Polar Measurement Data

General Description

Polar BLE SDK allows the possibility to stream either raw sensor data (accelerometer, ppg, gyro, ...) or computed data (Heart rate, PPI, ...) over BLE. Limitation is that the mobile application needs to keep the connection alive at all time to not lose data. Some B2B customers were interested about a feature where data could be stored in the device memory and synced later to mobile using Polar SDK when BLE connection is available.

Disclaimer

This document describes the current status of starting and stopping online recording, and parsing the online recording data to human readable form. This is not any kind of official specification. Polar Electro reserves all rights to modify any Polar SDK component at any time without any external communication or other prior notice. Polar Electro does not guarantee in any way that this document is compatible with any future Polar SDK release.

1 Start online recording

Online measurement can be implemented using Polar Measurement Data service (PMD). You will need Bluetooth session with the target device. Establish a Bluetooth session with the device. Then fetch instance of BlePMDClient using UUID "PMD service" from Table 3 from session(BleGattBase). That instance of BlePMDClient is being used in start the measurement with the connected device. Start the measurement by sending a PMD Control Point Command to the device. See Table 4. for control point commands.

To start the measurement a ByteArray containing information of the measurement request for PMD Control Point Command is required. Allocate the size of the ByteArray as the combined size of the "firstByte" and the size of the serialized settings object. To create that ByteArray set the first byte to 0 for online measurement. Next add the serialized settings for the measurement to the ByteArray. For measurement types "PPI" and "HR" settings are set to null. For other measurement types find settings by sending PMD Control Point command to the device. See Table 3.

```
sendControlPointCommand(1, requestByte.toByte())
```

where requestByte is result of bitwise OR operation of bitfield value of Recording Type from Table 1 shifted left by 7 as bit value and the measurement type from Table 2. For example for Accelerometer measurement requestByte is in online measurement,

```
recordingType = 0,
measurementType = 2,
requestByte = (0 shl 7) | 2 = 2
```

Now you can construct the measurement request ByteArray as described in the above paragraph and use it in the PMD Control Point Command.

```
sendControlPointCommand(2, requestBytesArray)
```

where,

- requestBytesArray, is the serialialized requestByte

Recording type	First Byte
Online	0
Offline	1

Table 1: Recording types

2 Stop online recording

To stop the online recording send PMD Control Point command "Stop measurement" to the device.

```
sendControlPointCommand(3, measurementType as ByteArray)
```

where measurementType is the value from Table 1

Measurement type	Value
ECG	0u
PPG	1u
ACC	2u
PPI	3u
GYRO	5u
MAGNETOMETER	6u
SDK$_{M}ODE$	9u
LOCATION	10u
PRESSURE	11u
TEMPERATURE	12u

Table 2: Measurement types

Description	UUID
PMD Service	FB005C80-02E7-F387-1CAD-8ACD2D8DF0C
PMD Control point	FB005C81-02E7-F387-1CAD-8ACD2D8DF0C8
PMD Data MTU Charasteristic	FB005C82-02E7-F387-1CAD-8ACD2D8DF0C8

Table 3: Measurement UUIDs

Control point command	Value
Get measurement settings	1
Request measurement start	2
Request measurement stop	3

Table 4: Control point command

3 PMD measurement settings

The following settings come in as a response to the measurement settings request. These values are used when parsing the measurement data. The conversion factor is mandatory to use when parsing measurement data, otherwise the sample values are not correct.

Setting type	Name	Size(Bytes)	Type	Usage
0	Sample rate (Hz)	2	uint16	Get Measurement Settings Response Request measurement start
1	Resolution (bits)	2	uint16	Get Measurement Settings Response Request measurement start Needed for delta compression decoding
2	Range ((+/-)(Unit))	2	uint16	Get Measurement Settings Response Request measurement start
3	Range (milliUnit)	8	uint16	Get Measurement Settings Response Request measurement start
4	Number of channels	1	uint8	Get Measurement Settings Response Request measurement start Needed for the delta compression decoding

5	Conversion factor	4	IEEE 754 single-precision binary floating-point format.
			Fraction = bit 0 ... 22
			Exponent = bit 23 ... 30
			Sign = bit 31
				Get Measurement Settings Response
				Request measurement start
				Factor to convert relative value to absolute value

Table 5: PMD measurement settings

4 Parsing the online data

4.1 Data structure

Device sends measured online data and that data has to be parsed to a human readable format. The incoming data may be uncompressed or compressed, depending on the measurement type and device. Whether the data is compressed or uncompressed can be found out by checking the MSB of the dataframe with bitwise operation 0x80 is 1. This document describes how to parse both uncompressed and compressed data. Note that epoch year for timestamps is 2000 January 1st 00:00:00 UTC, value in nanoseconds accumulated to the epoch. The data frame is sent over BLE in the form of notifications on PMD Data MTU Characteristic. PMD Data MTU Characteristic is a byte array with measurement type, 8-bit ring counter and payload data defined for each PMD Measurement Type as described in Table 6.

