Wasatch Photonics

ENG-0001 USB FID ICD

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

Revision 1.15

Mar 18, 2022

Contents

Revision Log

Doc #	Date	Author	Description	Rev.
ENG-1	9/25/14	S. Trani	Initial Release	1.0
	8/11/15	J. Dreitzler	Add second tier commands, model configuration and feature identification	1.1
	10/6/15	J. Dreitzler	Corrected model configuration format Multiple changes to align commands for FX2 and ARM-based products	1.2
	10/20/15	J. Dreitzler	Added additional commands	1.3
	1/25/17	J. Traud	Formatting, cleanup, and preparation for public release.	1.4
	10/2/17	J. Traud	Removed customer specific references	1.5
	10/8/17	J. Traud	Removed legacy and development command references no longer valid in current firmware branch	1.6
	8/24/18	M. Zieg	Update to reflect current firmware	1.7
	10/18/19	M. Zieg R. Dickerson	Added battery commands Added Get/Set CCD Start/Stop Line	1.8
	March, 2020	M. Zieg	Added Raman Mode, Raman Delay and Laser Watchdog	1.9
	Feb 3, 2021	M. Zieg B. Williams	Added Area Scan, Accessory Connector, Even/Odd Gain/Offset	1.10
	Feb 15, 2021	M. Zieg	Refactored section 4 (commands) into structured categories	1.11
	Feb 25, 2021	M. Zieg B. Williams	Added Lamp Enable, Ambient Temperature, Fan Control; updated Modulation Linked to Integration, Modulation Pulse Delay	1.12
	Mar 3, 2021	M. Zieg	Updated GET_MOD_PERIOD endianness, refactored Modulation section	1.13
	Apr 14, 2021	M. Zieg N. Baron	Added untethered opcodes	1.14
	Nov 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	1.15

1 General Description

This document describes the USB (Universal Serial Bus) API (Application Programming Interface) for all USB-based Wasatch Photonics Raman and non-Raman (UVVIS, VIS, VISNIR, NIR) spectrometers, whether benchtop or micro form-factor. It *does not* apply to spectrometers intended for the OCT market such as the Cobra series.

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

Control commands are sent as Vendor Requests (hence the "VR_" suffix on some commands), and the response is read on endpoint 0. The key exception is when reading spectra (with or without triggering), in which the response data is returned on the bulk-output endpoint 2 (and 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 Hamamatsu silicon detector)
- `0x2000` (FX2 Hamamatsu InGaAs detector)
- `0x4000` (ARM-based spectrometer, e.g. SiG)

2.1 USB device drivers

Wasatch Photonics spectrometers communicate primarily via USB. There are many low-level USB drivers available, including the open-source libusb (Linux/MacOS) and libusb-win32 (Windows), Microsoft's WinUSB.sys, Cypress' CyUSB etc. Technically, any USB library can be used to communicate with Wasatch spectrometers by following the documentation of that driver and our USB hardware/firmware API. (For Windows, you'd need a custom .inf file to point to 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 "libusb Windows"), but may do so if/when performance and required features dictate.

