HxM2 Application Developer Guide

# 

# Document Version Control

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 1.0 | 13th Aug 2012 | Initial Version |
| 1.1 | 27th Aug 2012 | Added details of Test Mode data to HxM2 custom service |

# References

|  |  |  |
| --- | --- | --- |
| **Ref #** | **ID** | **Description** |
|  | HRS\_SPEC V10r00 | Heart Rate Service Specification |
|  | BAS\_SPEC V10r00 | Battery Service Specification |
|  | DIS\_SPEC V11r00 | Device Information Service Specification |

# Definitions, Abbreviations and Glossary

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| BLE | Bluetooth Low Energy |
|  |  |

# Overview

This guide is aimed at developers of applications which interact with the HxM2 via Bluetooth Low Energy (BLE).

# BLE Services

The HxM2 supports a number of different services of which some are defined by the Bluetooth SIG and others that are specific to the HxM2 as listed in Table 4‑1.

|  |  |
| --- | --- |
| **Service Name** | **Adopted Ver** |
| Heart Rate Service | 1.0 |
| Battery Service | 1.0 |
| Device Information Service | 1.1 |
| HxM2 Custom Data Service | Custom |
| Firmware Update Service | Custom |

Table 4‑1 Available services in HxM2

## UUID’s

Each service is identified by a 128-bit UUID and each service is a collection of characteristics which are also identified by UUID’s. However services and characteristics that have been adopted by the Bluetooth SIG use 16-bit UUIDs which are aliases that actually represent 128-bit values derived from the Bluetooth Base UUID:

128\_bit\_value = 16\_bit\_value \* 296 + Bluetooth\_Base\_UUID

Bluetooth Base UUID = 0000**0000**-0000-1000-8000-00805F9B34FB

Therefore the 2 bytes underlined above are replaced by the 16-bit UUID.

For any custom services, a full 128-bit UUID must be used and because of this, an HxM2 128-bit UUID has been declared:

HxM2 Base UUID = BEFD**0000**-C979-11E1-9B21-0800200C9A66

All HxM2 custom services use UUID’s based on this UUID where the 2 bytes underlined above are changed to create a unique UUID for each custom service & characteristic.

Therefore whenever a 16-bit UUID is used it is assumed to be based upon the 128-bit Bluetooth Base UUID so anytime a custom service or characteristic is used, the full 128-bit UUID must be used.

## Standard Services

For details of all standard services refer to the specific documents from Bluetooth.org.

### Heart Rate Service

The Heart Rate Service is implemented within the HxM2 as documented in [1]. Energy Expended is not calculated by the HxM2 and therefore the Energy Expended field is not included. When the Client Characteristic Configuration descriptor is configured for notification, the Heart Rate Measurement characteristic is notified on average 1 time per second. The Heart Rate Measurement characteristic is notified on average 1 time per second.

### Battery Service

The battery service as specified in [2] is implemented within the HxM2. As well as supporting the Mandatory Read property, the HxM2 also supports the optional Notify property.

### Device Information Service

All characteristics of the Device Information service are supported as specified in [3]. Because the device contains 4 separate pieces of firmware (BLE Microcontroller Bootloader & Application Firmware + Host Microcontroller Bootloader & Application Firmware), the Firmware Revision & Software Revision strings are written as follows so that all version details can be communicated:

Firmware Revision String: BLE Firmware / BLE Bootloader (E.g. “v1.0.2.0/1.0.0.0”)

Software Revision String: Host Firmware / Host Bootloader (E.g. “v1.2.0.0/1.3.0.0”)

## Custom Services

There are some requirements that cannot be met by using standard services alone. Therefore it was necessary to create HxM2 specific services to satisfy these requirements.

### HxM2 Custom Data Service

This service is used by the client to receive activity and peak acceleration data from the HxM2.