Measurement type	data[0] (ACC, HR etc)	
Timestamp	data[1]...data[8]	
FrameType	data[9]	
Frame data content (delta frames)	data[10] ... data[size - 1]	

Table 6: Measurement data structure

4.2 Frame types

4.2.1 Acceleration frame types

Byte	Size	Name	Description
0	1	X value	8-bit signed value (G for Verity Sense, mG for other devices)
1	1	Y value	8-bit signed value (G for Verity Sense, mG for other devices)
2	1	Z value	8-bit signed value (G for Verity Sense, mG for other devices))

Table 7: Acceleration 8-bit, $TYPE_0$

Byte	Size	Name	Description
0	2	X value	16-bit signed value (mG)
2	2	Y value	16-bit signed value (mG)
4	2	Z value	16-bit signed value (mG)

Table 8: Acceleration 16-bit, $TYPE_1$

Byte	Size	Name	Description
0	3	X value	24-bit signed value (mG)
3	3	Y value	24-bit signed value (mG)
6	3	Z value	24-bit signed value (mG)

Table 9: Acceleration 24-bit, $TYPE_2$

4.2.2 ECG frame

Byte	Size	Name	Description
1 … n	3	sample value (µV)	24-bit signed value (µV)

Table 10: ECG 24-bit, TYPE$_0$

4.2.3 Gyroscope frame types

Byte	Size	Name	Description	
0 - 1	2	X-axis	16-bit int	
2 - 3	2	Y-axis	16-bit int	
4 - 5	2	Z-axis	16-bit int	

Table 11: Gyroscope 3D Sample Data, TYPE$_0$

Byte	Size	Name	Description
0 - 3	4	X-axis	32-bit float [degrees/s]. IEEE 754 single-precision binary floating-point format.
4 - 7	4	Y-axis	32-bit float [degrees/s]. IEEE 754 single-precision binary floating-point format.
8 - 11	4	Z-axis	32-bit float [degrees/s]. IEEE 754 single-precision binary floating-point format.

Table 12: Gyroscope Angular Rate Sample Data, TYPE$_1$

4.2.4 Magnetometer frame types

Byte	size	Name	Description	
0 - 1	2	X-axis	16-bit int	
2 - 3	2	Y-axis	16-bit int	
4 - 5	2	Z-axis	16-bit int	

Table 13: Magnetometer 3D Sample Data, TYPE$_0$

Byte	Size	Name	Description
0 - 1	4	X-axis	16-bit int [milligauss]
2 - 3	4	Y-axis	16-bit int [milligauss]
4 - 5	4	Z-axis	16-bit int [milligauss]
6 - 7	4	Calibration status	8-bit int [0=unknown, 1=poor, 2=ok, 3=good]

Table 14: Magnetometer Compass Sample Data, TYPE$_0$

4.2.5 Photoplethysmography (PPG) frame types

Byte	Size	Name	Description	
0 - 2	3	PPG0	16-bit int	
3 - 5	3	PPG1	16-bit int	
6 - 8	3	PPG2	16-bit int	
9 - 11	3	Ambient	16-bit int	

Table 15: PPG Data, TYPE$_0$

Byte	Size	Name	Description
0	1	NUMINT$_TS1$	8-bit uint
...	1	NUMINT$_TS1$	8-bit uint
11	1	NUMINT$_TS1$	8-bit uint
12	1 (3-bit)	TIA$_GAIN_CH1_TS1$	3-bit uint
13	1 (3-bit)	TIA$_GAIN_CH2_TS1$	3-bit uint
14	1 (3-bit)	TIA$_GAIN_CH1_TS2$	3-bit uint
15	1 (3-bit)	TIA$_GAIN_CH2_TS2$	3-bit uint
...			
32	1 (3-bit)	TIA$_GAIN_CH1_TS11$	3-bit uint
33	1 (3-bit)	TIA$_GAIN_C2_TS11$	3-bit uint
34	1 (3-bit)	TIA$_GAIN_CH1_TS12$	3-bit uint
35	1 (3-bit)	TIA$_GAIN_C2_TS12$	3-bit uint

Table 16: PPG Data, TYPE$_4$

Byte	Size	Name	Description
0	4	Operation mode	32-bit signed int

Table 17: PPG Data, TYPE$_5$

Byte	Size	Name	Description	
0	8	Sport id	64-bit uint	

Table 18: PPG Data, TYPE$_6$

Byte	Size	Name	Description
0	3	PPG0	24-bit signed int
3	3	PPG1	24-bit signed int
...	3	PPGx	24-bit signed int
45	3	PPG15	24-bit signed int
48	3	Status	24-bit signed int. bit 0: status for PPG0 channel. bit 15 status for PPG15 channel

Table 19: PPG Data, TYPE$_7$

Byte	Size	Name	Description
0	3	PPG0	24-bit signed int
3	3	PPG1	24-bit signed int
...	3	PPGx	24-bit signed int
69	3	PPG23	24-bit signed Int
72	3	Status	24-bit signed int. bit 0: status for PPG0 channel. bit 23 status for PPG23 channel