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

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

If you are interested in developing your spectroscopy application against our high-level drivers (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 *bytes,
    int size,
    int timeout);
```

Following are explanations of each 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.
- *bytes*: up to 64-byte data packet associated with control message. Used in some set commands to represent higher order parameter bytes. Unless specified, can be sent as NULL pointer on FX2-based models, although ARM-based spectrometer commands assume it will hold an 8-byte buffer of zeros even if unused.
    - Note: Commands taking uint40 parameters (such as used for laser modulation) will typically send the *most*-significant of the 5 parameter bytes as the first (data[0]) byte of the data payload, while sending the least-significant bytes in the *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*: time before timing out if an error occurs in milliseconds.

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
ACCESSORY_ENABLE	0x39	0x38	bool		1					(Gen1.5)		
ACQUIRE_SPECTRUM	0xad		void				8	NA				response read back on bulk endpoint 2 (and 6 for 2048px detectors)
~~ACTUAL_FRAMES~~	~~0xe4~~		~~uint16~~		~~2~~			~~MSB~~				
~~ACTUAL_INTEGRATION_TIME~~	~~0xdf~~		~~uint24~~		~~3~~	~~6~~		~~LSB~~				~~response of 0xffffff indicates error~~
AMBIENT_TEMPERATURE	0x35		int16		2			MSB		(Gen1.5)		
AREA_SCAN_ENABLE		0xeb	bool					NA				
BATTERY_STATE	0x13		mask	Y	3			MSB		(XS)		
BATTERY_REG	0x14		uint16	Y	2			??		(XS)		
COMPILATION_OPTIONS	0x04			Y			8	LSB				
~~CONTINUOUS_ACQUISITION~~	~~0xcc~~	~~0xc8~~	~~bool~~		~~1~~			~~NA~~				
~~CONTINUOUS_FRAMES~~	~~0xcd~~	~~0xc9~~	~~uint8~~		~~1~~			~~NA~~				
DETECTOR_GAIN	0xc5	0xb7	float16		2			LSB				half-precision float (MSB is integer, LSB is fraction)
~~DETECTOR_GAIN_ODD~~	~~0x9f~~	~~0x9d~~	~~float16~~		~~2~~			~~LSB~~		~~(InGaAs)~~		~~On InGaAs sets gain on odd-numbered pixels~~
DETECTOR_OFFSET	0xc4	0xb6	uint16		2			LSB				

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
~~DETECTOR_OFFSET_ODD~~	~~0x9e~~	~~0x9c~~	~~uint16~~		~~2~~			~~LSB~~		~~(InGaAs)~~		~~On InGaAs sets offset on odd-numbered pixels~~
DETECTOR_ROI		0x25	uint16[4]	Y				NA		(XS)		
~~DETECTOR_SENSING_THRESHOLD~~	~~0xd1~~	~~0xd0~~	~~uint16~~		~~2~~			~~LSB~~				
DETECTOR_START_LINE	0x22	0x21	uint16	Y	2	2		LSB		(XS)		
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_THRESHOLD_SENSING_MODE~~	~~0xcf~~	~~0xce~~	~~bool~~		~~1~~			~~NA~~				
DFU_MODE		0xfe	void					NA		(ARM)		DFU = “Dynamic Firmware Upgrade”; configures STM32 to accept firmware updates via DfuSe Demonstrator
ERASE_STORAGE		0x26		Y						(UT)		
FAN_ENABLE	0x37	0x36	bool		1			NA		(Gen1.5)		
FEEDBACK		0x27		Y				NA		(UT)		
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				

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
HIGH_GAIN_MODE_ENABLE	0xec	0xeb	bool		1			NA		(InGaAs)		
~~HORIZONTAL_BINNING~~	~~0xbc~~	~~0xb8~~	~~enum~~		~~1~~			~~NA~~	~~(NONE, TWO_PIXEL, FOUR_PIXEL)~~	~~(ARM)~~		
INTEGRATION_TIME	0xbf	0xb2	uint24		3			LSB				sent as 32-bit word (LSW wValue, MSW wIndex, big-endian within each)
LAMP_ENABLE	0x33	0x32	bool		1			NA		(Gen1.5)		
LASER_ENABLE	0xe2	0xbe	bool		1			NA		*	*	Used as STROBE_ENABLE on Gen 1.5 non-Raman
LASER_INTERLOCK	0xef		bool		1			NA		(FX2)	TRUE	developmental
LASER_IS_FIRING	0x0d		bool	Y	1			NA		(FX2)	TRUE	developmental
~~LASER_RAMPING_MODE~~	~~0xea~~	~~0xe9~~	~~bool~~		~~1~~			~~NA~~		~~(ARM)~~	~~TRUE~~	
LASER_TEC_SETPOINT	0xe8	0xe7	uint12		1		8	NA		(ARM)	TRUE	setter takes value in range (63, 127)
LASER_TEMPERATURE	0xd5		uint16		2			LSB			TRUE	developmental
LASER_WATCHDOG	0x15	0x16	uint16	Y	2			??		(XS)	TRUE	developmental
LINE_LENGTH	0x03		uint16	Y	2			LSB				
~~MOD_DURATION~~	~~0xc3~~	~~0xb9~~	~~uint40~~		~~5~~			~~LSB~~			~~TRUE~~	
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.
MOD_LINKED_TO_INTEGRATION	0xde	0xdd	bool		1			NA				

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
MOD_PULSE_PERIOD	0xcb	0xc7	uint40		5			LSB				Used for laser power on Raman models, and continuous strobe in non-Raman Gen 1.5 models. * API differs significantly between FX2 and ARM; see detailed description
MOD_PULSE_DELAY	0xca	0xc6	uint40		5			LSB				
MOD_PULSE_WIDTH	0xdc	0xdb	uint40		5			LSB				
MODEL_CONFIG	0x01	0xa2 or 0x02	byte[]	*	64			MSB				
~~OPT_ACTUAL_INTEGRATION_TIME~~	~~0x0b~~		~~bool~~	~~¥~~	~~1~~		~~8~~	~~NA~~				
~~OPT_AREA_SCAN~~	~~0x0a~~		~~bool~~	~~¥~~	~~1~~		~~8~~	~~NA~~				~~developmental~~
~~OPT_CF_SELECT~~	~~0x07~~		~~bool~~	~~¥~~	~~1~~		~~8~~	~~NA~~				
~~OPT_DATA_HEADER_TAB~~	~~0x06~~		~~enum~~	~~¥~~	~~1~~		~~8~~	~~NA~~	~~(NONE, OCEAN OPTICS, WASATCH)~~			
~~OPT_HORIZONTAL_BINNING~~	~~0x0c~~		~~bool~~	~~¥~~	~~1~~			~~NA~~				
~~OPT_INTEGRATION_TIME_RESOLUTION~~	~~0x05~~		~~enum~~	~~¥~~	~~1~~			~~NA~~	~~(ONE MS, TEN MS, SWITCHABLE)~~			
~~OPT_LASER_TYPE~~	~~0x08~~		~~enum~~	~~¥~~	~~1~~			~~NA~~	~~(NONE, INTERNAL, EXTERNAL)~~			
~~OPT_LASER_CONTROL~~	~~0x09~~		~~enum~~	~~¥~~	~~1~~			~~NA~~	~~(MODULATION, TRANSITION POINTS, RAMPING)~~			
PIXEL_MODE		0xfd						NA		(XS)		developmental

Name	Getter	Setter	Data Type	2nd Tier	Read Len	Read Back Len	Fake Get Buf Len	Getter Endian	Enums	Supports	Req. Laser	Notes
RAMAN_DELAY	0x19	0x20	uint16	Y	2			NA		(XS)	TRUE	developmental
RAMAN_MODE	0x17	0x18	bool	Y	1			NA		(XS)	TRUE	developmental
RESET_FPGA		0xb5	void					NA				attempts to perform a runtime reset (power cycle) of the FPGA
SELECTED_ADC	0xee	0xed	enum		1			NA	(PRIMARY, SECONDARY)			Typically laser thermistor or photodiode if present
SHUTTER_ENABLE	0x31	0x30	bool					NA		(Gen1.5)		
STORAGE_BLOCK	0x25		byte[]	Y				NA		(UT)		
TRIGGER_DELAY	0xab	0xaa	uint24		3			LSB		(ARM)		Delay is in 0.5us, supports 24-bit unsigned value (about 8.3sec max)
TRIGGER_FEEDBACK		0x27		Y				NA		(UT)		
~~TRIGGER_OUTPUT~~	~~0xe1~~	~~0xe0~~	~~enum~~		~~1~~			~~NA~~	~~(LASER MODULATION, INTEGRATION ACTIVE PULSE)~~			
TRIGGER_SOURCE	0xd3	0xd2	enum		1			NA	(USB, EXTERNAL)			
UNTETHERED_CAPTURE_STATUS	0xd4		enum		1				(IDLE, DARK, WARMUP, SAMPLE, PROCESSING)	(XS)		

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.

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.

4.1.2 Get FPGA Firmware Revision (GET_FPGA_FIRMWARE_VERSION)

Reads the revision of the FPGA firmware code.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xB4	Request
2	wValue	2	0xXXXX	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 value < 0 if unsuccessful.

4.1.3 Get Sensor Line Length (GET_LINE_LENGTH)

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

Format

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

Response

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

4.1.4 Get FPGA Compilation Options (READ_COMPILATION_OPTIONS)

Gets the FPGA compilation options register.

Format

