector Timeout (GET_DETECTOR_TIMEOUT_SEC) 78
4.16.5 Set Power Watchdog (SET_POWER_WATCHDOG_SEC) 78
4.16.6 Get Power Watchdog (GET_POWER_WATCHDOG_SEC) 79
5 Acquisition Workflow 80
5.1 Spectral Acquisition 81
5.2 Software-commanded USB Acquisition (internal laser @ 100% power) 82
5.3 Software-commanded USB Acquisition (external laser @ 50% power) 83
5.3.1 SPECIFIC EXAMPLE 84
6 Detector Timing and External Laser Triggering 85
6.1 External Triggering 87
FX2-based Spectrometers (110008 PCB) 87
Appendix: Proposed Changes 88
Appendix: CRC-8-CCITT Specification 89
CRC Calculation 89

Wasatch Photonics

Revision Log

Date	Author	Description	Rev.
9/25/14	S. Trani	Initial Release	0
8/11/15	J. Dreitzler	Add second tier commands, model configuration and feature identification	1
10/6/15	J. Dreitzler	Corrected model configuration format Multiple changes to align commands for FX2 and ARM-based products	2
10/20/15	J. Dreitzler	Added additional commands	3
1/25/17	J. Traud	Formatting, cleanup, and preparation for public release.	4
10/2/17	J. Traud	Removed customer specific references	5
10/8/17	J. Traud	Removed legacy and development command references no longer valid in current firmware branch	6
8/24/18	M. Zieg	Update to reflect current firmware	7
10/18/19	M. Zieg R. Dickerson	Added battery commands Added Get/Set CCD Start/Stop Line	8
Mar 2020	M. Zieg	Added Raman Mode, Raman Delay and Laser Watchdog	9
Feb 3, 2021	M. Zieg B. Williams	Added Area Scan, Accessory Connector, Even/Odd Gain/Offset	10
2/15/2021	M. Zieg	Refactored section 4 (commands) into structured categories	11
2/25/2021	M. Zieg B. Williams	Added Lamp Enable, Ambient Temperature, Fan Control; updated Modulation Linked to Integration, Modulation Pulse Delay	12
3/3/2021	M. Zieg	Updated GET_MOD_PERIOD endianness, refactored Modulation section	13
4/14/2021	M. Zieg N. Baron	Added untethered opcodes	14
11/2/2021	N. Baron E. Dort M. Zieg B. Williams J. Wach L. Brady	Added opcodes for Untethered Capture Status; updated Gen 1.5; added Detector ROI, Pixel Mode; Laser Interlock	15

11/4/2024	M. Zieg M. Malladi R. Krishnan J. Klein T. Stohrer	● added: ○ ACQUIRE_AUTO_RAMAN ○ AMBIENT_TEMPERATURE_ARM ○ BATTERY_CHARGER_ENABLE ○ DETECTOR_TIMEOUT ○ FRAME_HEADER ○ LASER_POWER_ATTENUATOR ○ LASER_TEC_MODE ○ LASER_WARNING_DELAY ○ POWER_WATCHDOG ○ R&D/Developmental opcodes ● updated docs: ○ FPGA_COMPILATION_OPTIONS ○ LASER_TEC_SETPOINT ● Rev 17 API previews for: ○ SCANS_TO_AVERAGE ○ POWER_SAVINGS_MODE	16
11/13/2024	V. Hannak M. Zieg	● updated SPECTRUM_HEADER format ● added Auto-Raman param ranges	17
4/16/2026	M. Zieg R. Krishnan	● added CPU_UNIQUE_ID format ● added GET_IMAGE_SENSOR_STATE, GET_USB_ADAPTER_INFO, GET_AUTO_RAMAN_PARAMS, GET_BLE_FIRMWARE_VERSION, RESET_UNIT ● changed CRC8 from Maxim to CCITT ● removed ~23pp into ENG-0001 Deprecated Annex ● added GET/SET_LASER_TEC_WATCHDOG_SEC	18

1 General Description

This document describes the USB (Universal Serial Bus) API (Application Programming Interface) for all USB-based X, XM, XC, XS series Wasatch Photonics spectrometers for Raman (248, 532, 633, 638, 785, 830 and 1064) and non-Raman (UVVIS, VIS, VISNIR, NIR), whether benchtop or handheld form-factor.

This document *does not* apply to:

- spectrometers intended for the OCT market such as the Cobra series
- the Bluetooth API of BLE-capable XS series spectrometers (see ENG-0120)
- XL-series spectrometers using Andor cameras

The USB data interface for Wasatch spectrometers is USB 2.0 compliant. FX2-based spectrometers using the 110008 or 220090 USB boards operate at High Speed (480Mbps), while ARM-based spectrometers currently operate at Full Speed (12Mbps).

Control commands are sent on Endpoint 0 (EP0) as Vendor Requests (hence the "VR_" suffix on some commands). 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 Endpoint 6 for 2048-pixel detectors on FX2).

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 one of three supported values representing Feature Identification Device (**FID**) models. PIDs used for feature identification include:

- `0x1000` (FX2 with Hamamatsu silicon detector)
- `0x2000` (FX2 with Hamamatsu InGaAs detector)
- `0x4000` (ARM-based spectrometer, e.g. XS)

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 the Wasatch Photonics USB hardware/firmware API. (For Windows, you would require a custom .inf file to indicate a driver other than libusb-win32.)

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 "libusbK 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), Windows 10 (x86/x64) and Windows 11 (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 lower-level USB drivers, and automate common operations by "wrapping" complex multi-step procedures into simple function calls (hiding the marshaling/demarshalling of arguments, endian issues etc).

If you are interested in developing your spectroscopy application against our high-level drivers (rather than using the low-level USB API defined in this document), please see:

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

2.2 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:

- C/C++: https://github.com/WasatchPhotonics/Wasatch.VCPP
- Python: https://github.com/WasatchPhotonics/Wasatch.PY
- C#: 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;
}
```

On POSIX, 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,
    uint8_t bmRequestType,
    uint16_t bRequest,
    uint16_t wValue,
    uint16_t wIndex,
    char *data,
    int size,
    int timeout);
```

Following are explanations of each of the parameters in the above call, which are representative of what you will find in any low-level USB driver library, as they map directly to the protocol's standard field names and datatypes for USB Control Packets.

- *dev*: handle returned by usb_open()
- ***bmRequestType***: bitmap, typically 0x40 for host-to-device "commands," or 0xC0 for device-to-host "requests" (queries).
- ***bRequest***: bRequest field in the setup packet. bRequest represents the specific command or 'opcode' being sent to the spectrometer, and most of this document is focused on identifying the different bRequest values and what they do.
- ***wValue***: value field in the setup packet. Often used to send parameters of a command.
- ***wIndex***: index field in the setup packet. Also, often used to send high-order byte information as parameter of command.
- ***data***: up to 64-byte data packet associated with control message. The payload may be used in some "setter" commands (such as laser modulation) 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 *wValue* field, and the middle-significant bytes in the *wIndex* field.
- *size*: size of bytes data packet buffer. Can be zero if a NULL data packet reference is used.
- *timeout*: period in milliseconds before timing-out if an error occurs.

Note that while Wasatch Photonics does not always use type-prefixes in variable names (i.e. "Hungarian Notation"), it is worth understanding the prefixes used in the above parameters:

- bm = "bitmask," indicating one byte encapsulating multiple other sub-byte values (up to 8 individual bits)
- b = "byte," an unsigned integral value 0-255 (2^8-1)
- w = "word," an unsigned integral value 0-65535 (2^{16}-1)