Table 20: PPG Data, TYPE$_8$

Byte	Size	Name	Description
0	1	NUMINT$_TS1$	8-bit uint
...	1	NUMINT$_TS1$	8-bit uint
11	1	NUMINT$_TS1$	8-bit uint
12	1 (3-bit)	TIA$_GAIN_CH1_TS1$	3-bit uint
13	1 (3-bit)	TIA$_GAIN_CH2_TS1$	3-bit uint
14	1 (3-bit)	TIA$_GAIN_CH1_TS2$	3-bit uint
15	1 (3-bit)	TIA$_GAIN_CH2_TS2$	3-bit uint
...			
32	1 (3-bit)	TIA$_GAIN_CH1_TS11$	3-bit uint
33	1 (3-bit)	TIA$_GAIN_C2_TS11$	3-bit uint
34	1 (3-bit)	TIA$_GAIN_CH1_TS12$	3-bit uint
35	1 (3-bit)	TIA$_GAIN_C2_TS12$	3-bit uint

Table 21: PPG Data, TYPE$_9$

4.2.6 Peak-to-Peak Interval (PPI) data

Byte	Size	Name	Description	
0	1	Heart rate	int	
1	2	PP interval	PPi in milliseconds, int	
3	2	PP error estimate	Expected estimation error in milliseconds	
5	1	PP Flags	bit 0: 1 if PP measurement is not valid. bit 1: 0 if poor/no skin contact. bit 2: 1 if sensor contact is not supported. bit 3: Reserved for future use	

Table 22: PPi Data, TYPE$_0$

4.2.7 Pressure data

Byte	Size	Name	Description	
0 - 3	4	Pressure	32-bit float value (*). Unit hPa (millibar)	
4 - 7	4	Pressure	32-bit float value (*). Unit hPa (millibar)	
...	4	Pressure	32-bit float value (*). Unit hPa (millibar)	
.				

Table 23: Pressure Data, TYPE$_0$

4.2.8 Temperature data

Byte	Size	Name	Description	
0	4	Temperature	32-bit float value, Celcius. IEEE 754 single-precision binary floating-point format.	
1	4	Temperature	32-bit float value, Celcius. IEEE 754 single-precision binary floating-point format.	
...	4	Temperature	32-bit float value, Celcius. IEEE 754 single-precision binary floating-point format	
.				

Table 24: Temperature Data, TYPE$_0$

4.2.9 Parse meta data

In order to find out which kind of measurement type, frame type and data compression one has to read meta data from the incoming byte array first. Determine data frame type from frame type value (data[9]) by bitwise AND operation with mask 0x7F. The data frame type describes the bit size of the data as well as the data type. Use chapter 3.2 to find out the data type characteristics. Find out if the frame is compressed or uncompressed frame by bitwise AND operation with frame type Byte and mask 0x80. Measurement type can be parsed from data[0] using bitwise AND operation with bitmask 0x3F. Data frame content is the remaining payload data starting from index at 10, from data[10] until the end of the payload data. Note! When parsing samples (compressed or uncompressed) remember to multiply the each sample value with the conversion factor described in Table 5.

4.3 Parse data frame

4.3.1 Parse uncompressed data frame

In uncompressed data frame the data can be read starting from the beginning of the frame, sample by sample, to the end of the frame data. Note that each sample may or may not have several values, depending on the measurement, data and frame type. Thus, one must use a step value of the size of the given frame type in Bytes when processing the data. For example step value for 16-bit acceleration data is 2 (size 2 Bytes), and each sample has 3 values. Use chapter 3.2 to find out the data type characteristics.

4.3.2 Parse compressed data frame

Compressed data has been constructed as delta compressed data where the previous value in data is used to calculate the next value as a sum of the two adjacent values (previous + next). The first value is the reference value.

Delta frame has a size (bits) determined by the device. The Delta frame size may differ. Also the number of samples in the delta frame may differ. You will find both the delta frame size and the sample count in front of the delta frame. Initially the delta frame size is at index (channels * ceil(resolution / 8.0)) + 1 and sample count at index (channels * ceil(resolution / 8.0)) + 2. Where resolution in is always in full Bytes. For example, 511 decimal value requires 9 bit expression that will use 2 Bytes.

4.3.3 Delta frame

Reference sample is the first ("seed") sample and it is being used in calculation of the subsequent delta samples. So the current delta sample will be summed up with the previous sample, and so on.

Calculate the reference sample. First, define mask for PMD data field encoding of signed int (negative numbers may be coming in) as mask = -0x1 shl resolution, where resolution is for example 2 (Bytes) for 14 bit value. Then take a chunk of values where number of values is determined but the “Resolution in bytes”. For example if resolution in Bytes is 2 Bytes, take two numbers for each chunk. Number of chunks is equal to the number of used channels. For example in acceleration measurement the number of channels three, thus three chunks. Now 2x3=6 first values are used in reference sample calculation for acceletometer data.

Next convert each chunk to signed or unsigned integer depending on the frame type (see table x):

bitmask = -0x1 shl resolution - 1

Iterate throught the values in chunk.

1. Signed integer (initially 0u) is the bitwise or together with the result of current iteration value as unsigned integer (convert value first to unsigned Byte) shift left by the current interation index multiplied by 8 (as the values are 8 bit). If resulted sample value bitwise AND bitmask is less than zero then the resulted sample is result of bitwise OR (0xFFFFFFFFu shl chunk.size * 8).

2. Convert chunk to Unsigned integer (initially 0u) is the bitwise or together with the result of current iteration value as unsigned integer (convert value first to unsigned Byte) shift left by the current interation index multiplied by 8 (as the values are 8 bit).