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

Response

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

~~4.1.5 Get Internal Frame Count (GET_ACTUAL_FRAMES)~~

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

This command is deprecated, and not regularly tested.

Format

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

Response

This command outputs two bytes containing the current frame count as a uint16 of unclear endianness.

~~4.1.6 Get Data Header or Tag (GET_OPT_DATA_HEADER_TAG)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

One byte containing:

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

4.2 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	0x0001	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 command was successful. Returns value 0 if unsuccessful.

4.3 Spectral Acquisition

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 Wasatch spectrometers with Hamamatsu detectors, an ACQUIRE command will instruct the spectrometer to START a NEW acquisition. That acquisition will not be available for readout over USB for at least the length of the current integration time, as the full integration will occur AFTER the ACQUIRE command is received.

On Sony IMX-based spectrometers, the sensor is continually acquiring in a "free-running" mode (but not attempting to send most acquisitions to the host over USB). On those spectrometers, 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.

In Multiple-ROI mode, the number of pixels output will be the sum of configured horizontal ROIs (so in the example given for SET_DETECTOR_ROI, 1001 + 1101 = 2102 pixels would be output, representing the vertically binned concatenated results of region 0 and 1 respectively).

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 for this command is not returned on the same "Control Endpoint 0" used to issue the command. 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, while 512-pixel detectors will return 1024 bytes, etc.

FX2-based models with 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.

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.

4.4 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 indiciate 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-signifiant 8 bits of value
6	wLength	2	0	Payload size

Integration time can be expressed as:

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

Response

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

4.4.2 Get Integration Time (GET_INTEGRATION_TIME)

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

Format

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

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

Response

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

~~4.4.3 Get Actual Integration Time (GET_ACTUAL_INTEGRATION_TIME)~~

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

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

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

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

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

This command is deprecated and not regularly tested.

Format

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

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

Response

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

~~4.4.4 Get Actual Integration Time Available (GET_OPT_ACTUAL_INTEGRATION_TIME)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

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

~~4.4.5 Get Integration Time Resolution (GET_OPT_INTEGRATION_TIME_RESOLUTION)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

One byte:

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

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

~~4.5.2 Set Detector Offset Odd (SET_DETECTOR_OFFSET_ODD)~~

Sets the detector offset for odd-numbered pixels (1, 3, 5…). See SET_DETECTOR_OFFSET for details.

This command is only available on InGaAs models.

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x9c	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 command was successful. Returns value < 0 if unsuccessful. FX2-based products return 1 for success and 0 if unsuccessful.

4.5.3 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.

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

4.5.4 Set Detector Gain Odd (SET_DETECTOR_GAIN_ODD)

Same as SET_DETECTOR_GAIN, but applies only to odd-numbered pixels (1, 3, 5…).

This command is only available on InGaAs models.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x9d	Request
2	wValue	2	Gain	Value
4	wIndex	2	0xXXXX	Index (n/a)
6	wLength	2	0	Payload size

Response

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

4.5.5 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 returns value < 0 if unsuccessful. FX2-based products return 1 if successful and 0 if unsuccessful.

~~4.5.6 Get Detector Offset Odd (GET_DETECTOR_OFFSET_ODD)~~

Same as GET_DETECTOR_OFFSET, but only applies to odd-numbered pixels (1, 3, 5…).

This command is only available on InGaAs models.

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x9e	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 returns value < 0 if unsuccessful. FX2-based products return 1 if successful and 0 if unsuccessful.

4.5.7 Get Detector Gain (GET_DETECTOR_GAIN)

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

~~4.5.8 Get Detector Gain Odd (GET_DETECTOR_GAIN_ODD)~~

Reads the detector gain for odd-numbered pixels, as an unsigned bfloat16 (seeSet Detector Gain Odd (SET_DETECTOR_GAIN_ODD)).

This command is only available on InGaAs models.

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x9f	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 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).

4.6.1 Laser Interlock Overview

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.

4.6.2 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.

4.6.3 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 (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.

4.6.4 Get Laser Temperature (GET_LASER_TEMPERATURE, aka GET_ADC)

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

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

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

You will note there is no "SET_LASER_TEMPERATURE" command. That is deliberate. The laser TEC setpoint is optimized in-factory to minimize mode-hopping and maximize performance and stability. Attempting to manually change the laser temperature is not recommended.

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.

4.6.5 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. Engineering documentation suggests this value should not be set outside the 7-bit range (0, 127) inclusive.

This value is used exclusively during manufacturing calibration to balance a hardware potentiometer (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 tunable or adjustable by customers.