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}; // 8 bytes for ARM
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);
```

A more complete C/C++ example, which can be compiled and tested from both Windows (Visual Studio) and Linux (GCC), can be found in the following open-source project:

https://github.com/WasatchPhotonics/Wasatch.VCPP

3 Command Matrix

In the following table of supported USB commands (or "opcodes"), columns are defined as follows:

- **Getter/Setter**: Most spectrometer features have both "get" and "set" versions. Some features can only be read (e.g. COMPILATION_OPTIONS) while others can only be written (e.g. FPGA_RESET).
- **Data Type**: indicates the 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 when writing "uint40" parameters, used by laser modulation and continuous strobe features, the integral value is split across the USB Control Packet's wValue, wIndex and payload[0] fields in little-endian sequence.
    - For example, the hexadecimal value 0x0123456789 (representing 4886718345µs, or about 1.36hr) would be sent as wValue = 0x6789, wIndex = 0x2345 and payload[0] = 0x01.
    - Likewise, uint24 values (such as used to set integration time in milliseconds) are written with the least-significant word in wValue, and the most-significant byte in wIndex. For example, 0x123456 (decimal 1193046ms, or about 20min) would be sent as wValue 0x3456 and wIndex 0x0012.
  - **bool** indicates only values of 1 or 0 are supported (other values yield undefined behavior)
  - **float16** is a custom floating-point format in which the MSB represents an integral value (0-255) and the LSB is to be divided by 256 to represent a fractional component.
    - For example, the value 0x1234 would be parsed as MSB 0x12 (dec 18), and LSB 0x34 (b0011 0100) as 1/8 + 1/16 + 1/64 = 13/64 = dec 0.203125. Therefore, a detector gain of 0x1234 would cause each pixel's intensity to be scaled by 18.203125 on CCD detectors (or indicate 18.2 dB on CMOS)
    - To be clear, it is *not* an IEEE 754 half-precision float; it is conceptually similar to an unsigned bfloat16.
    - See Set Detector Gain (SET_DETECTOR_GAIN) for details.
  - **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 7-bit ASCII characters). Strings can be optionally null-terminated if less than the length of their allocated field, but are not necessarily null-terminated if of the maximum field length. (I.e., a 15-char serial number should have a trailing '\0' in

the 16th element, but a 16-char serial number can fill the EEPROM field with no terminator.) Any characters following a null are ignored by Wasatch drivers.
  - **enum** indicate the argument is technically a uint8 octet (byte), but see the "Enums" column for interpretations of supported zero-indexed values
  - **void** indicates the opcode does not take or return a primary data value

- **2nd Tier:** So-called "second-tier" commands all share a bRequest of 0xff, and send the "opcode" as the wValue field, with the primary parameter in wIndex. In contrast, normal (non-2nd-tier) commands send the opcode as bRequest, and send the primary parameter in wValue.
- **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; no current spectrometer firmware is believed to exhibit this behavior.)
- **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. (To be clear, we are using endianness to refer to the order of BYTES, not the order of BITS within a byte; octets are transmitted atomically over USB, and will always have big-endian bit order on the host and microcontroller.)
- **Enums**: supported values for the "enum" Data Type.
- **Supports**: some commands only work on models with InGaAs detectors, FX2 or ARM microcontrollers, WP-XS models, untethered (UT) 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.
  - **"Deprecated"** indicates a feature is no longer recommended for use in shipping models and may not be functional. In some cases "deprecated" may mean that we are not actively testing this feature, but that it can be restored to fully supported use given customer interest.
  - **"Developmental"** means a feature is in development, but not yet fully released and may not be functional.

Previously-supported deprecated commands in the following table have been marked **deprecated** and ~~struckout~~.

3.1 Command Table

Table 1 FID Opcodes

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
ACQUIRE_SPECTRUM	0xad		void				8	NA				response read back on bulk endpoint 2 (and 6 for 2048px detectors)
ACQUIRE_AUTO_RAMAN	0xfd		uint8[23]							(XS)		Takes 23-byte payload of configuration options; overlaps SET_PIXEL_MODE
AMBIENT_TEMPERATURE_ARM	0x2a		char		1			NA		(XS)		
AREA_SCAN_ENABLE		0xeb	bool					NA				overloaded with SET_HIGH_GAIN_MODE
AUTO_RAMAN_PARAMS	0x9a		byte[23]		23					(XS)		
BATTERY_CHARGER_ENABLE		0x86	Bool		1					(XS)		
BATTERY_STATE	0x13		mask	Y	3			MSB		(XS)		
BLUETOOTH_FIRMWARE_VERSION	0x2d		char[32]	Y			32			(XS)		
COMPILATION_OPTIONS	0x04			Y	3		8	LSB				
CPU_UNIQUE_ID	0x2c		byte[12]		12					(XS)		
DETECTOR_GAIN	0xc5	0xb7	float16		2			LSB				half-precision float (MSB is integer, LSB is fraction)
DETECTOR_OFFSET	0xc4	0xb6	int16		2			LSB				
DETECTOR_START_LINE	0x22	0x21	uint16	Y	2	2		LSB		(XS)		

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
DETECTOR_STOP_LINE	0x24	0x23	uint16	Y	2	2		LSB		(XS)		
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_TIMEOUT_SEC	0x8f	0x8e	uint16		2			LSB		(XS)		
DFU_MODE		0xfe	void					NA		(ARM)		DFU = “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				
FRAME_HEADER	0x99	0x98	bool		1					(XS)		
HIGH_GAIN_MODE_ENABLE	0xec	0xeb	bool		1			NA		(InGaAs)		overloaded with SET_AREA_SCAN
IMAGE_SENSOR_STATE	0x97		byte		1			NA		(XS)		
INTEGRATION_TIME	0xbf	0xb2	uint24		3			LSB				sent as 32-bit word (LSW wValue, MSW wIndex, big-endian within each)

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
LASER_ENABLE	0xe2	0xbe	bool		1			NA		*	*	Used as STROBE_ENABLE on Gen 1.5 non-Raman
LASER_INTERLOCK	0xef		bool		1			NA			Y	developmental
LASER_IS_FIRING	0x0d		bool	Y	1			NA			Y	developmental
LASER_POWER_ATTENUATOR	0x83	0x82	uint8		1					(XS)	Y	
LASER_TEC_MODE	0x85	0x84	uint8_t		1				OFF, ON, AUTO, AUTO_ON	(XS)	Y	
LASER_TEC_SETPOINT	0xe8	0xe7	uint8/12		1/2			LSB?		(FX2, XS)	Y	110280: value is 7bit 220250: value is 12-bit
LASER_TEC_WATCHDOG	0x7e	0x7d	uint16	Y	2							
LASER_TEMPERATURE (GET_ADC)	0xd5		uint16		2			LSB			?	
LASER_WARNING_DELAY_SEC	0x8b	0x8a	uint8		1					(XS)	Y	
LASER_WATCHDOG	0x17	0x18	uint16	Y	1					(XS)	Y	
MOD_ENABLE	0xe3	0xbd	bool		1		8	NA				Used for laser power on Raman models, and continuous strobe in non-Raman Gen 1.5 models.

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
MOD_LINKED_TO_INTEGRATION	0xde	0xdd	bool		1			NA				Used for laser power on Raman models.
MOD_PULSE_PERIOD	0xcb	0xc7	uint40		5			LSB				
MOD_PULSE_WIDTH	0xdc	0xdb	uint40		5			LSB				
MODEL_CONFIG	0x01	0xa2 or 0x02	byte[]	*	64			MSB				* API differs significantly between FX2 and ARM; see detailed description
POLL_DATA	0xd4		enum		1				(IDLE, DARK, WARMUP, SAMPLE, PROCESSING, STABILIZING, DATA_READY, …, ERROR, UNDEFINED)	(XS)		see PollStatus in Wasatch.PY
POWER_OFF		0x87	Void							(XS)		
POWER_WATCHDOG	0x31	0x30	uint16	Y				LSB		(XS)		
RESET_FPGA		0xb5	void					NA				attempts to perform a runtime reset (power cycle) of the FPGA
RESET_UNIT		0x93	void	N						(XS)		
SCANS_TO_AVERAGE	0x63	0x62	uint16	Y				MSB		(XS)		
SELECTED_ADC	0xee	0xed	enum		1			NA	(PRIMARY, SECONDARY)			Typically laser thermistor or photodiode if present

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
TRIGGER_DELAY	0xab	0xaa	uint24		3			LSB		(ARM)		Delay is in 0.5us, supports 24-bit unsigned value (about 8.3sec max)
TRIGGER_SOURCE	0xd3	0xd2	enum		1			NA	(USB, EXTERNAL)			
USB_ADAPTER_INFO	0x78		byte[5]	N	2			NA		(XS)		

3.2 Developmental / R&D Opcodes

The following engineering commands are being documented for completeness, and to avoid opcode overloading. They aren't included in the above table because they aren't considered user-facing for most customer applications, and may change rapidly with engineering needs.

In the following, BLE refers to the XS BLE microcontroller (e.g. Ezurio BL652) and STM refers to the XS STM32 USB microcontroller (e.g. STM32H7).

Datatypes

- All uint32 getters are assumed to be little-endian unless stated otherwise.
- Char[n] fields are null-terminated if length is < n

Label	Code	2nd Tier	Datatype	Notes
GET_BLE_INTF_MSG_RX_CNT	0x38	Y	uint32	# of UART messages from BLE → STM
GET_BLE_INTF_MSG_TX_CNT	0x39	Y	uint32	# of UART messages from STM → BLE
GET_BLE_INTF_KA_REQ_TX_CNT	0x40	Y	uint32	# of keepalive messages from STM → BLE
GET_BLE_INTF_KA_RESP_RX_CNT	0x41	Y	uint32	# of keepalive messages from BLE → STM
GET_BLE_INTF_BATT_INFO_REQ_RX_CNT	0x42	Y	uint32	# of battery status requests from BLE → STM
GET_BLE_INTF_BATT_INFO_TX_CNT	0x43	Y	uint32	# of battery status responses from STM → BLE
GET_BLE_INTF_UART_RESET_CNT	0x44	Y	uint32	???
GET_BLE_INTF_TOTAL_KEEP_ALIVE_TMO_CNT	0x45	Y	uint32	???
GET_BLE_INTF_B2B_KEEP_ALIVE_TMO_CNT	0x46	Y	uint32	???
GET_BLE_INTF_FPGA_REG_RD_REQ_RX_CNT	0x47	Y	uint32	# of FPGA I2C pass-through read requests from BLE → STM
GET_BLE_INTF_FPGA_REG_WR_REQ_RX_CNT	0x48	Y	uint32	# of FPGA I2C pass-through write requests from BLE → STM

GET_BLE_INTF_FPGA_REG_DATA_TX_CNT	0x49	Y	uint32	???
GET_BLE_INTF_EEPROM_PAGE_RD_REQ_RX_CNT	0x4a	Y	uint32	# of EEPROM read requests from BLE → STM
GET_BLE_INTF_EEPROM_PAGE_DATA_TX_CNT	0x4b	Y	uint32	# of EEPROM write responses from STM → BLE
GET_BLE_INTF_BLE_FW_INFO_RX_CNT	0x4c	Y	uint32	# of 0x2d
GET_BLE_INTF_BLE_PART_NR_INFO_RX_CNT	0x4d	Y	uint32	# of 0x2e
GET_BLE_INTF_RADIO_STATE_UPD_RX_CNT	0x4e	Y	uint32	# of 0x2f
GET_BLE_PART_NR_INFO	0x2e	Y	char[32]	Report BLE IC part number ("BL652-SA-01-T/R")
GET_BLE_RADIO_STATE	0x2f	Y	uint8	Report BLE radio state (INACTIVE 0, ADVERTISING 1, PAIRED 2)
I2C_FPGA_PEEK	0x91	N	byte[]	reads configurable number of bytes from FPGA register (wValue addr)
I2C_FPGA_POKE	0x90	N	byte[]	writes configurable number of bytes to FPGA register (wValue addr)
BLE_POWER_ENABLE	0x88	N	bool	dis/enable BLE
GET_FW_PNS	0x89	N	?	??? we already have opcodes for this ???
SET_TEST_PATTERN	0xba	N	?	?
GET_TEST_PATTERN	0xf1	N	?	?
CONTROL_CDS	0xbb	N	?	maybe used on Hamamatsu?
GET_BATTERY_MILLIVOLTS	0x2b	Y	?	?
SC_GET_SYSTEM_CORE_CLOCK_RATE	0x70	Y	?	
SET_IMG_SNSR_STATE_TRANS_TIMEOUT	0x71	Y	uint16	assuming millisec
GET_IMG_SNSR_STATE_TRANS_TIMEOUT	0x72	Y	uint16	millisec, **little-endian**
SET_BLE_ADVERTISING_POWER_OFFSET_DBM	0x9c		uint8	Apply offset to BLE advertising power in dBm (supported values -40, -20, -16, -12, -8, -4, 0, 3, 4). Not currently used, added for internal testing only (default value 0).

3.3 Deprecated / Reserved Opcodes

The following are not actively used or supported, but are documented so we don't accidentally re-use the value elsewhere and complicate API history by letting the same command do different things on different firmware builds / models.

With regard to deprecated and currently unsupported features, if we do decide to resurrect similar capabilities in the future, we should probably re-use the previously-allocated command for that purpose, rather than allocate new commands which do the same thing.

Label	Code	2nd Tier	Datatype	Notes
READ_FPGA_CONFIG	0xa1	N	?	Maybe Hamamatsu?
READ_SERIAL_NUMBER	0xa3	N	?	deprecated by EEPROM
READ_ARM_CONFIG	0xa4	N	?	Early ARM dev?
NEW_PROTOCOL_MESSAGE	0xa5	N	?	Apparently never used…
GET_CHIP_REV	0xa6	N	?	?
GET_FPGA_STATUS	0xa7	N	?	?
PING	0xa8	N	?	keepalive?
SET_PASS_FAIL_LED	0xa9	N	?	unreleased product?
CLEAR_ACQUIRE_BUTTON_PRESSED	0xac	N	?	unreleased product?
SET_CRC_DATA	0xae	N	?	?
GET_CRC_DATA	0xaf	N	?	?
SET_RESET	0xb0	N	?	?
CLEAR_RESET	0xb1	N	?	?
SET_FPGA_CONFIG	0xb3	N	?	?

PROG_FPGA_PROM	0xc1	N	?	?
XSVF_COMMAND	0xc2	N	?	??? presumably FX2 ???
SET_LASER_RAMP	0xe9	N	?	no plans to resurrect
GET_LASER_RAMP	0xea	N	?	no plans to resurrect
GET_PIXEL_COUNT	0xf0	N	?	deprecated by EEPROM (and by 0xff 0x03)
GET_STATUS	0xe5	N	?	??? what does that mean
GET_EP_STATUS	0xe6	N	?	??? what does that mean
GET_FPGA_CONFIGURATION	0xf2	N	uint16	obviated by READ_FPGA_I2C
ERASE_FPGA_CONFIG	0xf3	N	?	whut
WRITE_FPGA_CONFIG	0xf4	N	?	???
WRITE_CALIBRATION	0xf5	N	uint32	???
WRITE_SERIAL_NUMBER	0xf6	N	?	obviated by WRITE_EEPROM
SAVE_FPGA_CONFIG	0xf7	N	?	obviated by WRITE_EEPROM
ERASE_ARM_CONFIG	0xf8	N	?	obviated by WRITE_EEPROM
WRITE_ARM_CONFIG	0xf9	N	?	obviated by WRITE_EEPROM
GET_LASER_TRANSITION_POINT	0xfa	N	?	deprecated feature
SET_LASER_TRANSITION_POINT	0xfb	N	?	deprecated feature
READ_EEPROM	0xfc	N	?	obviated by GET_MODEL_CONFIG
MICROSCOPE_LED_CONTROL	0x00	Y	?	deprecated feature
GET_ERROR_CODES	0x0e	Y	?	deprecated feature
SET_EXTERNAL_LASER_POWER	0x0f	Y	?	deprecated feature

GET_EXTERNAL_LASER_POWER	0x10	Y	?	deprecated feature
SET_SONY_IMX_REGISTER	0x11	Y	?	deprecated feature
GET_SONY_IMX_REGISTER	0x12	Y	?	deprecated feature
WRITE_LIBRARY	0x28	Y	?	deprecated feature
PROCESS_LIBRARY	0x29	Y	?	deprecated feature

4 Command Detail

Following are the valid commands for the USB interface. All commands are sent as Vendor Requests.

The USB Packet "payload" for "set" commands can generally be a NULL pointer. There are two noteable exceptions to this:

1. For ARM-based spectrometers, all commands require a payload of at least 8 bytes. These can be set to zero, and the value of these bytes does not affect command operation, but the buffer must be present.
2. Some commands which take a bigger numeric parameter than will fit in the combined 32 bits afforded by the combined wValue and wIndex 16-bit fields, will "overflow" into the payload vector. The primary example for this is the uint40 fields used by laser modulation and continuous strobe.

In this section's tables, note the following:

- **bmRequestType**: "bm" stands for "bitmask", 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 (0x2 = Vendor)
  Bits 4-0: Recipient (0x0 = Device)

  All bmRequestType values in our ICD will be either 0x40 (setters, i.e. commands from the host) or 0xC0 (getters, i.e. requests from the host)

- **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 Metadata

These commands help you find out about the spectrometer to which you're connected, including available features, options, installed firmware etc.

4.1.1 Get Microcontroller Firmware Version (GET_FIRMWARE_VERSION)

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

Format

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

You could render the returned firmware version into an ASCII string using code such as the following:

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

Response

The response is four bytes. The response is LSB-first, so if you read [0xaa, 0xbb, 0xcc, 0xdd] that indicates firmware version dd.cc.bb.aa. Returns four 0xff on error.

4.1.2 Get Microcontroller Serial Number (GET_MICROCONTROLLER_SERIAL_NUMBER, GET_CPU_UNIQUE_ID, etc)

Currently only supported on XS.

Read the unique ID (serial number) of the microcontroller. For STM32 chips, this will be a 12-byte payload in the following format:

- 2 bytes of X coordinate (4 BCD nibbles)
- 2 bytes of Y coordinate (4 BCD nibbles)
- 1 byte of wafer number (8 bit unsigned) Bits 32:39
- 3 bytes of LOT number (ASCII encoded) Bits 40:63
- 4 bytes of LOT number (ASCII encoded) Bits 64:95

Example:

```
[ x45, 0x00, 0x48, 0x00, 0x14, 0x51, 0x33, 0x33, 0x32, 0x36, 0x31, 0x30 ]
  ----x----  -----y----  wafer ------LOT 1-----  --------LOT 2---------
```

yields: pos (45, 48), wafer 20, lot Q33-2610

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0x2c	Request

2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

The CPU serial number will appear as 12 bytes on endpoint 0.

4.1.3 Get Bluetooth Firmware Revision (GET_BLE_FIRMWARE_VERSION)

This command is only available on XS V2.

Read the revision of the Bluetooth firmware code.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xff	Command
2	wValue	2	0x2d	Request
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

The Bluetooth firmware revision will appear as a null-terminated string of up to 32 bytes on endpoint 0.

4.1.4 Get FPGA Firmware Revision (GET_FPGA_FIRMWARE_VERSION)

Read 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	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

The FPGA revision will appear as seven bytes of ASCII codes on endpoint 0. Returns seven 0xff on error.

4.1.5 Get FPGA Compilation Options (READ_COMPILATION_OPTIONS)

Gets the FPGA compilation options register.

On Hamamatsu-based spectrometers, these are the bit definitions:

```
0-2: IntegrationTimeResolution
3-5: DataHeader
  6: HasCFSelect
7-8: LaserType
```