The following characteristics are exposed in the HxM2 Custom Data Service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristic Name** | **Requirement** | **Mandatory Properties** | **Optional Properties** | **Security Permissions** |
| Custom Measurement Data | Mandatory | Notify |  | None. |
| Custom Measurement Data Client Characteristic Configuration descriptor | Mandatory | Read, Write |  | None. |
| Test Mode Data | Mandatory | Notify |  | None. |
| Test Mode Data Client Characteristic Configuration descriptor | Mandatory | Read, Write |  | None. |

Table 4‑2 HxM2 Custom Data Service characteristics

shows the complete attribute table for the activity service. Services are shown in yellow, characteristics are shown in blue and characteristic values / descriptors are shown in grey. All UUID’s are shown as 16-bit but where they are described as 128-bit, the value shown must be used with the HxM2 Base UUID to create a 128-bit UUID.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type (hex)** | **Type (Define)** | **Value (default)** | **GATT Server Permissions** | **Notes** |
| 0x2800 | GATT\_PRIMARY\_SERVICE\_UUID | 0xFF10 (HXM2\_CUSTOM\_SERV\_UUID)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Start of Custom Service |
| 0x2803 | GATT\_CHARACTER\_UUID | 10(properties: notify only)  xx xx(handle)  11 FF (UUID: 0xFF11)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Custom Measurement Characteristic Declaration |
| 0xFF11  (128-Bit) | ACTIVITY\_MEAS\_UUID | 0 (1 Byte) | (none) | Custom Measurement Characteristic value |
| 0x2902 | GATT\_CLIENT\_CHAR\_CFG\_UUID | 00 00(2 Bytes) | GATT\_PERMIT\_READ| GATT\_PERMIT\_WRITE | Custom Measurement Characteristic configuration |
| 0x2803 | GATT\_CHARACTER\_UUID | 10(properties: notify only)  xx xx(handle)  12 FF (UUID: 0xFF12)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Test Mode Data Characteristic Decleration |
| 0xFF12  (128-Bit) | TESTMODE\_DATA\_UUID | 0 (1 Byte) | (none) | Test Mode Data Characteristic value |
| 0x2902 | GATT\_CLIENT\_CHAR\_CFG\_UUID | 00 00(2 Bytes) | GATT\_PERMIT\_READ| GATT\_PERMIT\_WRITE | Test Mode Data Characteristic configuration |

Table ‑ HxM2 Custom HxM2 Service Attribute List

#### HxM2 Custom Measurement Notification Data

Once notification of HxM2 custom measurement is enabled, the Custom Measurement characteristic is used by the HxM2 to send custom measurement data to the client device. Included in the characteristic are a Flags field (for showing the presence of optional fields) and depending upon the contents of the Flags field, an Activity field and a Peak Acceleration field.

A notification is sent by the HxM2 at a rate of once per second.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **RFU** | | | | | | **PA** | **ACT** |

*Table 4‑4 Activity Measurement Characteristic Flags Field*

**ACT** Bit 0 Activity Flag

0 Activity Field not included

1 Activity Field included

**PA** Bit 1 Peak Acceleration

1. Peak Acceleration Field not included
2. Peak Acceleration Field included

**RFU** Bits 7-2 Reserved for Future Use

Following the flags field will be any included fields as described by the flags as shown in Table 4‑5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flags Field | Activity Field | | Peak Accel Field | |
| LSB | MSB | LSB | MSB |

Table ‑ *Custom Measurement Characteristic Data*

##### Activity Field

The current activity level. Valid range is 0…16 with 0.01 unit resolution, e.g. a value of 100 represents activity level = 1.00.

##### Peak Acceleration

The highest g-force measured in the last measurement period of 1 second. Valid range is 0…16 with 0.01 unit resolution e.g. a value of 250 represents 2.50g.

#### HxM2 Custom Test Mode Notification Data

Once notification of HxM2 test mode data is enabled, the Test Mode Data characteristic is used by the HxM2 to send test mode data to the client device. Included in the characteristic are the fields described in Table 4‑6.

A test mode data notification is sent by the HxM2 at a rate of once per second but only when the HxM2 is in factory test mode.