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

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

Format

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

4.6.6 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

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

4.6.7 Get Laser Type Available (GET_OPT_LASER_TYPE)

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

Format

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

Response

One byte:

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

4.6.8 Get Laser Interlock (GET_LASER_INTERLOCK)

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

This command is developmental and not yet widely deployed.

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 1 (interlock circuit open, laser armed and ready to fire) on endpoint 0.

4.6.9 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 command 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.

This command is developmental and not yet widely deployed.

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.

~~4.6.10 Set Laser Power Ramping (SET_LASER_RAMPING_MODE)~~

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

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

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

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

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

This command is deprecated and not regularly tested.

Format

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

Response

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

~~4.6.11 Get Laser Power Ramping Mode (GET_LASER_RAMPING_MODE)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

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

~~4.6.12 Get Laser Control Type Available (GET_OPT_LASER_CONTROL)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

One byte:

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

4.7 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.

Non-Raman Gen 1.5 Spectrometers

On Non-Raman spectrometers with Gen 1.5 electronics, the same PULSE_PERIOD and PULSE_WIDTH described above are used to control the "continuous strobe" pin on the external accessory connector.

Uint40 Parameters

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.

4.7.1 Set Modulation Pulse Period (SET_MOD_PULSE_PERIOD)

This command sets the modulation period in microseconds. Typical values used are 100µs or 1000µs.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
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	Payload byte 0 contains bits 32..39 of 40-bit value (remaining 7 bytes are ignored)

4.7.2 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

Uint40 value in 5 little-endian bytes.

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

4.7.3 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	Payload byte 0 contains bits 32..39 of 40-bit value (remaining 7 bytes are ignored)

4.7.4 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

Uint40 value in 5 little-endian bytes.

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

4.7.5 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.

Non-Raman Gen 1.5

On Non-Raman spectrometers this command is used to dis/enable continuous strobe on the external accessory connector. If cont_strobe is disabled, the cont_strobe output pin on the accessory connector will remain "logic low". If cont_strobe is enabled, the pin will output a square wave pattern with the duty cycle specified by MOD_PULSE_PERIOD and MOD_PULSE_WIDTH.

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

4.7.6 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.

Non-Raman Gen 1.5

On Non-Raman Gen 1.5 spectrometers, this command returns whether continuous strobe is enabled.

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.

4.7.7 Set Modulation Pulse Delay (SET_MOD_PULSE_DELAY)

Raman Spectrometers

When laser modulation is enabled, and when laser modulation is "linked" to integration, this command selects the delay from the start of integration to the beginning of laser emitting (in µs).

Non-Raman Gen 1.5

On non-Raman Gen 1.5 spectrometers, this command affects the continuous strobe feature on the external accessory connector in the same manner described above.

If cont_strobe is enabled, and if MOD_LINKED_TO_INTEGRATION is enabled, then this delay will be inserted between the beginning of an integration (which will start when commanded, or after "trigger_delay" has elapsed following an external hardware trigger), and when the first strobe in a continuous series is set to begin. Subsequent continuous pulses will not be so delayed, until the integration ends (at which point the "linked" pulses will summarily cease, until the next integration begins).

Format

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

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

Response

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

4.7.8 Get Modulation Delay (GET_MOD_PULSE_DELAY)

The command reads the modulation delay (µs). See SET_MOD_DELAY for details.

Format

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

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

Response

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

~~4.7.9 Set Modulation Duration (SET_MOD_DURATION)~~

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

This command is deprecated and not actively tested.

Format

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

So Laser Modulation Duration can be expressed as:

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

Response

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

~~4.7.10 Get Modulation Duration (GET_MOD_DURATION)~~

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

This command is deprecated and not actively tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xC3	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

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

Response

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

4.7.11 Set Modulation Linked to Integration (SET_MOD_LINKED_TO_INTEGRATION)

Raman Spectrometers

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).

Non-Raman Gen 1.5

On Non-Raman systems, this setting determines whether continuous strobe is “linked” to integrations or not.

If the link is DISABLED, then the continuous strobe will continue to modulate according to its defined pulse period and pulse width regardless of integration operations. The generated output modulation will represent an unchanging wall-clock, unaffected by whatever else the spectrometer or detector are doing at the time.