```
  9-11: LaserControl
    12: HasAreaScan
    13: HasActualIntegTime
    14: HasHorizBinning
 23-15: Reserved
```

This command is deprecated on ARM spectrometers.

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	Ignored
6	wLength	3	3	Payload size

Response

Three bytes of FPGA compilation options. Returns three 0xff on error.

4.2 EEPROM Control

4.2.1 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	0x01	Value
4	wIndex	2	Page index (0-7)	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.

4.2.2 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 the command was successful. Returns value 0 if unsuccessful.

4.3 Spectral Acquisition and Frame Header

4.3.1 Acquire Spectrum (ACQUIRE)

Commands the detector to *generate or send* a new measurement using the currently-applied integration time, gain/offset etc.

On most Wasatch spectrometers, the CCD or CMOS sensor is continually acquiring in a "free-running" mode (but not attempting to send most acquisitions to the host over USB). The ACQUIRE command will instruct the spectrometer to send the NEXT COMPLETED acquisition to the host over USB. Therefore, the measurement could be returned anywhere from "immediately" (if the background continuous acquisition had almost completed when the ACQUIRE command arrived) to a full integration time thereafter.

In Area Scan mode, one ACQUIRE command will generate a series of spectral readouts on the bulk endpoints (should be as many as listed in the EEPROM active_pixels_vertical field).

On untethered devices, wValue may be used to indicate "acquisition type" if different types are supported.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xAD	Request
2	wValue	2	0xXXXX	Acquisition Type (default zero)
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Unlike most other spectrometer commands, the response payload for this command is not returned on the same "Control Endpoint 0" used to issue the command.

FX2-based spectrometers with 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 on EP2, while 512-pixel detectors will return 1024 bytes, etc.

FX2-based models with 2048 pixels will output the first half of the spectrum on endpoint 2 as described above, and the second half on endpoint 6 (0x86).

ARM-based spectrometers output all pixels on endpoint 2 (0x82), regardless of pixel count or configured Region of Interest. If the Frame Header is enabled, it will be prefixed to the EP2 output (occurring before the spectral data).

It is important to recognize that the spectrum will be returned over USB in **detector pixel order**, which is not necessarily **spectral order** (wavelength or wavenumber). Older Wasatch spectrometers consistently returned spectral data in increasing order of wavelength, essentially "blue to red" when read left-to-right. For optomechanical reasons, many newer designs mount the detector "upside down" with respect to the grating diffraction order, such that the physical

order of pixel data (counting pixels from 0-1023 etc) will be in *decreasing* order by wavelength (red to blue).

There is a boolean flag called “invertXAxis” in the “featureMask” EEPROM field (documented in ENG-0034) which specifies whether the receiving software library should reverse the order of uint16 pixels received to restore a consistent “blue-to-red” increasing wavelength order. In general Wasatch-supplied driver libraries (Wasatch.NET/PY/VCPP/etc) will automate this processing and always deliver consistently-ordered spectra from calls to “getSpectrum()” regardless of spectrometer model or hardware design.

4.3.2 Acquire Auto-Raman (ACQUIRE_AUTO_RAMAN)

This is similar to ACQUIRE (0xad) in that it commands the spectrometer to generate a spectrum, but different in several key respects:

- takes a number of marshalled parameters
- causes the laser to fire
- runs for an non-deterministic (but bounded) period of time

In general, Acquire Auto-Raman does the following:

- enables the laser
- waits for the laser to energize and stabilize per configured settings
- optimizes integration time (and gain, on CMOS sensors) to achieve a target intensity
- given the optimized integration time, computes the number of averages it can fit within a configured "total measurement time"
- collects the computed number of Raman sample measurements and takes their average
- disables the laser
- collects an equal number of "dark" measurements and takes their average
- performs dark correction
- returns a single dark-corrected averaged Raman spectrum

The optimized integration time, gain and scan averaging are left configured in the spectrometer, and will remain in effect unless changed by the operator.

The averaged dark spectrum is *not* returned or made available, nor is the pre-corrected "averaged sample" Raman spectrum.

The command takes a 23-byte payload, which contains the following marshalled arguments. All values are little-endian unless otherwise specified. Defaults are used if the relevant bytes are 0xff for the entire field length.

Parameter Block Structure

Offset	Length (bytes)	Name	Default	Description
0	2	maxMS	10000	(uint16) total averaged dark-corrected measurement time (Raman + darks); does not

Offset	Length (bytes)	Name	Default	Description
				include optimization or laser stabilization (ms); (range 1000, 60000); only used to compute the number of averages, and averaging can never be <1, so max_ms is not "enforced" in the case where 2x optimized integration time exceeds max_ms.
2	2	startIntegMS	100	(uint16) initial integration time (ms) (range 1, 5000)
4	1	startGainDB	0	(uint8) initial gain (integral dB) (range 0, 30)
5	2	maxIntegMS	2000	(uint16) max integration time (ms) (range 10, 60000)
7	2	minIntegMS	10	(uint16) min integration time (ms) (range 1, 1000)
9	1	maxGainDB	30	(uint8) max gain (integral dB) (range 0, 30)
10	1	minGainDB	0	(uint8) min gain (integral dB) (range 0, 30)
11	2	targetCounts	45000	(uint16) optimized Raman peak intensity (range 1000, 65000)
13	2	maxCounts	50000	(uint16) max Raman peak intensity (range 1000, 65535)
15	2	minCounts	40000	(uint16) min Raman peak intensity (range 100, 65000)
17	1	maxFactor	5	(uint8) scaling factor when increasing integration time or gain (range 2, 100)
18	2	dropFactor	0.5	(FunkyFloat) scaling factor when decreasing integration time or gain (range 0.01, 0.99)
20	2	saturation	60000	(uint16) absolute spectrum max (range 100, 65535)
22	1	maxAvg	10	(uint8) regardless of optimized integration time and maxMS, the most averages to be collected (range 1, 255)

Note that whatever LASER_WARNING_DELAY_SEC has been previously configured (0x8a) will be used during the "wait for laser stabilization" period, along with the laserWarmupSec configured in the EEPROM (see ENG-0034).

An ongoing ACQUIRE_AUTO_RAMAN collection may be aborted by sending another ACQUIRE_AUTO_RAMAN command with zero maxMS (simply initialize the entire payload to all 0x00). Additional context on the configuration parameters and their defaults can be found in Wasatch.PY's wasatch.AutoRamanRequest class. An approximation of the algorithm applied within the spectrometer can be found in wasatch.AutoRaman.

Besides terminating the optimization/collection loop, this will also ensure the laser is disabled.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xfd	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	23	see above	Payload (see above)

Response

No response is returned back on EP0 from this opcode. However, the caller should open a blocking read on EP2 to await the dark-corrected, averaged Raman spectrum (starting new reads as timeouts expire).

4.3.3 Get Auto-Raman Params (GET_AUTO_RAMAN_PARAMS)

This command is only supported on XS V2.

This command can be used to retrieve the "optimized" acquisition parameters generated by ACQUIRE_AUTO_RAMAN.

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x9a	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response
Returns the same byte-structure accepted by ACQUIRE_AUTO_RAMAN.

4.3.4 Get Auto-Raman Status (GET_AUTO_RAMAN_STATUS)

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host

Offset	Field	Size	Value	Description
1	bRequest	1	0x94	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.3.5 Set Frame Header (SET_FRAME_HEADER)

On ARM-based spectrometers, enables or disables a 64-byte "Frame Header" which is prefixed to spectral output.

The frame header must be enabled via an FPGA register to be included in the frame data. The following table outlines the latest frame header structure. Previous frame header versions are documented in the FPGA Theory of Operations document and in the source code associated with a specific FPGA build version.

Offset	Datatype	Name	Description
0	uint16	startMarker	Constant (0xffff). Assumes spectral data is clamped to 0xfffe.
2	uint8	frameHeaderVersion	• 0x01 for 220250/220320 • 0x10 for 220321
3	uint16	fpgaVersionHi	First two ASCII chars ("01" of "01.2.34")
5	uint8	fpgaVersionMid	"2" of "01.2.34"
6	uint16	fpgaVersionLow	"34" of "01.2.34"
8	uint16	totalLengthBytes	Payload size (usually 2 x pixelCount)
10	uint16	requestedUsbFrameCount	Number of USB triggers received since boot or counter reset
12	uint16	deliveredUsbFrameCount	Number of USB frames delivered since boot or counter reset
14	uint16	requestedBleFrameCount	Number of BLE triggers received since boot or counter reset
16	uint16	deliveredBleFrameCount	Number of BLE frames delivered since boot or counter reset
18	uint32	timestampMS	milliseconds since boot (~50 days)
22	uint8	hostControlReg	Reg 0x03 (currently usbMicroActive, usbEnumerated, bleMicroActive,

			blePaired, triggerSelect)
23	uint8	fpgaStatusReg	Reg 0x04 (currently laserActive, laserTimedOut, interlockClosed, sensorSleeping, batLaserReady)
24	uint16	irqErrorShadowRegLo	Reg 0x08 (non-cleared version of IRQ error register lower two bytes)
26	uint8	irqErrorShadowRegHi	Reg 0x08 (non-cleared version of IRQ error register upper byte)
27	uint8	reserved	Reserved for future error bits
28	uint8	irqStatusShadowReg	Reg 0x07 (non-cleared version of IRQ status register)
29	uint8	configurationReg	Reg 0x10 (currently laserEnable, laserMode, sensorReadMode, batteryLedMode, frameHeaderEnable)
30	uint24	integrationTimeReg	Reg 0x11 (integration time register, big-endian)
33	uint8	reserved	Reserved to pad back out to 16-bit boundary for subsequent entries
34	sint16	sensorOffsetReg	Reg 0x12 (signed offset register, big-endian)
36	uint16	sensorGain	16-bit placeholder for gain value to be inserted by STM32 FW
38	uint16	startLineReg	Reg 0x15 (vertical ROI start line of sensor within active imaging region 0-1079)
40	uint16	stopLineReg	Reg 0x16 (vertical ROI ending line of sensor within active imaging region 0-1079)
42	uint16	laserTemperature	From microcontroller
44	uint16	sensorTemperature	Reserved
46	uint16	ambientTemperature	From STM32 thermistor
48	uint16	batteryStatus	From microcontroller
50	uint8[10]	reserved	Future growth
60	uint8	crc8	Payload checksum (does not include

			header) (see Appendix)
61	uint8	padding	Ensure 2-byte boundary
62	uint16	endBytes	Constant (0x5aa5)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x98	Request
2	wValue	2	0xXXXX	Value (1 to enable, 0 to disable)
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

4.3.6 Get Frame Header (GET_FRAME_HEADER)

On ARM-based spectrometers, it returns the enable status of the 64-byte "Frame Header" prefixed to spectral output.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0x99	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns 1 byte (1 if header enabled, 0 if disabled).

4.3.7 Get Image Sensor State (GET_IMAGE_SENSOR_STATE)

Only supported on XS V2.

The command returns a value indicating the current image sensor state.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0x97	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored

6	wLength	2	0	Payload size

Response

Response is 1 byte on Endpoint 0:

- UNKNOWN = 0
- STANDBY = 1
- TRANS_IN_OUT_STANDBY = 2
- REG_HOLD = 3
- ACTIVE = 4
- ERROR = 5
- BUSY = 6

4.3.8 POLL_DATA / ACQUISITION_STATUS

The command returns a value indicating the current spectrometer acquisition state.

Format

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

Response

Response as 1 byte on Endpoint 0.

Byte Number	Size	Value (XS)
1	1	0 = IDLE 1 = dark measurement 2 = laser warmup 3 = sample measurement 4 = processing 5 = stabilizing 6 = data ready for readout 254 = ERROR 255 = UNDEFINED

4.4 Integration Time Control

4.4.1 Set Integration Time (SET_INTEGRATION_TIME)

Sets the integration time which will be used in subsequent acquisitions. In all currently supported spectrometers, the unit is in milliseconds; check FPGA compilation options (READ_COMPILATION_OPTIONS) to confirm integration time resolution. On most spectrometers, the default integration time will be zero to indicate the shortest-possible acquisition supported by the detector.

Spectrometers can specify a desired startup integration time using the "STARTUP_INTEGRATION_TIME" field in the EEPROM (see ENG-0034); however, this field is not automatically read or applied by firmware, and the value must be explicitly set by software to override the hardware default.

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-significant 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.

4.4.2 Get Integration Time (GET_INTEGRATION_TIME)

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

Note that integration time changes are *applied* in the spectrometer at the end of the current integration. As this command reads the current "applied" integration time, the value read may "lag" slightly, as the returned value will not reflect newly assigned integration times until those times have been fully applied to the sensor.

Format

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

Integration time (ms) = data[0] + data[1]<<8 + data[2]<<16 // convert little-endian response

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. Returns all 0xff on error.

4.5 Detector Gain and Offset Control

4.5.1 Set Detector Offset (SET_DETECTOR_OFFSET)

Sets the offset added to the detector pixel values. Defaults to 0 on reset.

The desired detector offset value can be configured in the STARTUP_OFFSET_EVEN and STARTUP_OFFSET_ODD EEPROM fields (see ENG-0034); however, those values are not read or applied automatically by firmware, and must be explicitly set by software to override hardware defaults.

On Hamamatsu and IMX sensors, the offset is a SIGNED int16 (big-endian two's-complement) and ADDED to each vertically binned pixel's intensity.

On InGaAs models, this command only applies to the even-numbered pixels (0, 2, 4…).

In Area Scan mode, offset is not added to the individual spectra output from the 2D detector. On FX2 spectrometers in Area Scan mode, the configured "offset" value is instead used to indicate an integral delay in milliseconds between output lines.

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	Ignored
6	wLength	2	0	Payload size

Response

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

4.5.2 Set Detector Gain (SET_DETECTOR_GAIN)

Sets the gain by which detector pixel values (intensity) are increased (scaled UP). The gain value has a different *unit* on Hamamatsu (CCD) vs IMX (CMOS) sensors, but the unitless fractional value itself has the same format and is passed the same way over USB.

The intended startup gain value can be stored in the EEPROM via the DETECTOR_GAIN_EVEN and DETECTOR_GAIN_ODD EEPROM fields (see ENG-0034); however, those fields are not read or applied automatically by firmware, and must be explicitly set by software to override hardware defaults.

Float16 format

Gain is always passed as an unsigned bfloat16 16-bit float. To marshall or demarshall a 16-bit float, refer to the following bit table:

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

Hamamatsu CCD Notes

On Hamamatsu detectors, the gain value is a simple scalar factor which is multiplied against the raw vertically-binned pixel values to scale them up in a linear fashion. If gain was 1.9, then every pixel in the spectrum would be multiplied by 1.9, so a raw value of 1234 would become 2344 after truncation. Gain is multiplied into raw spectra *before* offset is added.

On older spectrometers, gain defaults to 1.9 on reset (FW hardcode); newer models default to 1.0.

Values less than 1.0 can be used to scale-down spectra, but this is not recommended as it would reduce the effective dynamic range of the sensor and ADC, essentially losing information.

On InGaAs models, this command only applies to even-numbered pixels (0, 2, 4…). See SET_DETECTOR_GAIN_ODD for odd-numbered pixels.

Sony IMX Implementation

On IMX sensors, gain is treated as a fractional value in decibels (dB), with a supported precision of 0.1dB. The supported range of gain in dB varies by sensor, but for the Sony IMX385 the sensor's ADCs support an analog gain from 0.0 – 30.0dB; values of up to 72.0dB are accepted, but the provided gain above 30.0dB will be digitally scaled by the sensor electronics.

On Sony IMX detectors, gain is sent big-endian (same as read back by GET_DETECTOR_GAIN).

Format

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

Response

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

4.5.3 Get Detector Offset (GET_DETECTOR_OFFSET)

Reads the signed offset added to the vertically-binned detector pixel values.

On InGaAs models, this command only applies to even-numbered pixels (0, 2, 4…).

Format

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

Response

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

4.5.4 Get Detector Gain (GET_DETECTOR_GAIN)

Reads the detector gain, as an unsigned bfloat16 (see Set Detector Gain (SET_DETECTOR_GAIN)).

On FX2 / Hamamatsu spectrometers, the returned gain is little-endian (opposite of SET_DETECTOR_GAIN).

On ARM / XS spectrometers, the returned gain is big-endian (same as SET_DETECTOR_GAIN).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xC5	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.6 Scan Averaging

Onboard scan-averaging allows multiple spectra to be acquired and averaged within the spectrometer, such that only the final (averaged) spectra is output over USB.

Note that even averaged spectra is returned as an array of uint16 pixels, so averaged pixels will be *rounded* to the nearest positive integer before transmission. **This will incur some loss of precision**; if greater precision is required, users are recommended to read the full set of measurements directly, and average them in host software.

4.6.1 Set Scan Averaging (SET_SCANS_TO_AVERAGE)

Configures on-board scan averaging. This function is called internally by Auto-Raman mode, so the value you last configured may be overridden by Auto-Raman measurements.

Values are 0 are treated equivalent to 1.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xff	Request
2	wValue	2	0x62	Command
4	wIndex	2	1-65535	value (uint16)
6	wLength	2	0	Payload size

4.6.2 Get Scan Averaging (GET_SCANS_TO_AVERAGE)

Returns the current on-board scan averaging configuration. This may have been set via an explicit call to SET_SCANS_TO_AVERAGE, or via a BLE write to the same register, or could have been derived internally using Auto-Raman mode.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xff	Request
2	wValue	2	0x63	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns a little-endian uint16 indicating the active scan averaging (1-65535).

4.7 Laser Control

This section describes commands controlling the laser_enable setting that determines whether the laser is firing or not, as well as other laser-related features such as laser temperature.

Note that the following section on **Modulation Control** is also highly relevant to laser control, as pulse-width modulation (PWM) is used to control the average output laser power of Raman systems (see Modulation Control).

4.7.1 Laser Interlock Overview

FX2-based Spectrometers

Wasatch Photonics spectrometers with integrated multi-mode lasers (MML) include a laser interlock board to support FDA laser-safety requirements. Laser interlock features include a removable two-position key-switch and an optional “continuity interlock circuit” which may be integrated into external laser-safety systems (such as laboratory door “crash-bar” circuits).

MML-equipped systems also include two laser status LEDs to communicate laser status to the user: a yellow “Laser Armed” LED, and a red “Laser Firing” LED.

The laser is considered “armed” (literally, is powered) if and only if:

- the key-switch is inserted and turned to the upright “firing” position; AND
- the continuity circuit is closed, typically via the included microphone jack dongle; AND
- the spectrometer is plugged into 12VDC power; AND
- the spectrometer is switched “on” (for units with a power switch).

If the laser is armed, the yellow status LED will flash at a 1Hz rate (50% duty cycle).

In addition, if the spectrometer has been commanded to fire the laser, the red status LED will flash at the same 1Hz rate (50% duty cycle). Note that there is an in-built delay between the spectrometer accepting the command to fire the laser, and when the laser actually begins emitting a beam. The red status LED will begin flashing as soon as the “laser_enable” command is received (if the laser was already properly armed), not when the laser actually begins firing.

XS Spectrometers

XS Raman spectrometers use a single-mode laser (SML) with a front-mounted Reed switch magnetic interlock, and a keypad including a red "laser status" LED. The key LED flashes when the laser is "ARMED," and illuminates solid when the laser is either firing, or is about to fire (per LASER_WARNING_DELAY). The laser is considered "ARMED" when the interlock is closed, and at least one of the following is true:

- the spectrometer has enumerated to a host PC over USB, *and/or*
- the spectrometer has successfully paired to a BLE host ("Central" in BLE parlance) over Bluetooth

XS spectrometers also have a configurable LASER_WATCHDOG which will automatically disable the laser after a configured firing time.

4.7.2 Set Laser Enable (SET_LASER_ENABLE)

On Raman spectrometers, 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.

On Gen 1.5 non-Raman spectrometers, this opcode is also used for SET_STROBE_ENABLE.

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	Ignored
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.

4.7.3 Get Laser Enable (GET_LASER_ENABLE)

On Raman spectrometers, 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.

Also note that some lasers include a deliberate delay between when firing has been requested, and when the laser actually begins emitting. This delay is provided for operator safety reasons, giving the user time to observe and react to the external warning LED before the laser actually emits hazardous radiation.

Furthermore, even if a laser has been commanded to fire, it will not do so if the laser safety interlock has been engaged, for instance by removing the key or switching it to the “safe” position, or by leaving the continuity interlock safety circuit open (removing the audiojack dongle).

In all such cases, the “laser enable” status returned by this command indicates *whether the laser has been commanded to fire by software, not whether the laser is actually physically firing*; for that, see GET_LASER_IS_FIRING.

On Gen 1.5 non-Raman spectrometers, this opcode is also used for GET_STROBE_ENABLE.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xE2	Request
2	wValue	2	0xXXXX	Ignored

4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.7.4 Get Laser Temperature (GET_LASER_TEMPERATURE, aka GET_ADC)

In normal operation, it 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 an equation specific to the thermistor.

XM-ILPSM-OEM models with IPS SML lasers are monitored with thermistor TDK B57862S0103F040 and can be converted to Celsius using the following equation:

```
// 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
```

XS spectrometers using an IPS or RealLight SML can be converted to °C using this equation:

```
# conversion for 220250 Rev4 MAX1978ETM-T 3V3 buffer -> 12-bit DAC -> degC
coeffs = [ 1.5712971947853123e+000,
           1.4453391889061071e-002,
          -1.8534086153440592e-006,
           4.2553356470494626e-010 ]
for i, coeff in enumerate(coeffs):
    degC += coeff * pow(raw, i)
```

This command relates to Raman spectrometers with an IPS single-mode laser (SML); the laser driver board currently used with Ondax multi-mode lasers (MML) does not support temperature readout.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xD5	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored

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.

4.7.5 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. On X/XM models this is 7-bit value (0-127), while on XS models it is 12-bit (0-4095).

This value is used exclusively during manufacturing calibration to balance a hardware potentiometer (locked after calibration), and is not recommended for end-user manipulation. The supported temperature maintained in the laser is prescribed by the laser manufacturer and is not intended to be tuned or adjusted 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 X/XM models, the laser TEC setpoint "at boot" will always be 63, the midpoint of the 7-bit range. On XS models, the laser TEC setpoint will automatically be configured from the EEPROM, or default to 800 if no valid EEPROM setting is found.

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 (7-bit X/XM, 12-bit XS)
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.7.6 Get Laser TEC Setpoint (GET_LASER_TEC_SETPOINT)

The command reads the laser temperature set point. See "SET_LASER_TEC_SETPOINT" 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	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload

Response

X/XM: 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).

XS: On ARM, this is a 12 bit value that appears in the first two bytes on endpoint 0.

4.7.7 Set Laser TEC Mode (SET_LASER_TEC_MODE)

On XS spectrometers, the SML TEC has four selectable modes:

- 0 = OFF (disabled, stays off until otherwise commanded)
- 1 = ON (enabled, stays on until otherwise commanded)
- 2 = AUTO (on whenever the laser is emitting, off when the laser is off)
- 3 = AUTO_ON (the TEC stays off until the laser is fired; the TEC enables as soon as the laser begins firing; the TEC remains operating after the laser ceases to fire, and remains on until the unit is shutdown or the LASER_TEC_MODE is changed)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x84	Request
2	wValue	2	enum (0-3)	set wValue to desired mode
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

0 on success, non-zero on failure.

4.7.8 Get Laser TEC Mode (GET_LASER_TEC_MODE)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x85	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

One byte, matching the 4-value enum of SET_LASER_TEC_MODE. 0xff indicates an error.

4.7.9 Get Laser Interlock (GET_LASER_INTERLOCK, aka CAN_LASER_FIRE)

The command returns the status of the laser interlock as specified in Laser Interlock Overview.

Format

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

Response

Response is 0 (interlock circuit open, laser cannot fire) or non-zero (interlock circuit closed, laser armed and able to fire) on endpoint 0.

On XS spectrometers, the non-zero value is currently 0x08, as the byte returned is literally the FPGA_STATUS register and'ed with the INTERLOCK_CLOSED bit (bit 3).

4.7.10 Get Laser Is Firing (GET_LASER_IS_FIRING)

This command returns whether or not the laser is currently firing.

This is subtly distinct from the Get Laser Enable (GET_LASER_ENABLE) command, which indicates whether the spectrometer has been "requested" to fire the laser (or alternatively, whether the spectrometer is "trying" to fire the laser). This is also distinct from the Get Laser Interlock (GET_LASER_INTERLOCK) command, which returns whether the spectrometer is able to fire the laser. This is even somewhat different from the visual status indicated by the red "Laser Firing" status LED described in Laser Interlock Overview, as that LED includes a short FDA-recommended padding in which the LED begins flashing before the laser actually starts emitting energy; this command has no such padding and returns the physical instantaneous state of emission.

Unlike any of the above similar-and-related commands, this command quite simply returns whether the integrated laser, to the best of the electronics' current knowledge, actively is firing. Conceptually, this command should return a similar status to what you could measure by coupling an optical power meter to the laser and detecting the physical emission directly.

This should not be affected by PWM duty-cycle, so long as the configured pulse width is non-zero and pulse period no more than 1ms.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xff	2nd-Tier Command
2	wValue	2	0x0d	Request
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Response is 0 (laser is not currently firing) or 1 (laser is actively firing) on endpoint 0.

4.7.11 Set Laser Power Attenuator (SET_LASER_POWER_ATTENUATOR)

On XS-series spectrometers, allows the integrated laser power attenuator digital potentiometer to be set to an 8-bit DAC value (higher values = greater attenuation, zero = no attenuation).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x82	request
2	wValue	2	0x00XX	value (0-255)
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns one byte (0 = SUCCESS, 4 = I2C_READ_WR_ERROR, 7 = SETPOINT_RD_BACK_MISMATCH_FLR, 8 = OTP_FUSES_IN_BAD_STATE, 9 = SETPOINT_LOCKED, 10 = UNKNOWN_OTP_FUSE_STATE)

4.7.12 Get Laser Power Attenuator (GET_LASER_POWER_ATTENUATOR)

On XS-series spectrometers, it reads the integrated laser power attenuator digital potentiometer's 8-bit DAC value (higher values = greater attenuation, zero = no attenuation).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0x83	request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response: two bytes:

- byte 0: error code (0 = SUCCESS, 4 = I2C_READ_WR_ERROR, 8 = OTP_FUSES_IN_BAD_STATE, 9 = SETPOINT_LOCKED, 10 = UNKNOWN_OTP_FUSE_STATE)
- byte 1: raw DAC value (should mirror most-recent SET_LASER_POWER_ATTENUATOR)

4.7.13 Set Laser Warning Delay (SET_LASER_WARNING_DELAY_SEC)

On XS-series spectrometers, configures the laser warning delay which determines how long the red keypad LED will illuminate solid, after the laser is instructed to fire, before the laser actually begins emitting.

This delay is provided in deference to 21 CFR 1040.10(f)(5)(ii), and defaults to 3 seconds. Supported values are 0-254 seconds. A value of 255 should be rejected and generate an error.

A value of 0 essentially disables the delay, which users should exercise with caution as this **may contravene applicable laser safety regulations** in your jurisdiction.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x8a	request
2	wValue	2	0x00XX	requested delay in seconds (0-254)
4	wIndex	2	0xXXXX	ignored
6	wLength	2	0	Payload size

Response

Returns one byte (0 = SUCCESS, non-zero for error)

4.7.14 Get Laser Warning Delay (GET_LASER_WARNING_DELAY_SEC)

On XS-series spectrometers, reads the currently configured laser warning delay in seconds.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x8b	request
2	wValue	2	0xXXXX	ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response: one byte, indicating the currently configured delay in seconds. A value of 0xff indicates an error.

4.7.15 Set Laser Watchdog (SET_LASER_WATCHDOG)

This command (and laser watchdog functionality) is only available on XS-Series spectrometers.

Whether the Laser Mode is Manual or Raman, the Laser Watchdog will specify a period of seconds after which the laser will automatically be disabled.

- Writing a 0x0000 value will disable the watchdog.

- Any other value will be taken to be an unsigned count of seconds, after which the laser should be automatically disabled.

The laser watchdog is reset (stopwatch set to zero) whenever the laser is turned *on*. That is, if you set the laser watchdog to 10sec, fire the laser, manually turn the laser off after 4sec, wait 4sec, then turn the laser on with another expected 4sec integration, the laser will not automatically cut-off 2sec into the second integration. The 10sec watchdog will have been reset at the beginning of each “laser on” command.

Similarly, if the watchdog expires when the laser is already disabled, no change will occur or notification sent. (Example: if you set a 10sec watchdog, turn the laser on for 3sec, and then manually turn the laser off, nothing will happen 7sec later.)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xFF	Request
2	wValue	2	0x18	Command
4	wIndex	2	seconds	Watchdog timeout in sec (big-endian uint16)
6	wLength	2	0	Payload size

Response

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

4.7.16 Get Laser Watchdog (GET_LASER_WATCHDOG)

Returns the current value of the Laser Watchdog.

This command is only available on XS-Series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0x17	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns current laser watchdog timeout value in seconds (two byte big-endian uint16).

4.7.17 Set Laser TEC Watchdog (SET_LASER_TEC_WATCHDOG)

Format

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

1	bRequest	1	0xFF	Request
2	wValue	2	0x7d	Command
4	wIndex	2	seconds	Watchdog timeout in sec (big-endian uint16)
6	wLength	2	0	Payload size

Response

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

4.7.18 Get Laser TEC Watchdog (GET_LASER_TEC_WATCHDOG)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0x7e	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns current laser TEC watchdog value in seconds (two byte big-endian uint16).

4.8 Modulation Control

Modulation commands have different effects on Raman vs non-Raman systems.

Raman Spectrometers

On Raman systems, modulation commands control the average output laser power. They do this by configuring a "duty cycle" in the laser, in which the laser is rapidly turned on and off. The laser is technically a CW (Continuous Wave) laser, and its instantaneous output power is always either "full power" or "off" (other than ramp-up fluctuations). However, by manually pulsing the laser via PWM (Pulse-Width Modulation), we can approximate lower "average" power levels, which can roughly be considered a "percentage" of full power.

We say "roughly" a percentage because when you first turn a laser on, there is a brief period of instability in output power. When you are turning a laser on and off rapidly, that initial instability is amplified because it is the "unstable" initial period which is now being repeatedly pulsed at a duty cycle.

The duty cycle is expressed programmatically as a PULSE_PERIOD and PULSE_WIDTH, both in microseconds. The laser will fire for the first PULSE_WIDTH microseconds of every PULSE_PERIOD, and be switched off for the remainder of the PULSE_PERIOD. That is, if the PULSE_PERIOD is 100µs, and the PULSE_WIDTH is 20µs, then the laser will fire for 20µs and then switch off for 80µs, repeating the pattern every 100µs, for a 20% duty cycle which should provide "approximately" 20% of the laser's full power.

Uint40 Parameters

On FX2-based (X/XM series) spectrometers, all modulation values are expressed in 40-bit unsigned integers. As the combined storage of the standard USB Control Packet's wValue and wIndex fields only provide 32 bits of storage, one byte of payload space is used as well.

40-bit integers in all commands are sent from the host to the device as follows:

- The least-significant word (LSW, bits 0..15) is sent as the wValue
- The next-most significant word (MSW, bits 16-31) are sent as the wIndex
- The most-significant byte (MSB, bits 32-39) are sent as the first byte of the payload

It is recommended that a full 8 bytes be provided as payload, but only the first byte is used.

When Uint40 values are returned by "getter" commands, they are sent as 5-byte little-endian sequences.

ARM-based (XS) spectrometers use 32-bit PWM, so no payload is required.

