Verity Sense Offline Measurement

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 offline recording, retrieval of offline recording files from device and parsing the offline recording files 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.

Verity Sense Offline Measurement

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 offline recording, retrieval of offline recording files from device and parsing the offline recording files 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 offline recording

Offline and online measurements 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 1 for offline 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 Accelometer measurement requestByte is in offline measurement,

```
recordingType = 1,
measurementType = 2,
requestByte = (1 shl 7) | 2 = -123
```

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 offline recording

To stop the offline recording send PMD Control Point command ”Stop measurement” to the device.

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

where measurementType is the Value if Measurement type from Table 1

Measurement type	Value
ECG	0u
PPG	1u
ACC	2u
PPI	3u
GYRO	5u
MAGNETOMETER	6u

Table 2: Measurement types

Description	UUID
PMD Service	FB005C80-02E7-F387-1CAD-8ACD2D8DF0C8
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 Retrieving offline recording files with PFTP service

3.1 List files in device

PFTP service is Polar specific defined service that can be used in reading PMD files from a Polar Bluetooth device. To read a file from a device one must first establish a Bluetooth connection to the device a.k.a session. For file transfer purposes one has to fetch a PTFP client that handles the file transfer. One can do it by fetching the client (BlePsFtpClient) with service UUID for PFTP (”0000FEEE-0000-1000-8000-00805f9b34fb”). Verify that client has the service available (should be OK for Verity Sense).

In order to find the files from the device with the PFTP service client one shall create a request to the device by using compiled (Java, Swift) PftpRequest protobuf. Use PbPFtpOperation command ”GET” and set path to ”/U/0/” of the PtfpRequest class. Build the request class into PftpRequest protobuf file and then serialize the file into a byte array type. Create a request to the BlePsFtpClient client and

give the serialized PftpRequest protobuf file as a parameter to the request. PTFP service will return a ByteBuffer response. Parse that response with PftpPDirectory class instance and you will get a list of directory entry. Find the desired files from the directory and retrieve the files on-by-one as described in the next chapter.

```
client.sendNotification(8, null)
```

3.2 Directory structure

Polar devices follow a predefined directory structure for measurement data. The path parts are as follows "/U/X/YYYYMMDD/R/HHMMSS/XXX.REC", where

- U, The root folder for measurement data
- YYYYMMDD, Date of the measurement start date
- HHMMSS, local time at which the recording was started
- .REC, extension indicating that this file is Polar sensor device compatible offline recording file
- X, User number
- R, date based folder containing recording data for this day
- XXX, name of the file, depending on which type is recorded

4 Retrieve a file from a device

Initially get an instance of PSFTP (Polar Simple File Transfer Protocol) client (BlePsFtpClient) from Bluetooth session (BleGattBase) with UUID (”FB005C51-02E7-F387-1CAD-8ACD2D8DF0C8”). Before trying to retrieve a file a notifation to the sensor device about the incoming operation must be sent. Send notification to the device with compiled PftpNotification procobuf,

```
client.sendNotification(PftpNotification.PbPFtpHostToDevNotification
  .INITIALIZE_SESSION_VALUE)
```

To retrieve the file send a PftpRequest to the device with the desired path to retrieve the file using that client. Use PbPFtpRequest protobuf with PbPFtpOperation command ”GET” and set path to the desired.

```
val request = PftpRequest.PbPFtpOperation.newBuilder()
request.command = PftpRequest.PbPFtpOperation.Command.GET
request.path = "/U/0/20240411/R/202543/HR.REC" // Example path
request.build
client.request(request.toByteArray())
```

5 Parsing the file data

Once the recording has been retrieved from the device the recording file data has to be parsed to a human readable format. This document assumes that the reader already can parse and read uncompressed recording file data. However, some measurements have been compressed. This chapter describes how to decompress that recording file data.

Measurement type	Compressed
PPG	yes
PPI	no
Acceleration	yes
Heart rate	no
Magnetometer	yes
Gyroscope	no
Temperature	yes

Table 5: Measurement types

5.1 Parsing compressed data

Recording file data comes in as PmdDataFrame payload. Payload is of type ByteArray.

5.2 Parsing metadata from PmdDataFrame from the Payload

Meta data consists of: Offline recording header, measurement start time, recording settings and data payload size. Parse each item as a new object and construct a new ”MetaData” object. Metadata is in the beginning of the ByteArray of PmdDataFrame. Please, ignore first byte as it is not relevant here.