If the link is ENABLED, then continuous strobe will NOT PULSE (will remain “logic-low”) when the detector is not integrating, and will RE-INITIALIZE (repeat its initial “delay” as per SET_ MOD_DELAY) each time a new integration begins. This allows every acquisition to have a deterministic number of pulses during every measurement with the same integration time, and for the “location” of those pulses (timing offset from the start of the acquisition) to be deterministic and repeatable.

This command is developmental and being redefined.

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

4.7.12 Get Modulation Linked to Integration Time (GET_MOD_LINKED_TO_INTEGRATION)

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.

4.8 Accessory Connector Control

These commands enable and disable the entire accessory connector, meaning they make individual accessory commands like LAMP_ENABLE, CONT_STROBE_ENABLE, TRIGGER_SOURCE etc possible.

4.8.1 Set Accessory Enable (SET_ACCESSORY_ENABLE)

This command enables or disables the bank of FPGA switches supporting the LAMP, STROBE and TRIGGER pins on the Gen 1.5 accessory connector.

This command only operates on a Non-Raman spectrometer with Gen 1.5 electronics.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x38	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

4.8.2 Get Accessory Enable (GET_ACCESSORY_ENABLE)

This command returns the enable status of the bank of FPGA switches supporting the LAMP, STROBE and TRIGGER pins on the Gen 1.5 accessory connector.

This command only operates on a Non-Raman spectrometer with Gen 1.5 electronics.

Format

Offset	**Field**	**Size**	**Value**	**Description**
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x39	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

One byte (0 accessory pins disabled, non-zero accessory pins enabled)

4.9 Lamp Control

Given the inherent safety issues in laser operation, it was decided not to re-use the existing "laser_enable" command to control external lamps for non-Raman Gen 1.5 spectrometers.

4.9.1 Set Lamp Enable (SET_LAMP_ENABLE)

This command will set the lamp_enable output pin on the Gen 1.5 accessory connector.

This command has no connection to other Gen 1.5 accessory connector outputs such as continuous strobe. Lamp_enable and cont_strobe_enable are orthogonal, and may be both enabled at the same time, or neither, or one may be enabled but not the other; they do not intercommunicate.

This command only operates on a Non-Raman spectrometer with Gen 1.5 electronics.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x33	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.9.2 Get Lamp Enable (GET_LAMP_ENABLE)

This command will return the current value of the lamp_enable pin on the Gen 1.5 accessory connector.

This command only operates on a Non-Raman spectrometer with Gen 1.5 electronics.

Format

Offset	**Field**	**Size**	**Value**	**Description**
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x32	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

One byte (0 lamp disabled, non-zero lamp enabled)

4.9.3 Set Strobe Enable (SET_STROBE_ENABLE)

On non-Raman spectrometers, this command enables or disables the strobe function on the external accessory connector.

On Raman spectrometers, this same opcode is used as SET_LASER_ENABLE.

Format

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

4.9.4 Get Strobe Enable (GET_STROBE_ENABLE)

On Gen 1.5 non-Raman spectrometers, this command returns the current enable status of the continuous strobe pin on the external accessory connector.

On Raman spectrometers, this same opcode is used for GET_LASER_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

One byte (0 strobe disabled, non-zero strobe enabled)

4.10 Detector Temperature Control

4.10.1 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.

4.10.2 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.

4.10.3 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:

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

Format

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

4.10.4 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:

$$\texttt{Uint16 raw\_dac\_value} = \text{round}(C_0 + C_1T + C_2T^2)$$

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.

4.10.5 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).

4.10.6 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.

4.10.7 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.

4.11 Trigger Control

4.11.1 Set Trigger Source (SET_TRIGGER_SOURCE)

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.

4.11.2 Get Trigger Source (GET_TRIGGER_SOURCE)

The command reads the trigger source for data acquisition.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD3	Request
2	wValue	2	0xXXXX	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.

4.11.3 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.

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

Format

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

Response

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

4.11.4 Get Trigger Delay (GET_TRIGGER_DELAY)

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

Format

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

~~4.11.5 Select Trigger Output (SET_TRIGGER_OUTPUT)~~

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

This command is deprecated, and not currently tested.

Format

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

~~4.11.6 Get Trigger Output (GET_TRIGGER_OUTPUT)~~

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

This command is deprecated, and not currently tested.

Format

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

Response

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

4.12 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.

4.12.1 Set High-Gain Mode Enable (SET_HIGH_GAIN_MODE_ENABLE)

This command turns high-gain mode on or off.

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

4.12.2 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

4.12.3 Get CF Select Available (GET_OPT_CF_SELECT)

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

It is expected that all spectrometers with an InGaAs detector (all models with PID 0x2000, and any InGaAs-based ARM) will have this feature enabled.

Format

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

Response

One byte:

- 0 = CF Select not available
- 1 = CF Select available

4.13 Battery Control

4.13.1 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: integral battery charge level (valid values 0-100)
- Byte 2: charging state (0 if falling e.g. discharging, non-zero if rising e.g. charging)

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

4.13.2 Get battery register (GET_BATTERY_REG)

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

Format

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

Response

Two bytes (nterpretation depending on register).

4.14 Raman Mode and Laser Watchdog