4.8.1 Set Modulation Pulse Period (SET_MOD_PULSE_PERIOD)

This command sets the modulation period in microseconds. Typical values used are 1000µs (default in ENLIGHTEN and Wasatch drivers) or 100µs.

Format

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

Offset	Field	Size	Value	Description
1	bRequest	1	0xc7	Request
2	wValue	2	LSW	Bits 0..15 of 40-bit value
4	wIndex	2	MSW	Bits 16..31 of 40-bit value
6	wLength	8	MSB + extra	On FX2 spectrometers, payload byte 0 contains bits 32..39 of 40-bit value (remaining 7 bytes are ignored)

4.8.2 Get Modulation Pulse Period (GET_MOD_PULSE_PERIOD)

This command gets the currently configured modulation pulse period in microseconds.

Format

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

Response

FX2 spectrometers: uint40 value in 5 little-endian bytes.

> Example: [0xaa 0xbb 0xcc 0xdd 0xee] = 0xeeddccbbaa = 1025923398570µs, or about 12 days

XS spectrometers: uint32 value (little-endian)

4.8.3 Set Modulation Pulse Width (SET_MOD_PULSE_WIDTH)

This command sets the modulation width in microseconds. If the modulation period is 1000µs, a pulse width of 333µs would represent a 33% duty cycle, or roughly 33% of the laser's full power.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xdb	Request
2	wValue	2	LSW	Bits 0..15 of 40-bit value
4	wIndex	2	MSW	Bits 16..31 of 40-bit value
6	wLength	8	MSB + extra	On FX2 spectrometers, payload byte 0 contains bits 32..39 of 40-bit value (remaining 7 bytes are ignored)

4.8.4 Get Modulation Pulse Width (GET_MOD_PULSE_WIDTH)

This command gets the currently configured modulation pulse width in microseconds.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xdc	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

FX2 spectrometers: uint40 value in 5 little-endian bytes.

> Example: [0xaa 0xbb 0xcc 0xdd 0xee] = 0xeeddccbbaa = 1025923398570µs, or about 12 days

XS spectrometers: uin32 value (little-endian)

4.8.5 Set Modulation Enable (SET_MOD_ENABLE)

Raman Spectrometers

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

4.8.6 Get Modulation Enable (GET_MOD_ENABLE)

Raman Spectrometers

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	Ignored
4	wIndex	2	0xXXXX	Ignored
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.

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

4.8.7 Set Modulation Linked to Integration (SET_MOD_LINKED_TO_INTEGRATION)

Raman Spectrometers

This command is planned for XS-series spectrometers, but not currently implemented.

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

When this linkage is enabled AND laser modulation is enabled, the laser **emits** only while integrations are taking place, and the laser **stops emitting** 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 MOD_ENABLE is disabled, the laser will not be modulated. Likewise, if 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	Ignored
6	wLength	2	0	Payload size

Response

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

4.8.8 Get Modulation Linked to Integration Time (GET_MOD_LINKED_TO_INTEGRATION)

This command is planned for XS-series spectrometers, but not currently implemented.

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

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xDE	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
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.

4.9 Detector Temperature Control

4.9.1 Set Detector Thermo-Electric Cooler Enable (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	Ignored
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.

4.9.2 Get 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	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.9.3 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:

Temperature °C = $C_0 + C_1(raw) + C_2(raw^2)$

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD7	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
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.

4.9.4 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	0xXnnn	12-bit setpoint value (most-significant nibble ignored)
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.

4.9.5 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	Ignored
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).

4.9.6 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	Ignored
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.

4.9.7 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	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.10 Trigger Control

4.10.1 Set Trigger Source (SET_TRIGGER_SOURCE)

This command is planned for XS-series spectrometers, but not currently implemented.

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

"Software triggering," the default, simply means that the spectrometer will not expect to generate and return a spectrum to the host until it receives an ACQUIRE command (0xad).

"Hardware triggering," the only currently supported alternative, is to wait until a rising edge signal arrives on the "EXTERNAL_HARDWARE_TRIGGER_IN" pin of the external accessory connector. At that point, the spectrometer will generate and return a spectrum exactly as though it had received an ACQUIRE command via USB, but with lower latency and timing jitter.

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	Ignored
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.

4.10.2 Get Trigger Source (GET_TRIGGER_SOURCE)

This command is planned for XS-series spectrometers, but not currently implemented.

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	Ignored
4	wIndex	2	0xXXXX	Ignored
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.

4.10.3 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 yet implemented on XS series spectrometers

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

4.10.4 Get Trigger Delay (GET_TRIGGER_DELAY)

The command reads the trigger delay. Enabled only on X series spectrometers. Not yet implemented on XS series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xAB	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
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) = $B_0 + B_1 << 8 + B_2 << 16$.

4.11 High-Gain Mode

InGaAs-based spectrometers (NIR1/2, WP-1064) support selectable analog gain modes on the photodiode array. This is separate from the digital gain applied in the FPGA. Although InGaAs detectors may support more than 2 selectable analog gain modes, only two ("low," the firmware default, and "high") are implemented and selectable in Wasatch electronics.

In electrical documentation, this feature is referred to as "CF_SELECT," the chip-level signal used to enable the feature.

These commands are not available on CCD or CMOS detectors.

4.11.1 Set High-Gain Mode Enable (SET_HIGH_GAIN_MODE_ENABLE)

This command turns high-gain mode on or off.

Note this is overloaded with SET_AREA_SCAN on silicon detectors.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0xeb	Request
2	wValue	2	0 = disable high gain mode 1 = enable high gain mode	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.11.2 Get High Gain Mode Enable (GET_HIGH_GAIN_MODE_ENABLE)

This command returns the current state of high-gain mode.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xec	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

One byte:

- 0 = High gain mode disabled (default in firmware)
- 1 = High gain mode enabled

4.12 Battery Control

4.12.1 Get Battery state (GET_BATTERY_STATE)

Gets the battery percentage from the gas gauge. This command is only supported on XS-Series 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	Command
4	wIndex	2	0xXXXX	Ignored
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: integer 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.

4.12.2 Set Battery Charger Enable (SET_BATTERY_CHARGER_ENABLE)

Allows software to optionally dis/enable the battery charger so that USB communications can remain engaged for monitoring and driving load, while deliberately allowing the battery to discharge during lifetime testing.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x86	Request
2	wValue	2	0 or 1	0 for disable, 1 for enable
4	wIndex	2	0xXXXX	ignored
6	wLength	2	0	Payload size

4.13 Area Scan and Detector ROI

4.13.1 Set Area Scan Enable (SET_AREA_SCAN_ENABLE)

This command is intended for manufacturing use during initial spectrometer build and alignment, and is not recommended for customer use.

This command is supported on all 2D detectors, meaning all silicon detectors but not InGaAs.

In Area Scan mode, an ACQUIRE command will trigger a single acquisition on the 2D detector. However, instead of that acquisition being vertically binned (summed down) into a single row containing all the aggregate intensities of each column, a series of lines will be read-out containing the raw pixel data for each row of the 2D image.

That is, if the detector has 64 lines (per the EEPROM's active_pixels_vertical field), the spectrometer will output 64 distinct "spectra" in rapid order. Each "spectrum" will correspond to a single physical row on the detector. To aid in processing, the first pixel (spectrum[0]) is automatically overwritten with that spectrum's row index (0-63 in this example).

The caller will be expected to rapidly read the normal spectral endpoint(s) (0x82 and perhaps 0x86 depending on model). The caller should read the expected number of spectra, as reported in the EEPROM active_pixels_vertical field.

Re-marshalling the collected lines into a cohesive 2D image for operator analysis is then left to the caller in software, but Wasatch can provide examples in Python or C# if needed.

Note this command is overloaded with SET_HIGH_GAIN_MODE on InGaAs detectors.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0xeb	Request
2	wValue	2	0 or 1	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.13.2 Set Detector Start Line (SET_DETECTOR_START_LINE)

Sets 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 XS-Series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0xFF	Request

2	wValue	2	0x21	Command
4	wIndex	2	line	16-Bit Unsigned, 0-Based Line Number
6	wLength	2	0	Payload size

Response

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

4.13.3 Get Detector Start Line (GET_DETECTOR_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 XS-Series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0x22	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.13.4 Set Detector Stop Line (SET_DETECTOR_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 XS-Series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xFF	Request
2	wValue	2	0x23	Command
4	wIndex	2	line	16-Bit Unsigned, 0-Based Line Number
6	wLength	2	0	Payload size

Response

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

4.13.5 Get Detector Stop Line (GET_DETECTOR_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 XS-Series spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0x24	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.14 Ambient Temperature

4.14.1 Get Ambient Temperature ARM (GET_AMBIENT_TEMPERATURE_ARM)

This queries the spectrometer to provide the ARM's built-in PCB temperature. Note this is not "air" temperature within the housing, but a readout from a thermistor built into the STM32H7. It can be used in conjunction with the laser temperature and detector temperature (on systems where those are available) for added insight into prevailing environmental conditions across a system.

The value is a signed char (-128, 127) and should contain the temperature in integral degrees Celsius.

This command only applies to XS spectrometers.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0xff	Second tier command
2	wValue	2	0x002a	Request
4	wIndex	2	0xXXXX	Ignored
6	wLength	1	0	Payload size

Response

Returns temperature in degrees Celsius as a signed char (-128, 127). For clarity, 31°C would be 0x1f.

4.14.2 Get Ambient Temperature LM57B (GET_AMBIENT_TEMPERATURE_LM57B)

This queries the spectrometer to provide a reading of ambient temperature in degrees Celsius using a board-mounted LM75B. Note that this differs from GET_DETECTOR_TEMPERATURE, which specifically measures the temperature of the sensor itself.

Supported theoretical values (per the component datasheet, not the spectrometer's operating range) are:

- −55°C to +125°C
- 11-bit ADC with resolution of 0.125°C
- Accuracy
    - ±2°C from −25°C to +100°C
    - ±3°C from −55°C to +125°C

This command is only available on non-Raman spectrometers with Gen 1.5 electronics.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x35	Request
2	wValue	2	0xXXXX	Ignored

4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns temperature in degrees Celsius as a big-endian 2-byte value.

See extract of LM75B below for how to interpret the two bytes.

The Temperature register (Temp) holds the digital result of temperature measurement or monitor at the end of each analog-to-digital conversion. This register is read-only and contains two 8-bit data bytes consisting of one Most Significant Byte (MSByte) and one Least Significant Byte (LSByte). However, only 11 bits of those two bytes are used to store the Temp data in two's complement format with the resolution of 0.125 °C. Table 9 shows the bit arrangement of the Temp data in the data bytes.

Table 9. Temp register

MSByte								LSByte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X

When reading register Temp, all 16 bits of the two data bytes (MSByte and LSByte) are provided to the bus and must be all collected by the controller to complete the bus operation. However, only the 11 most significant bits should be used, and the 5 least significant bits of the LSByte are zero and should be ignored. One of the ways to calculate the Temp value in °C from the 11-bit Temp data is:

1. If the Temp data MSByte bit D10 = 0, then the temperature is positive and Temp value (°C) = +(Temp data) × 0.125 °C.
2. If the Temp data MSByte bit D10 = 1, then the temperature is negative and Temp value (°C) = −(two's complement of Temp data) × 0.125 °C.

11-bit binary (two's complement)	Hexadecimal value	Decimal value	Value
011 1111 1000	3F8	1016	+127.000 °C
011 1111 0111	3F7	1015	+126.875 °C
011 1111 0001	3F1	1009	+126.125 °C
011 1110 1000	3E8	1000	+125.000 °C
000 1100 1000	0C8	200	+25.000 °C
000 0000 0001	001	1	+0.125 °C

11-bit binary (two's complement)	Hexadecimal value	Decimal value	Value
000 0000 0000	000	0	0.000 °C
111 1111 1111	7FF	−1	−0.125 °C
111 0011 1000	738	−200	−25.000 °C
110 0100 1001	649	−439	−54.875 °C
110 0100 1000	648	−440	−55.000 °C

Figure 1 Extract from https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

4.15 Board State

4.15.1 Get Ambient Temperature ARM (GET_AMBIENT_TEMPERATURE_ARM)

On spectrometers with a suitable ARM-based microcontroller, this will return the "ambient" temperature as returned by the ARM's built-in thermistor in degrees Celsius.

WARNING: Conflicts with GET_BATTERY_MILLIVOLTS.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xFF	Request
2	wValue	2	0x2a	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Single byte containing ambient temperature (at ARM) in degrees Celsius (integral portion only, no decimal).

4.15.2 Set DFU Mode (SET_DFU_MODE)

This command will immediately transition ARM-based spectrometers into DFU (Dynamic Firmware Update) mode. As soon as a spectrometer enters DFU mode, it is no longer responsive to the Wasatch Photonics USB API, and essentially ceases to be a “spectrometer” for purposes of data communications. It has become an ARM chip awaiting firmware installation via DeFUse Demonstrator or similar utility.

Once DFU mode is enabled, the only way to disable it (as it no longer listens to USB spectrometer commands) is to power-cycle the device.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xfe	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

No response is possible, as the spectrometer will no longer be acting under the original VID and PID; ongoing communications will be disrupted entirely.

4.15.3 Get USB Adapter Info (GET_USB_ADAPTER_INFO, GET_POWER_CONNECTION_STATE)

This command is only supported on XS V2.

Read information about the upstream USB charging source plugged into the spectrometer's USB-C connector.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x78	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns 5 bytes:

- Byte 0: BC1.2 Adapter Type (0: Not found, 1: SDP, 2: CDP, 3: DCP)
- Byte 1: BC1.2 Proprietary Charger Type (0: None, 1: Samsung 2A, 2: Apple 0.5 A, 3: Apple 1A, 4: Apple 2A, 5: Apple 12W, 6: DCP 3A, 7: Unknown)
- Byte 2: Type C CC Current Capability (0: Not connected, 1: 500 mA, 2: 1500 mA, 3: 3000 mA)
- Bytes 3 and 4: Current Limit in mA (Network Byte / Big Endian format)

4.15.4 Reset Field-Programmable Gate Array (RESET_FPGA)

This command will attempt to “reset” the FPGA if it has entered a non-cooperative state.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xb5	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.16 Power Management

4.16.1 Power Off (VR_POWER_OFF)

On XS-series spectrometers, this command can be used to power-off the device (normally accessed by pressing the “power” button on the keypad).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x87	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.16.2 Reboot Device (RESET_UNIT)

This command is only supported on XS V2.

Allows software to reboot the spectrometer over USB.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x93	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.16.3 Set Detector Timeout (SET_DETECTOR_TIMEOUT_SEC)

On XS series spectrometers, the sensor can be configured to automatically power-down (enter a power-saving sleep mode) after a configured number of seconds have elapsed without an ACQUIRE request arriving over either USB or BLE. This is distinct from the "power watchdog" which sends the entire spectrometer into a low-power sleep mode (equivalent to pressing the "power" button on the keypad) in that the microcontroller, BLE controller and FPGA remain powered and responsive; only the image sensor has been deactivated.

Note that re-activating the sensor can be accomplished simply by sending an ACQUIRE opcode, which will force re-enabling of the sensor. A handful of "throwaway stabilization" spectra may be required after rebooting an image sensor.

The default timeout is 60 seconds, which prioritizes user convenience and low-latency responsiveness over power savings. A shorter value such as 10sec would reduce power (and heat, and therefore TEC power and more heat), at the cost of more-frequent throwaways after re-awakening.

A value of 0 will disable the timeout, meaning the sensor will be on as long as the unit is powered (optimal latency yet worst-case heat and power).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0x8e	Request
2	wValue	2	0xXXXX	configured timeout (0-65535 sec)
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

One byte (zero on success, non-zero on error)

4.16.4 Get Detector Timeout (GET_DETECTOR_TIMEOUT_SEC)

Returns the currently-configured detector timeout in seconds.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0x8f	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns 2 bytes (little-endian detector timeout in seconds).

4.16.5 Set Power Watchdog (SET_POWER_WATCHDOG_SEC)

On battery-powered XS series spectrometers, a power management watchdog can be configured to automatically put the spectrometer into "sleep" mode after a configured length of time with no activity.

The power-management watchdog only runs if the unit is operating off battery power, meaning the battery is not charging and the unit is not enumerated over USB. By definition, the only "activity" which could therefore occur would necessarily be over BLE.

At present, any BLE activity over a paired Bluetooth connection (including passive Characteristic reads like BatteryLevel or LaserState) are sufficient to "reset" the watchdog and prevent the unit from automatically transitioning to power-savings mode.

If the spectrometer is not paired over BLE, then no qualifying activity can occur and the unit will automatically power-down after the configured watchdog period.

Use-Case

- User connects spectrometer to laptop over USB and turns on with the keypad power button. User collects spectra for awhile, then unplugs spectrometer and goes to lunch (or takes laptop to meeting, etc). Spectrometer was never actually turned "off" via keypad, so sits drawing power, LEDs blinking, advertising over BLE, etc. Goal is for the

Power Management Watchdog to automatically put the spectrometer to sleep after 10min of being "unused."

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host ⟶ Device
1	bRequest	1	0xff	2nd-tier command
2	wValue	2	0x0030	request
4	wIndex	2	0xXXXX	configured timeout (0-65535 sec)
6	wLength	2	0	Payload size

Response

One byte (zero on success, non-zero on error)

4.16.6 Get Power Watchdog (GET_POWER_WATCHDOG_SEC)

Returns the currently-configured power watchdog timeout in seconds.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device ⟶ Host
1	bRequest	1	0xff	2nd-tier command
2	wValue	2	0xXX31	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns 2 bytes (little-endian power watchdog timeout in seconds).

5 Acquisition Workflow

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

- C/C++: https://github.com/WasatchPhotonics/Wasatch.VCPP
- 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.

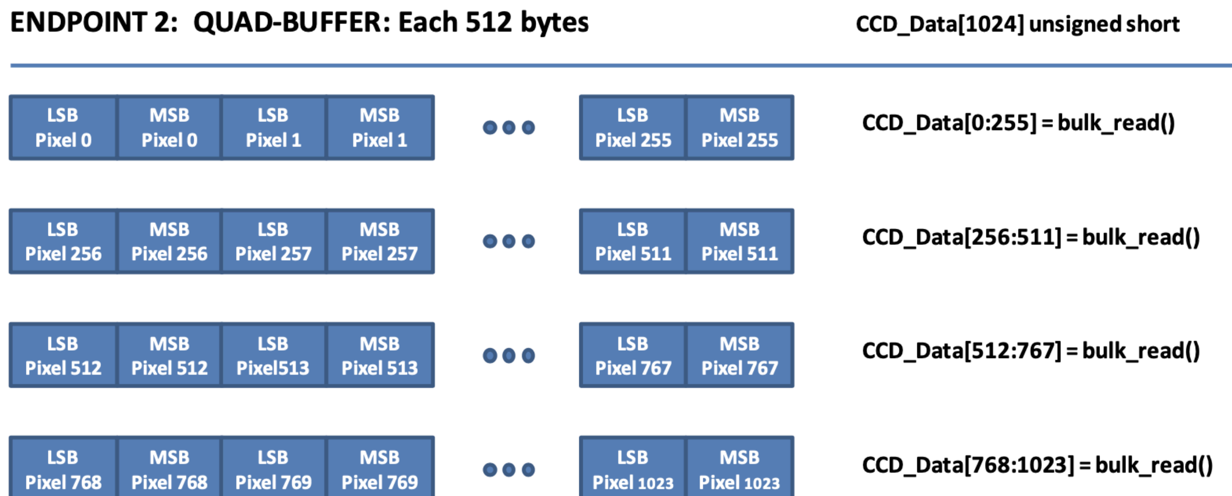
5.1 Spectral Acquisition

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).

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 XS-Series 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 libusb-win32 drivers):

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

1. LINK_MOD_TO_INTEGRATION = 0
2. MOD_ENABLE = 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 libusb-win32 drivers):

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

1. MOD_LINKED_TO_INTEGRATION = 1
2. MOD_ENABLE = 1
3. 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. MOD_PULSE_DELAY = 1500 // 1.5ms in µs
7. MOD_PULSE_WIDTH = 2500 // 50% of PULSE_PERIOD
8. MOD_PULSE_PERIOD = 5000
9. MOD_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)

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 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. MOD_LINKED_TO_INTEGRATION = 1

2. MOD_ENABLE = 1
3. LASER_ENABLE = 1 // per linkage, shouldn't actually fire until "get_spectrum"

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

4. 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.

5.3.1 SPECIFIC EXAMPLE

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

1. INTEGRATION_TIME = 4 // (ms) Integration time < pulse delay + period.
2. MOD_PULSE_DELAY = 1500 // (µs)
3. MOD_PULSE_WIDTH = 5000 // (µs) (full power)
4. MOD_PULSE_PERIOD = 5000 // (µs)
5. MOD_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

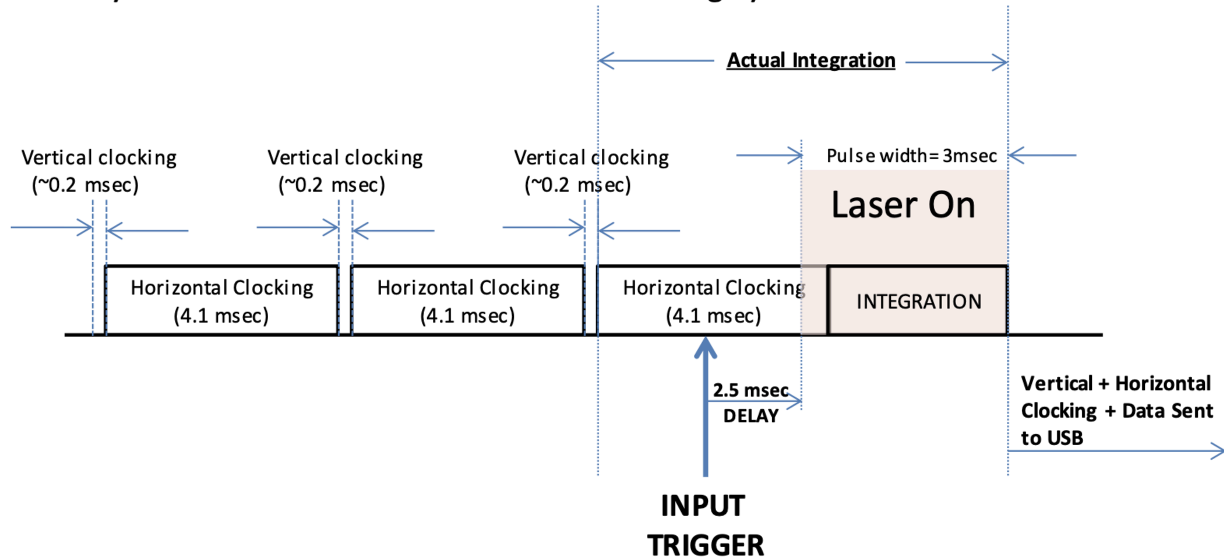
The acquisition state machine should proceed as follows:

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.



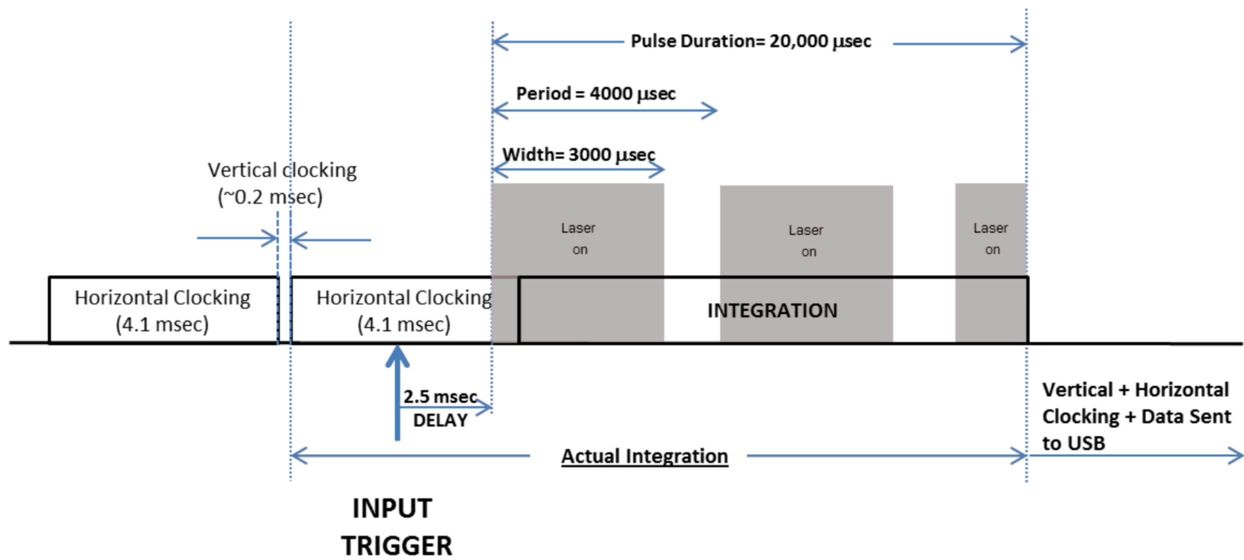
In the example above, the input trigger occurred during a horizontal clocking cycle. Since the vertical pixels are untouched, this can be counted as integration. In the above scenario, the firmware latches the input trigger, delays 2.5 ms, turns on the laser for 3 ms. So the actual integration would 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 `MOD_ENABLE = 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 a 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.

6.1 External Triggering

External triggering options vary by spectrometer, depending on the electronics and boards used for a specific model.

FX2-based Spectrometers (110008 PCB)

The external trigger is injected on pin 2 of connector J6 on the 110008 USB Main Board.

Appendix: Proposed Changes

- Add USB opcode to insert string into log.py debug stream
- Add USB opcode to poll for large bitmask containing all internal errors of interest (add same bitmask structure as GENERIC BLE notification in ENG-0120)
- Add opcodes to control GPIO and Current Supply on OEM Accessory Connector (Remote On/Off and Remote Interlock do not require USB API changes)

Appendix: CRC-8-CCITT Specification

CRC Calculation

The Cyclic Redundancy Check (CRC) byte for error detection uses the CRC-8-CCITT algorithm. This calculation produces an 8-but CRC value using the polynomial:

$$x^8 + x^2 + x + 1$$

The following code is a simple C implementation of this CRC (source):