Each delta frame must go through operation where each value in frame is being transformed into binary format. Then each sample (whose length of the delta lenght), in binary format, is iterated through. If value currently in iteration is 1 the sample value (initially 0) is ((value bitwise OR 0x01) shift left 1), otherwise value is ((value bitwise or 0x00) shift left 1). (Zero remains zero, one remains one.) Finally, if the resulted value is not zero, the resulted value is pushed through bitwise OR operation (Int.MAX_VALUE shl bitWidth - 1) Now you can calculate the actual values by summing up the two adjacent values.

5 Sequence diagrams for typical operations

5.1 Read Features from device

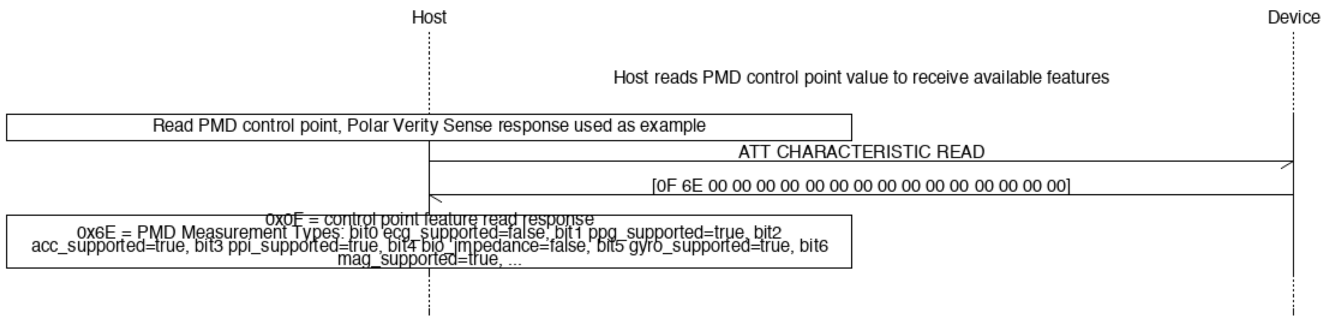

Host
Device
Host reads PMD control point value to receive available features
Read PMD control point, Polar Verity Sense response used as example
ATT CHARACTERISTIC READ
[0F 6E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00]
0x0F = control point feature read response
0x6E = PMD Measurement Types: bit0 ecg_supported=false, bit1 ppg_supported=true, bit2 acc_supported=true, bit3 ppi_supported=true, bit4 bio_impedance=false, bit5 gyro_supported=true, bit6 mag_supported=true, ...

Figure 1: Example on how read the Polar Measurement data features that the device supports