4.14.1 Set Raman Mode (SET_RAMAN_MODE)

This command is only available on XS-series spectrometers.

There are two modes provided for laser use:

- Manual Mode: Where the user manually turns the laser on and off. (Default at system boot)
- Raman Mode: Where the laser turns on before acquiring spectra and turns off after acquiring spectra.

Note that enabling Raman Mode and taking an acquisition is not sufficient to fire the laser. You must also explicitly enable the laser normally using SET_LASER_ENABLE. The normal command sequence is therefore:

1. SET_RAMAN_MODE -> 1 (laser will not fire)
2. SET_LASER_ENABLE -> 1 (laser will not fire)
3. ACQUIRE (laser will fire during integration, then switch off)
4. ACQUIRE (laser will fire during integration, then switch off)
5. SET_LASER_ENABLE -> 0 (laser still off)
6. SET_RAMAN_MODE -> 0 (laser still off)

Raman Mode is recommended whenever possible, for the following reasons:

- Maximizes battery performance by minimizing laser on-time
- Maximizes laser MTBF by minimizing lifetime use
- Minimizes sample burning
- Maximizes operator safety by minimizing laser on-time

These benefits are mitigated by the trade-off that in Raman Mode, it is key to recognize that every measurement will involve automatically turning on the laser. If non-Raman measurements are intended (such as emission, reflectance, absorbance or transmission spectroscopy), it would be both incorrect and dangerous to use Raman mode.

For this reason, Raman Mode is not the default power-on behavior, but its efficacious use by software is recommended to maximize system performance and longevity.

Note that both Manual and Raman modes are subject to the Laser Watchdog (see below), which will automatically deactivate the laser after the prescribed timeout regardless of how it was turned on.

Therefore, in Raman Mode, ensure that the Laser Watchdog timeout is greater than the integration time (at least one second greater is recommended, to account for warning LED windows). And likewise, in Manual mode, ensure the Laser Watchdog timeout is appropriate for whatever activity you are undertaking.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xFF	Request
2	wValue	2	0x16	Command
4	wIndex	2	0 = disabled 1 = enabled	Raman Mode
6	wLength	2	0	Payload size

Response

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

4.14.2 Get Raman Mode (GET_RAMAN_MODE)

Queries whether the system is currently in "Raman Mode" or not. (See SET_RAMAN_MODE for description.)

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	0x15	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns 1 if Raman Mode enabled, 0 otherwise.

4.14.3 Set Raman Delay (SET_RAMAN_DELAY)

This command (like Raman Mode) is only available on XS-Series spectrometers.

In Raman Mode, the spectrometer is responsible for turning the laser on at the beginning of an acquisition, and turning it off again at the end of the acquisition. However, some lasers require a warm-up delay to stabilize the output power before a repeatable (deterministic) measurement can be taken. This command lets the host configure that delay between the start of laser power, and the start of the actual integration.

The value is in milliseconds, which is internally converted to an appropriate clock rate based on the current integration time (internal resolution approximately 20µs depending on FPS).

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xFF	Request
2	wValue	2	0x20	Command
4	wIndex	2	milliseconds	Raman delay in ms (0-65535)
6	wLength	2	0	Payload size

Response

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

4.14.4 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 command was successful, non-zero if unsuccessful.

4.14.5 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).

4.14.6 Get Raman Delay (GET_RAMAN_DELAY)

Returns the current value of the Raman Delay (laser warm-up in Raman Mode).

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	0x19	Command
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns current Raman Delay in milliseconds (big-endian uint16).

4.15 Area Scan and Detector ROI

4.15.1 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 over-written 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.

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 command was successful, non-zero if unsuccessful.

4.15.2 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 command was successful, non-zero if unsuccessful.

4.15.3 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.

4.15.4 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 command was successful, non-zero if unsuccessful.

4.15.5 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.

4.15.6 Set Detector ROI (SET_DETECTOR_ROI)

For XS-Series spectrometers with advanced region-of-interest (ROI) control, this command allows the ROI for a single region to be set in the FPGA.

An ROI record contains several pieces of information:

- Region index (0, 1, etc): on a spectrometer supporting two detector ROIs, they will be numbered 0 and 1. This index is passed as the wIndex to the SET_DETECTOR_ROI command.
- ROI start line (row): y_0 coordinate of the ROI
- ROI stop line (row): y_1 coordinate of the ROI + 1
- ROI start pixel (column): x_0 coordinate of the ROI
- ROI stop pixel (column): x_1 coordinate of the ROI + 1

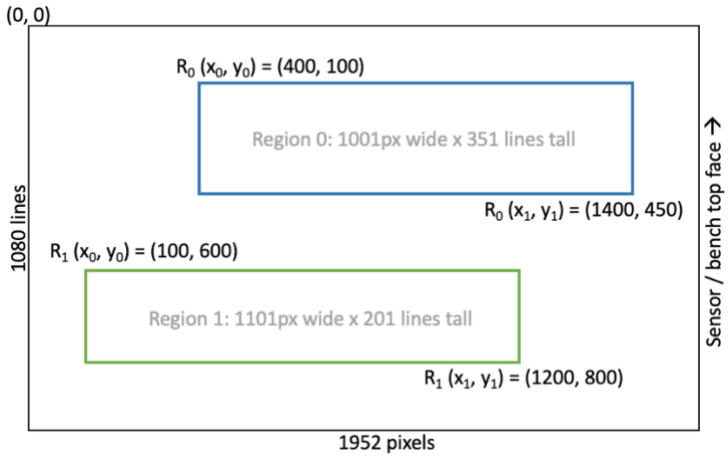
All of the above values are unsigned shorts (uint16). Valid ranges will be based on the installed detector, but for an IMX385, valid rows may be considered to fall in the 1080p range (0, 1079), and valid columns in the range (0, 1951).

Note that all ranges are defined as [start, stop), where the start value is *included* (closed) in the configured range, while the stop value is *excluded* (open). Therefore, the start/stop lines [300, 401) indicates 100 lines will be vertically binned (lines 300 through 400, but not 401).

The SET_DETECTOR_ROI payload will therefore comprise 8 bytes (all uint16 values are little-endian):

Offset	Size	Name	Datatype	Description
0	2	y_0	uint16	Start line
2	2	y_1	uint16	Stop line
4	2	x_0	uint16	Start pixel
6	2	x_1	uint16	Stop pixel

Example:



To configure the two ROIs (R_0 and R_1) on an IMX385 detector, you would send the following two SET_DETECTOR_ROI commands:

wIndex	wLength	Payload (y_0, y_1, x_0, x_1)
0	8	Logical: (100, 450, 400, 1400) Physical: [0x64, 0x00, 0xc2, 0x01, 0x90, 0x01, 0x78, 0x05]
1	8	Logical: (600, 800, 100, 1200) Physical: [0x58, 0x02, 0x20, 0x03, 0x64, 0x00, 0xb0, 0x04]

The configured ROIs must obey these rules:

- must be specified in ascending row order ($R_0Y_0 < R_0Y_1 < R_1Y_0 < R_1Y_1$)
- ROIs cannot overlap

To "delete" a configured ROI, you would set a fresh command to configure the specific ROI, using values of zero for all parameters.

Example: to delete R_1 (the "second" ROI, after R_0), send the following Control Message:

```
sendCmd(bmRequestType=0x40,    # HOST_TO_DEVICE
        bRequest=0xff,         # SECOND_TIER_COMMAND
        wValue=0x25,           # SET_DETECTOR_ROI
        wIndex=0x01,           # ROI INDEX
        len=8,                 # PAYLOAD SIZE
        data=[ 0, 0, 0, 0, 0, 0, 0, 0 ]) # ALL VALUES ZERO
```

This command is only available on specific XS-Series models.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xFF	Request
2	wValue	2	0x25	Command
4	wIndex	2	Region	Which ROI region (0-3)
6	wLength	2	8	Payload size

4.15.7 Set Pixel Mode (SET_PIXEL_MODE)

IMX detectors can operate in either 10-bit or 12-bit "pixel depth." Similarly, they can be populated from an ADC with either 10-bit or 12-bit dynamic range.

Pixel Width (OD) is set by bit 0, and the ADC (AD) is set by bit 1. In both cases, a zero indicates 10-bit mode, and a one indicates 12-bit mode.

This allows four possible pixel modes:

Mode (dec)	Mode (binary)	ADC (AD)	Pixel Width (OD)
0	00b	10-bit	10-bit
1	01b	10-bit	12-bit
2	10b	12-bit	10-bit
3	11b	12-bit	12-bit

Note that the detector ROI is vertically binned into a uint16[] spectrum regardless of individual pixel mode, so pixel mode will not change the dimensions or datatype of the output spectra.

This command is only available on specific XS-Series models.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xfd	Request
2	wValue	2	0-3	Pixel Mode (b00, b01, b10 or b11)
4	wIndex	2	Region	Which ROI region (0, 1…n)
6	wLength	2	8	Payload size

~~4.15.8 Get Area Scan Available (GET_OPT_AREA_SCAN)~~

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

This command is deprecated and not regularly tested. All non-InGaAs models are expected to support area scan mode.

Format

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

Response

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

4.16 Shutter Control

4.16.1 Set Shutter Enable (SET_SHUTTER_ENABLE)

This command enables (closes, obstructs light) or disables (opens, passes light) the internal shutter in appropriately equipped non-Raman spectrometers with Gen 1.5 electronics and a suitable shutter module.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x30	Request
2	wValue	2	0 (disable) or 1 (enable)	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.16.2 Get Shutter Enable (GET_SHUTTER_ENABLE)

This queries the spectrometer to determine whether the integrated shutter is currently enabled (closed, obstructing light) or disabled (open, transmitting light).

This command is only available on non-Raman spectrometers with Gen 1.5 electronics and an appropriate shutter module.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x31	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns a single byte: 1 if shutter enabled (closed, obstructing light), 0 otherwise.

4.17 Fan Control

4.17.1 Set Fan Enable (SET_FAN_ENABLE)

This command enables (spins) the fan in suitably equipped non-Raman spectrometers with Gen 1.5 electronics and an integrated fan.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0x36	Request
2	wValue	2	0 (disable) or 1 (enable)	Value
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.17.2 Get Fan Enable (GET_FAN_ENABLE)

This queries the spectrometer to determine whether the integrated fan is currently enabled.

This command is only available on spectrometers with Gen 1.5 electronics and an appropriate fan module.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0x37	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

Returns a single byte: 1 if fan enabled, 0 otherwise.

4.18 Ambient Temperature

4.18.1 Get Ambient Temperature (GET_AMBIENT_TEMPERATURE)

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

4.19 Untethered Devices

These commands are used to administer untethered devices which can operate independent of USB or BLE control.

4.19.1 GET_STORAGE_BLOCK

This command is functionally similar to GET_MODEL_CONFIG, but references a device's onboard FRAM non-volatile storage rather than EEPROM. The caller is expected to know the maximum supported range of 64-byte addressable pages.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xff	Second tier command
2	wValue	2	0x25	Value
4	wIndex	2	Page index	Range variable by model
6	wLength	2	64	Payload size

Response

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

4.19.2 ERASE_STORAGE

This command erases storage on supported devices

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xff	2nd-tier command
2	wValue	2	0x26	Request
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

4.19.3 TRIGGER_FEEDBACK

On devices which have some form of direct user feedback, this allows a connected host to trigger (generate, or initiate) a pre-configured response by integral value. Consult device documentation for the supported range of configured sequences on a particular device.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xff	2nd-tier command
2	wValue	2	0x27	Request
4	wIndex	2	Index	Index of pre-configured sequence or UX pattern
6	wLength	2	0	Payload size

Response

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

4.19.4 UNTETHERED_CAPTURE_STATUS / POLL_DATA

The command returns a value indicating the current untethered capture 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
1	1	0 = IDLE 1 = dark measurement 2 = laser warmup 3 = sample measurement 4 = processing

4.20 Board State

4.20.1 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.

4.20.2 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

4.20.3 Set Feedback (SET_FEEDBACK)

On devices which have some form of direct user feedback, this allows a connected host to initiate a pre-configured response by integral value. Consult device documentation for the supported range of configured feedback sequences on a particular device. (wIndex may be internally coded to support repetitions of a particular sequence.)