Table ‑ Test Mode Characteristic Data

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Accel X | | Accel Y | | Accel Z | | ECG Min | | ECG Max | | Flash ID | | |
| LSB | MSB | LSB | MSB | LSB | MSB | LSB | MSB | LSB | MSB | Man ID | Mem Type | Mem Cap |

Each axis in accelerometer data is a signed value between -2048 and +2047 with 1G equal to 83. Data is averaged over the last 1 second period to remove noise/vibrations.

ECG minimum and maximum are 10-bit unsigned values between 0 and 1023 and indicate the lowest and highest sample over the last 1 second period.

The Flash ID is split up into 3 bytes which are Manufacturer ID, Memory Type and Memory Capacity. For the M25PX80 the correct ID is

* Manufacturer ID = 0x20
* Memory Type = 0x71
* Memory Capacity = 0x14

### Firmware Update Service

This service is used by the client to send a new firmware image to the server (HxM2).

The following characteristics are exposed in the Firmware Update Service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristic Name** | **Requirement** | **Mandatory Properties** | **Optional Properties** | **Security Permissions** |
| Firmware Update Status | Mandatory | Notify |  | None. |
| Firmware Update Status Client Characteristic Configuration descriptor | Mandatory | Read, Write |  | None. |
| Firmware Image Data | Mandatory | Write |  | None. |

Table 4‑7 Firmware Update Service characteristics

Table 4‑8 shows the complete attribute table for the firmware update service. Services are shown in yellow, characteristics are shown in blue and characteristic values / descriptors are shown in grey. All UUID’s are shown as 16-bit but where they are described as 128-bit, the value shown must be used with the HxM2 Base UUID to create a 128-bit UUID.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type (hex)** | **Type (Define)** | **Value (default)** | **GATT Server Permissions** | **Notes** |
| 0x2800 | GATT\_PRIMARY\_SERVICE\_UUID | 0xFFB0 (FWUPDATE\_SERV\_UUID)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Start of Firmware Update Service |
| 0x2803 | GATT\_CHARACTER\_UUID | 10(properties: notify only)  xx xx(handle)  B1 FF (UUID: 0xFFB1)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Firmware Update Status Characteristic Declaration |
| 0xFFB1  (128-Bit) | FWUPDATESTATUS\_UUID | 0 (1 Byte) | (none) | Firmware Update Status Characteristic value |
| 0x2902 | GATT\_CLIENT\_CHAR\_CFG\_UUID | 00 00(2 Bytes) | GATT\_PERMIT\_READ| GATT\_PERMIT\_WRITE | Firmware Update Status Characteristic configuration |
| 0x2803 | GATT\_CHARACTER\_UUID | 08(properties: write only)  xx xx(handle)  B2 FF (UUID: 0xFFB2)  (HxM2 128-bit) | GATT\_PERMIT\_READ | Firmware Image Data Characteristic Declaration |
| 0xFFB2  (128-Bit) | FWIMAGEDATA\_UUID | xx xx….xx(70 Bytes) | GATT\_PERMIT\_WRITE | Firmware Image Data Characteristic value |
| 0x2901 | GATT\_CHAR\_USER\_DESC\_UUID | "Firmware Image Data" | GATT\_PERMIT\_READ | Firmware Image Data Characteristic description |

Table 4‑8 Firmware Update Service Attribute List

#### Firmware Update Status

The Firmware Update Status is used by the HxM2 to indicate a change of status during a firmware update. Therefore, the Client Characteristic Configuration descriptor must be configured for notification of the Firmware Update Status at the beginning of the firmware update process.

The Notification is always 1 byte which will be one of the following values.

|  |  |
| --- | --- |
| **Status** | **Value** |
| STOP | 0x00 |
| START | 0x01 |
| WAIT | 0x02 |
| CONTINUE | 0x03 |
| COMPLETE | 0x04 |

Table 4‑9 Firmware Update Status Notification Values

#### Firmware Image Data

As shown in Table 4‑8 Firmware Update Service Attribute List, the firmware image data characteristic value has a length of 70 bytes and has write only properties. Therefore, when the client device has firmware image data to send, it must write 70 bytes to this characteristic.

This message requests that the firmware image data within the message be stored in the HxM2 for programming when complete.