5.2 Request Stream Settings

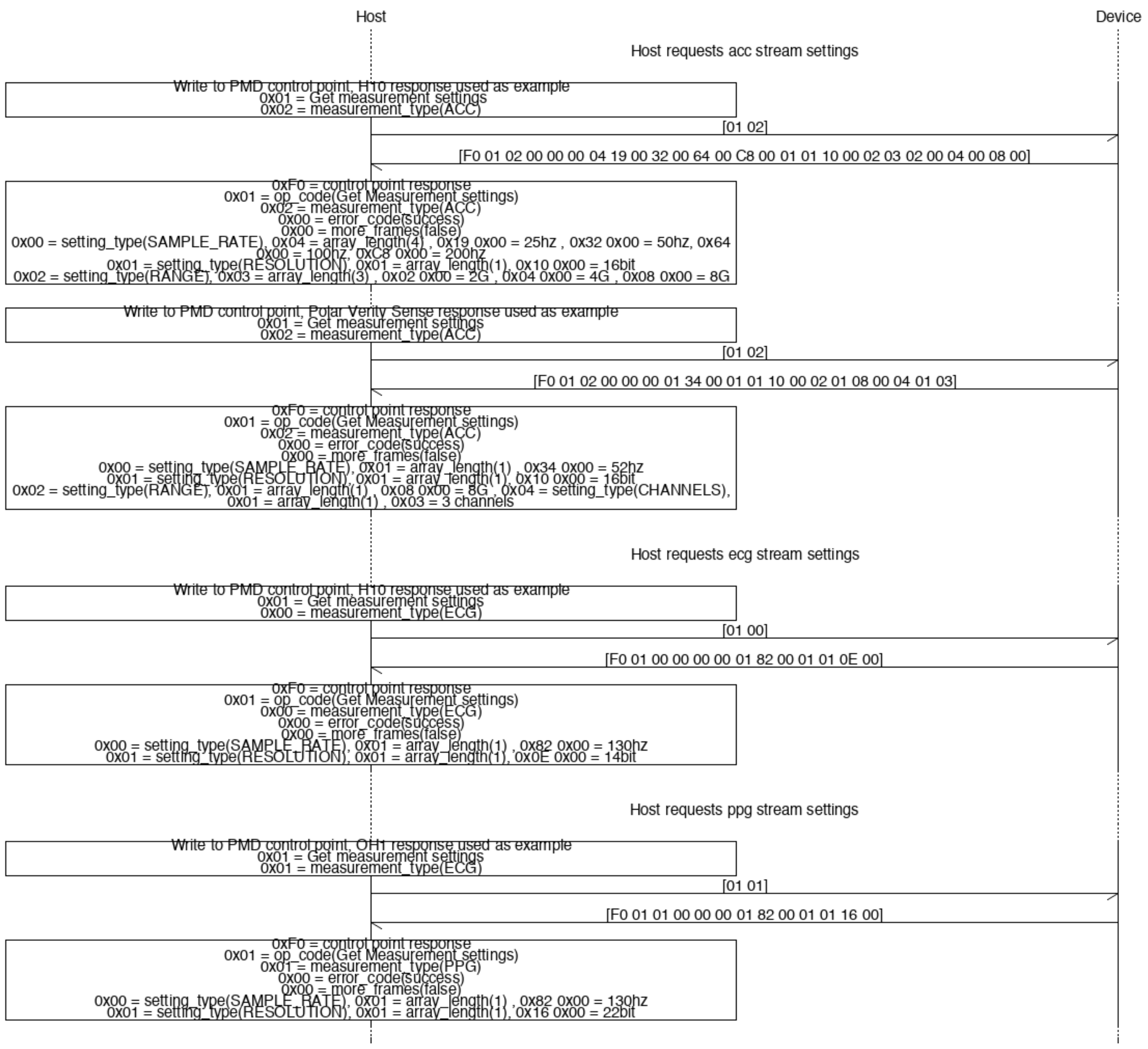
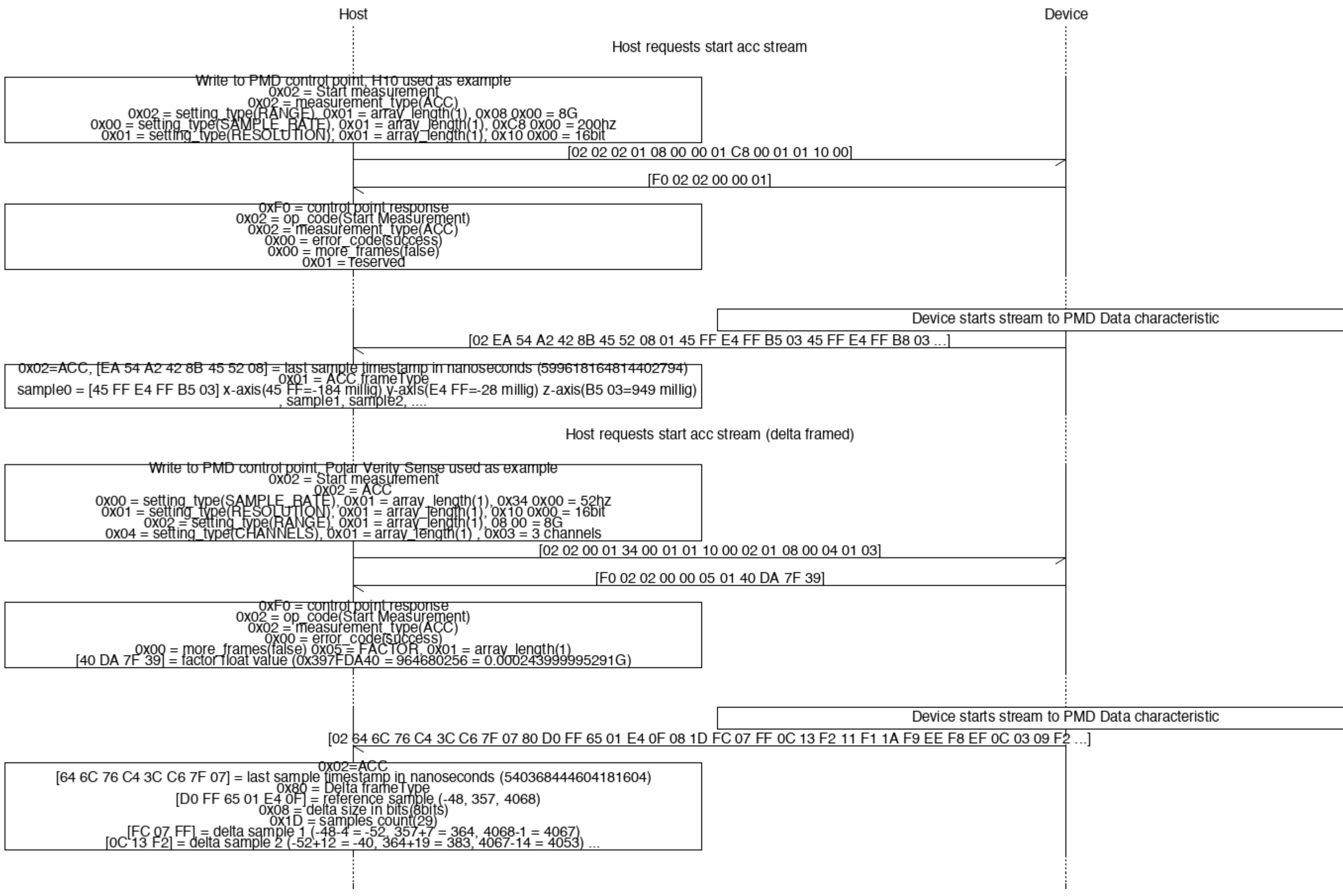