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xff	2nd-tier command
2	wValue	2	0x27	Request
4	wIndex	2	0xAABB	AA (MSB): feedback function (e.g. 0-6) BB (LSB): repetitions (0-255)
6	wLength	2	0	Payload size

~~4.21 Horizontal Binning Commands (Deprecated)~~

~~4.21.1 Select Horizontal Binning (SET_HORIZONTAL_BINNING)~~

The command selects the type of horizontal binning performed on the sensor data, if any. 0 selects no binning. 1 selects two-pixel binning. 2 selects four-pixel binning. On reset no binning is selected.

This command is deprecated and not regularly tested.

Format

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

Response

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

~~4.21.2 Get Horizontal Binning (GET_HORIZONTAL_BINNING)~~

The command returns the selected horizontal binning mode.

This command is deprecated and not currently tested.

Format

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

Response

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

~~4.21.3 Get Horizontal Binning Available (GET_OPT_HORIZONTAL_BINNING)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

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

~~4.22 Sensor Data Threshold Commands (Deprecated)~~

These commands were added in the past for legacy products, but are not regularly tested or verified in our current firmware. They may work; they may not; they may produce undefined behavior. Any could be "resurrected," updated and revalidated if customer requirements merit.

~~4.22.1 Set Detector Data Threshold Sensing (SET_DETECTOR_THRESHOLD_SENSING_MODE)~~

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

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xCE	Request
2	wValue	2	0 = disable 1 = enable	Value
4	wIndex	2	0xXXXX	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.

~~4.22.2 Get Detector Data Threshold Sensing Mode (GET_DETECTOR_THRESHOLD_SENSING_MODE)~~

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

This command is deprecated, and not regularly tested.

Format

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

Response

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

~~4.22.3 Set Detector Data Sensing Threshold Level (SET_DETECTOR_SENSING_THRESHOLD)~~

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

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xD0	Request
2	wValue	2	Threshold	Value (uint16)
4	wIndex	2	0xXXXX	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.

~~4.22.4 Get Detector Data Sensing Threshold (GET_DETECTOR_SENSING_THRESHOLD)~~

The command reads the detector data sensing threshold.

This command is deprecated, and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xC0	Device → Host
1	bRequest	1	0xD1	Request
2	wValue	2	0xXXXX	Ignored
4	wIndex	2	0xXXXX	Ignored
6	wLength	2	0	Payload size

Response

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

~~4.23 Continuous Acquisition (Deprecated)~~

~~4.23.1 Enable/Disable Continuous Acquisition (SET_CONTINUOUS_ACQUISITION)~~

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

This command is deprecated and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC8	Request
2	wValue	2	0 = disabled 1 = enabled	Value
4	wIndex	2	0xXXXX	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.

~~4.23.2 Get Continuous Acquisition (GET_CONTINUOUS_ACQUISITION)~~

This command reads whether continuous acquisition mode is enabled.

This command is deprecated and not regularly tested.

Format

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

Response

Returns 0 if disabled, 1 if enabled.

~~4.23.3 Set Continuous Frame Count (SET_CONTINUOUS_FRAMES)~~

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

This command is deprecated and not regularly tested.

Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host → Device
1	bRequest	1	0xC9	Request
2	wValue	2	frames	Number of frames (0-255)
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.

~~4.23.4 Get Continuous Frame Count (GET_CONTINUOUS_FRAMES)~~

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

This command is deprecated and not regularly tested.

Format

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

Response

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

5 Acquisition Workflows

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.

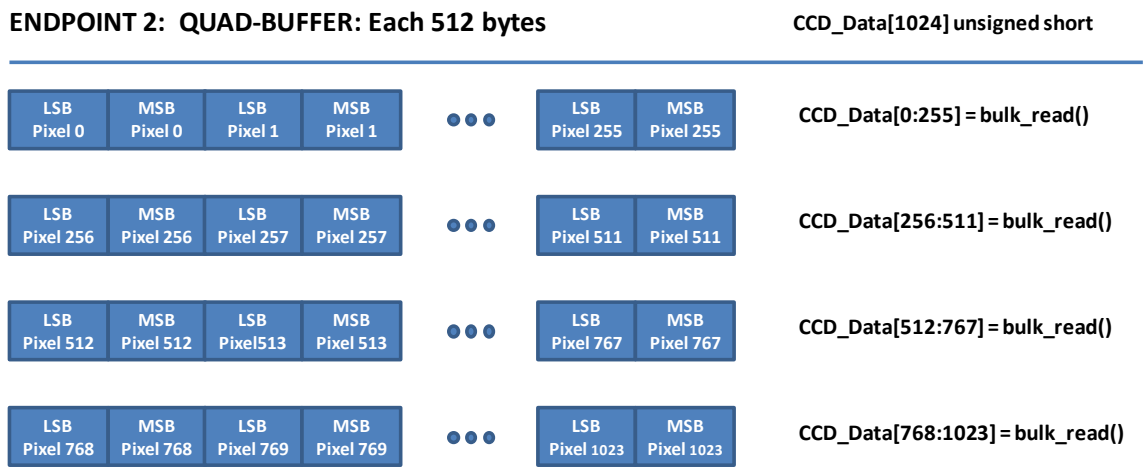
5.1 Spectral Acquisition

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). The external trigger is injected on pin 2 of USB Board connector J6.

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

5.4 Externally Triggered Acquisition

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

An externally triggered capture event would typically follow these steps (these assume you've found and gotten a handle to an open USB device – See 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.

5.4.1 SPECIFIC EXAMPLE

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

1. 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

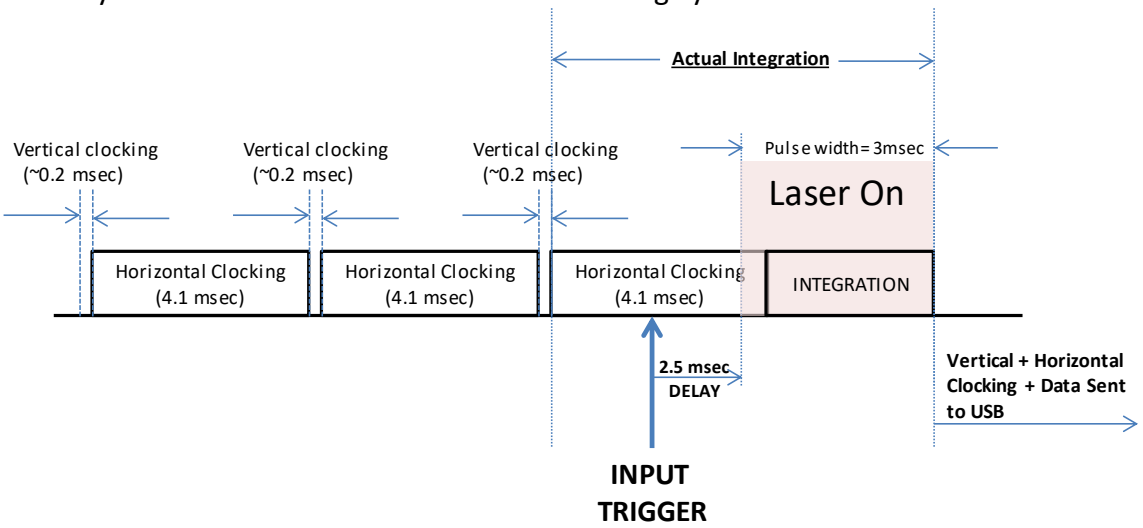
The acquisition state machine should proceed as follows:

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.



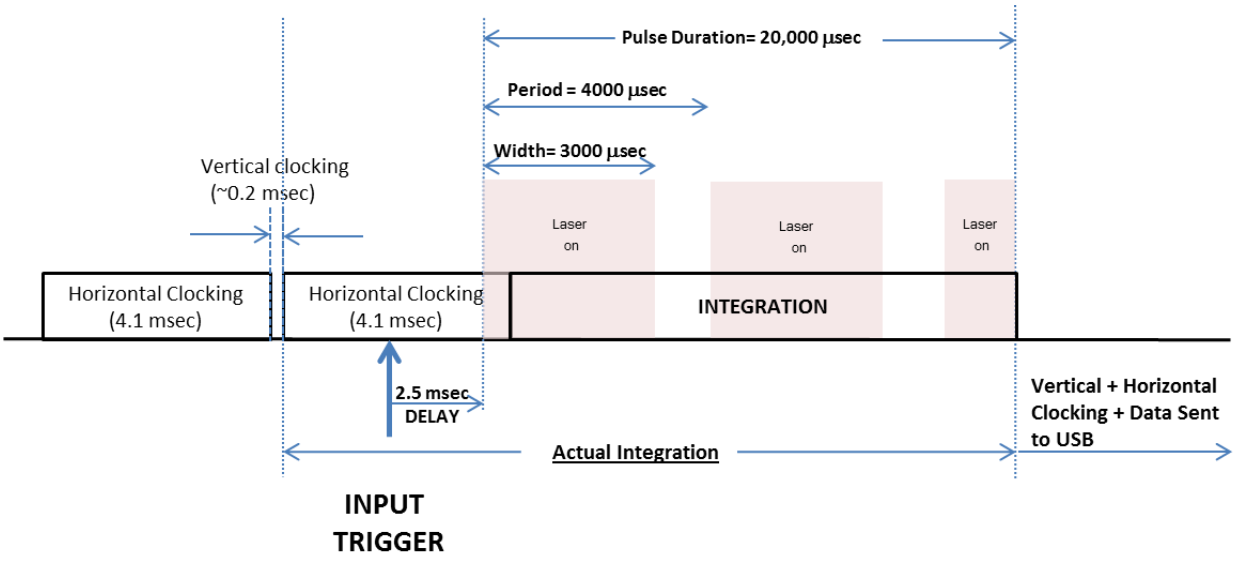
In the example above, the input trigger occurred during a horizontal clocking cycle. Since the vertical pixels are untouched, this can be counted as integration. In the above scenario, the firmware latches the input trigger, delays 2.5 ms, turns on the laser for 3 ms. So the actual integration would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