5.3 Parse offline recording header

Offline recording header is 16 Bytes long and consists of fields:

- magic
- version
- free
- esw hash

and must be parsed from the incoming ByteArray in this order. Size of each value is 4 Bytes. Remember to start reading from Byte 1, ignore first Byte at index 0.

5.4 Parse start time

Start time value is 20 Bytes long and it starts from the last index of the offline recording header. (start index = 1 + offline recording header length) You can ignore the last Byte, it is not relevant. Start time follows ISO 8601 standard.

5.5 Parse recording settings

Recording settings starts from index 36 of meta data byte array, ends at the end of the byte array. First, discover the first setting setting type in the index 0 of the byte array. Secondly, discover the byte count of the setting in the index 1 of the byte array. Iterate the same byte array by the setting count. Determine the settings field size by mapping the integer value from Table 5. for the current setting type. Determine the setting item by converting a byte array part to unsigned integer value where offset (initially 2) plus field size and field size is the field size from Table 5. Add the setting item to a list, where setting item is a pair of { Setting name, Setting value }. Repeat until the end of the byte array. Start next iteration at index iteration count multiplied by two (0,1,2,...) plus the ”Field size” in Table 5 of the byte array.

Example:

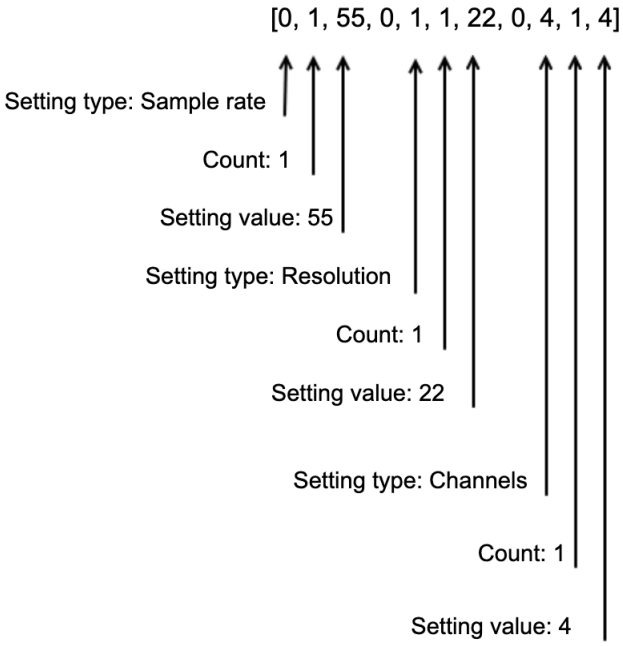

Figure 1: Example on how recording settings can be found from byte array

Setting type Field size (Bytes)	Enumeration
Sample rate 2	0
Resolution 2	1
Range 2	2
Range milli unit 4	3
Channels 1	4
Factor 4	5

Table 6: Measurement types

5.6 Parse packet size

Data packet size is calculated from the metadata. The start index for the data packet size is 36 + settings size + 1. In the above example the start index is thus 36 + 12 +1 = 49. The end index for data packet size is start index plus 1 resulting

end index of value 50 in the example. Calculate the data packet size by converting the resulted byte array to unsigned integer with offset value of 0 and length is 2.

5.7 Constitute the meta data object

Create an object with information of,

```
Object MetaDataPair {
Metadata {
 Offline recording header
 Recording settings
 Start time
 Size of data packet
}
 Size of the meta data byte array
}
```

6 Parse data

Get a the recorded data without the meta data by dropping the meta data bytes in the beginning of the recorded measurement data byte array.

Find the packet size and sample rate from the previously parsed meta data. Iterate through the recorded measurement data byte array (w/o metadata) and process a chunk of data size of previously calculated packet size in each iteration. Construct a new PmdDataFrame from each chunk, where PmdDataFrame is;

```
Object PmdDataFrame {
data = chunk /* Chunk as ByteArray */
timestamp = timestamp from previous frame /*Can be initially zero */
sampleRate = metaData.sampleRatefactor = 1
}
```

Then, parse the data from the PMD data frame similarly as when parsing online recording measurement data.