Host
Device
Host requests acc stream settings
Write to PMD control point, H10 response used as example
0x01 = Get measurement settings
0x02 = measurement_type(ACC)
[01 02]
[F0 01 02 00 00 00 04 19 00 32 00 64 00 C8 00 01 01 10 00 02 03 02 00 04 00 08 00]
0xF0 = control point response
0x01 = op_code(Get Measurement settings)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false)
0x00 = setting_type(SAMPLE_RATE), 0x04 = array_length(4) , 0x19 0x00 = 25hz , 0x32 0x00 = 50hz, 0x64
0x00 = 100hz, 0xC8 0x00 = 200hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x10 0x00 = 16bit
0x02 = setting_type(RANGE), 0x03 = array_length(3) , 0x02 0x00 = 2G , 0x04 0x00 = 4G , 0x08 0x00 = 8G
Write to PMD control point, Polar Verity Sense response used as example
0x01 = Get measurement settings
0x02 = measurement_type(ACC)
[01 02]
[F0 01 02 00 00 00 01 34 00 01 01 10 00 02 01 08 00 04 01 03]
0xF0 = control point response
0x01 = op_code(Get Measurement settings)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false)
0x00 = setting_type(SAMPLE_RATE), 0x01 = array_length(1) , 0x34 0x00 = 52hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x10 0x00 = 16bit
0x02 = setting_type(RANGE), 0x01 = array_length(1) , 0x08 0x00 = 8G , 0x04 = setting_type(CHANNELS),
0x01 = array_length(1) , 0x03 = 3 channels
Host requests ecg stream settings
Write to PMD control point, H10 response used as example
0x01 = Get measurement settings
0x00 = measurement_type(ECG)
[01 00]
[F0 01 00 00 00 00 01 82 00 01 01 0E 00]
0xF0 = control point response
0x01 = op_code(Get Measurement settings)
0x00 = measurement_type(ECG)
0x00 = error_code(success)
0x00 = more_frames(false)
0x00 = setting_type(SAMPLE_RATE), 0x01 = array_length(1) , 0x82 0x00 = 130hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x0E 0x00 = 14bit
Host requests ppg stream settings
Write to PMD control point, OH1 response used as example
0x01 = Get measurement settings
0x01 = measurement_type(ECG)
[01 01]
[F0 01 01 00 00 00 01 82 00 01 01 16 00]
0xF0 = control point response
0x01 = op_code(Get Measurement settings)
0x01 = measurement_type(PPG)
0x00 = error_code(success)
0x00 = more_frames(false)
0x00 = setting_type(SAMPLE_RATE), 0x01 = array_length(1) , 0x82 0x00 = 130hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x16 0x00 = 22bit

Figure 2: Example on how to request the stream settings for accelerometer measurement

5.3 Start Stream


Host

Device

Host requests start acc stream

Write to PMD control point, H10 used as example
0x02 = Start measurement
0x02 = measurement_type(ACC)
0x02 = setting_type(RANGE), 0x01 = array_length(1), 0x08 0x00 = 8G
0x00 = setting_type(SAMPLE_RATE), 0x01 = array_length(1), 0xC8 0x00 = 200hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x10 0x00 = 16bit

[02 02 02 01 08 00 00 01 C8 00 01 01 10 00]

[F0 02 02 00 00 01]

0xF0 = control point response
0x02 = op_code(Start Measurement)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false)
0x01 = reserved

Device starts stream to PMD Data characteristic

[02 EA 54 A2 42 8B 45 52 08 01 45 FF E4 FF B5 03 45 FF E4 FF B8 03 ...]

0x02=ACC, [EA 54 A2 42 8B 45 52 08] = last sample timestamp in nanoseconds (599618164814402794)
0x01 = ACC frameType
sample0 = [45 FF E4 FF B5 03] x-axis(45 FF=-184 millig) y-axis(E4 FF=-28 millig) z-axis(B5 03=949 millig)
, sample1, sample2, ...

Host requests start acc stream (delta framed)

Write to PMD control point, Polar Verity Sense used as example
0x02 = Start measurement
0x02 = ACC
0x00 = setting_type(SAMPLE_RATE), 0x01 = array_length(1), 0x34 0x00 = 52hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x10 0x00 = 16bit
0x02 = setting_type(RANGE), 0x01 = array_length(1), 08 00 = 8G
0x04 = setting_type(CHANNELS), 0x01 = array_length(1) , 0x03 = 3 channels

[02 02 00 01 34 00 01 01 10 00 02 01 08 00 04 01 03]

[F0 02 02 00 00 05 01 40 DA 7F 39]

0xF0 = control point response
0x02 = op_code(Start Measurement)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false) 0x05 = FACTOR, 0x01 = array_length(1)
[40 DA 7F 39] = factor float value (0x397FDA40 = 964680256 = 0.000243999995291G)

Device starts stream to PMD Data characteristic

[02 64 6C 76 C4 3C C6 7F 07 80 D0 FF 65 01 E4 0F 08 1D FC 07 FF 0C 13 F2 11 F1 1A F9 EE F8 EF 0C 03 09 F2 ...]