```
static const uint8_t CRC_TABLE[256] = {
    0x00, 0x07, 0x0E, 0x09, 0x1C, 0x1B, 0x12, 0x15,
    0x38, 0x3F, 0x36, 0x31, 0x24, 0x23, 0x2A, 0x2D,
    0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65,
    0x48, 0x4F, 0x46, 0x41, 0x54, 0x53, 0x5A, 0x5D,
    0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5,
    0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
    0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85,
    0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
    0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2,
    0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
    0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2,
    0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 0x9A,
    0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32,
    0x1F, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0D, 0x0A,
    0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42,
    0x6F, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A,
    0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C,
    0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
    0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC,
    0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 0xD4,
    0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B, 0x7C,
    0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44,
    0x19, 0x1E, 0x17, 0x10, 0x05, 0x02, 0x0B, 0x0C,
    0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34,
    0x4E, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5C, 0x5B,
    0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63,
    0x3E, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B,
    0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14, 0x13,
    0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB,
    0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
    0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB,
    0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3
};

uint8_t crc8ccitt(const void * data, size_t size) {
     uint8_t val = 0;
```

```
    uint8_t * pos = (uint8_t *) data;
    uint8_t * end = pos + size;

    while (pos < end) {
        val = CRC_TABLE[val ^ *pos];
        pos++;
    }

    return val;
}
```