0x02=ACC
[64 6C 76 C4 3C C6 7F 07] = last sample timestamp in nanoseconds (540368444604181604)
0x80 = Delta frameType
[D0 FF 65 01 E4 0F] = reference sample (-48, 357, 4068)
0x08 = delta size in bits(8bits)
0x1D = samples count(29)
[FC 07 FF] = delta sample 1 (-48-4 = -52, 357+7 = 364, 4068-1 = 4067)
[0C 13 F2] = delta sample 2 (-52+12 = -40, 364+19 = 383, 4067-14 = 4053) ...

Figure 3: Example on how to start the accelerometer stream

5.4 Stop Stream

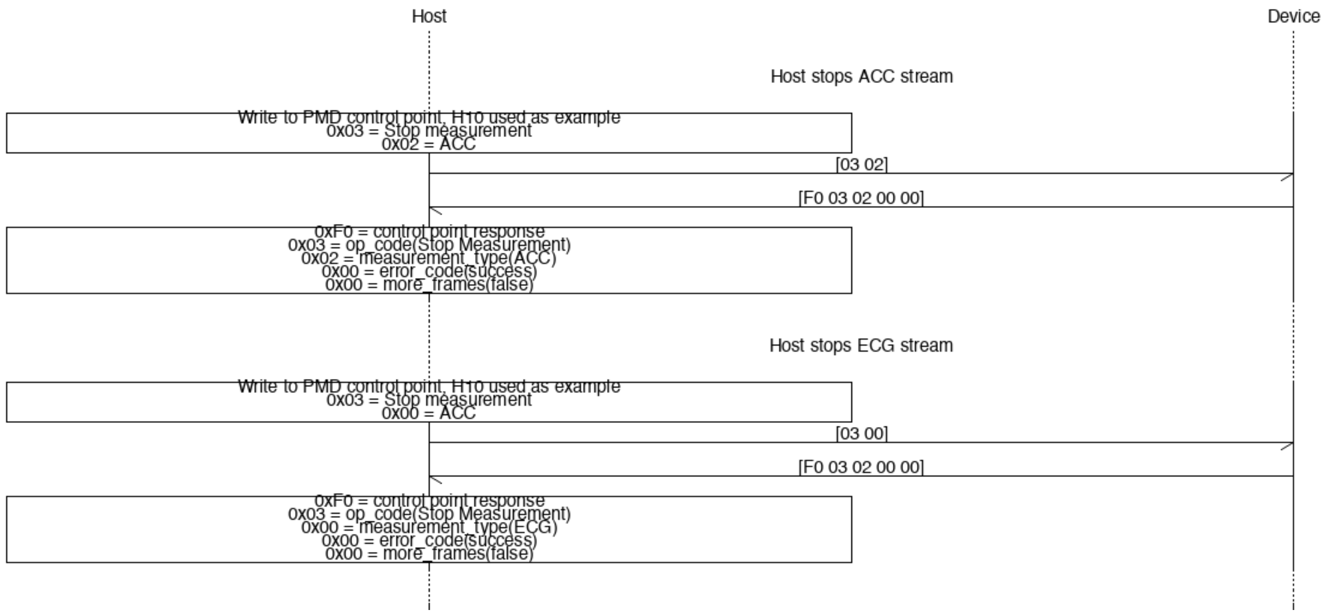

Host
Device
Host stops ACC stream
Write to PMD control point, H10 used as example
0x03 = Stop measurement
0x02 = ACC
[03 02]
[F0 03 02 00 00]
0xF0 = control point response
0x03 = op_code(Stop Measurement)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false)
Host stops ECG stream
Write to PMD control point, H10 used as example
0x03 = Stop measurement
0x00 = ACC
[03 00]
[F0 03 02 00 00]
0xF0 = control point response
0x03 = op_code(Stop Measurement)
0x00 = measurement_type(ECG)
0x00 = error_code(success)
0x00 = more_frames(false)

Figure 4: Example on how to stop the accelerometer stream

5.5 Start SDK mode

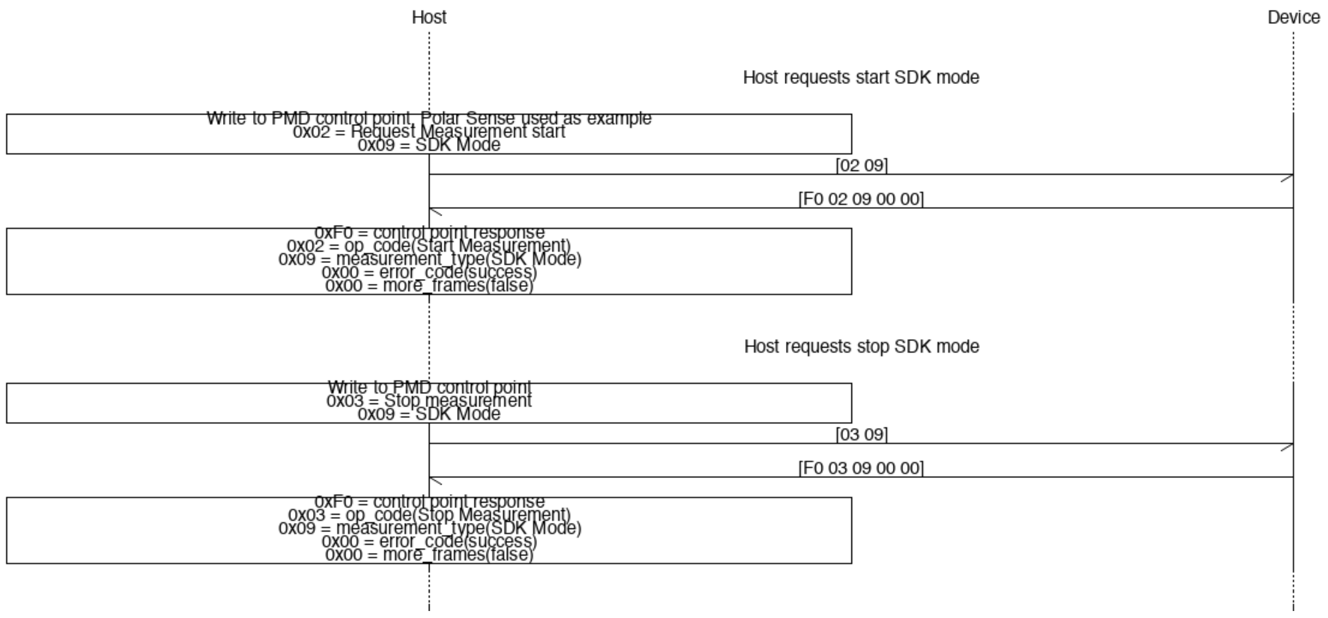

Host
Device
Host requests start SDK mode
Write to PMD control point, Polar Sense used as example
0x02 = Request Measurement start
0x09 = SDK Mode
[02 09]
[F0 02 09 00 00]
0xF0 = control point response
0x02 = op_code(Start Measurement)
0x09 = measurement_type(SDK Mode)
0x00 = error_code(success)
0x00 = more_frames(false)
Host requests stop SDK mode
Write to PMD control point
0x03 = Stop measurement
0x09 = SDK Mode
[03 09]
[F0 03 09 00 00]
0xF0 = control point response
0x03 = op_code(Stop Measurement)
0x09 = measurement_type(SDK Mode)
0x00 = error_code(success)
0x00 = more_frames(false)

Figure 5: Example on how to start SDK mode with the device

5.6 Get SDK mode measurement settings

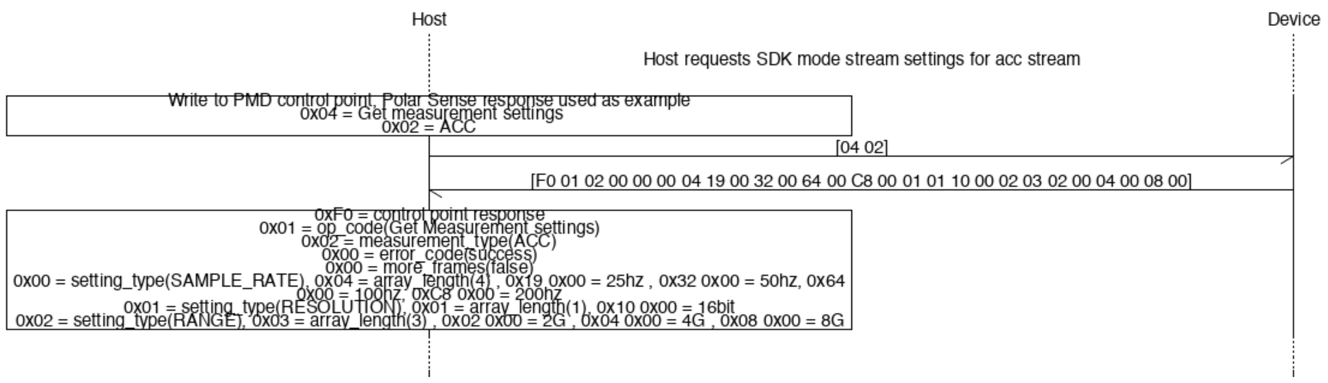

Host
Device
Host requests SDK mode stream settings for acc stream
Write to PMD control point, Polar Sense response used as example
0x04 = Get measurement settings
0x02 = ACC
[04 02]
[F0 01 02 00 00 00 04 19 00 32 00 64 00 C8 00 01 01 10 00 02 03 02 00 04 00 08 00]
0xF0 = control point response
0x01 = op_code(Get Measurement settings)
0x02 = measurement_type(ACC)
0x00 = error_code(success)
0x00 = more_frames(false)
0x00 = setting_type(SAMPLE_RATE), 0x04 = array_length(4) , 0x19 0x00 = 25hz , 0x32 0x00 = 50hz, 0x64
0x00 = 100hz, 0xC8 0x00 = 200hz
0x01 = setting_type(RESOLUTION), 0x01 = array_length(1), 0x10 0x00 = 16bit
0x02 = setting_type(RANGE), 0x03 = array_length(3) , 0x02 0x00 = 2G , 0x04 0x00 = 4G , 0x08 0x00 = 8G

Figure 6: Example on how to request the stream settings for accelerometer measurement in SDK mode