The data written to this characteristic must be in the format given in Table 4‑10. As the DLC takes up 2 of the 70 bytes, this gives a maximum DLC value of 68 but sometimes the DLC will be less than this and in these cases 70 bytes should always be written but the remainder of the 70 bytes that are unused should all be set to 00.

|  |  |
| --- | --- |
| **Byte** |  |
| **0** | DLC (LS Byte) |
| **1** | DLC (MS Byte) |
| **3** | Image Type |
| **4** | Image Sequence Number (LS Byte) |
| **5** | Image Sequence Number (MS Byte) |
| **6** | Data Type |
| **7** | Firmware Image Data First Byte |
| **:** | : |
| **:** | : |
| **69** | Firmware Image Data Last Byte |

Table 4‑10 *Firmware Image Data Characteristic*

* The Image Type describes the type of firmware Image included within the message

|  |  |
| --- | --- |
| Code | Description |
| 0 | - |
| 1 | MSP430 HxM2 Application |
| 2 | CC2541 HxM2 BLE Stack |

Table 4‑11 *Image Types*

* The Image Sequence Number is incremented for every message containing data relating to a particular Firmware image and ranges from 0 to 65535.
* The Data Type describes the type of data following this byte.

|  |  |  |
| --- | --- | --- |
| Code | Description | Data Length |
| 0 | - |  |
| 1 | Prepare for new Firmware Image | 0 |
| 2 | Firmware Image Address | 4 (LS Byte First) |
| 3 | Firmware Image Data | Variable Length |
| 4 | Firmware Image Complete | 0 |

Table 4‑12 *Data Types*

* Firmware image data is the only parameter which does not have a fixed length, therefore the length of this can be determined from the DLC and the fixed length of all other parameters.

# Firmware Image Transfer Client Procedure

The following procedure explains how a client device (e.g. PC or Phone) can use the Firmware Update and Device Information Services as described in this document to perform a Firmware Update. application that initiates the Firmware Update is The procedure below is followed by the client to perform the update:

1. Connect to HxM2.
2. Query version number of both MSP430 (Host) & CC2541(BLE) application firmware by using the Device Information Service as described in 4.2.3.
3. Compare the version details of the device with the latest firmware versions available. If one or both pieces of firmware have a newer version available, then they should be updated one at a time.
4. Open binary image file and confirm that image is valid by checking the App Identifier in the Application Info Header which is located at the very start of a valid HxM2 image file. This App Identifier will read either “APP0” for an MSP430 image or “APP1” for a CC2541 image as shown in Table 5‑1.
5. Enable Firmware Update Notifications by writing 01:00 to the Firmware Update Client Characteristic Configuration (see Table 4‑8).
6. Send device a message to prepare for a firmware update (see Table 4‑12) by writing to the Firmware Image Data Characteristic value (see Table 4‑8).
7. Wait for notification of change in firmware update status to START (see Table 4‑9).
8. Start sending the firmware image from the binary file by first sending an address message which is the offset into the binary file at which the next data to be sent starts.
9. Continue by sending the firmware image data in blocks of up to 64 bytes at a time until the entire contents of the binary file have been sent, then send a Firmware Image Complete message (see Table 4‑12).
10. If any part of the binary file contains a series of 0xFF’s, this is code space that is currently unused by the device. If there is a sequence of greater than 64 successive 0xFF’s, it is more efficient not to send this data but instead, either send an address message containing the offset into the file where the next block of non 0xFF data exists then continue sending data from here or if the 0xFF’s continue to the end of the file, just send a Firmware Image Complete message.
11. While sending the blocks of firmware image data, keep checking for notification of a change in Firmware Update Status. If at any time, this status changes to STOP (see Table 4‑9), the image transfer should be aborted and re-started.
12. When the Firmware Image Complete message has been sent, wait for notification that the Firmware Update Status has changed to COMPLETE (see Table 4‑9).
13. Terminate BLE connection then wait while the device actually performs the firmware update before attempting to re-connect.
14. When connection is re-established, check Firmware version numbers again by using the Device Information service.
15. Confirm that the firmware image sent has been successfully updated.

|  |  |  |
| --- | --- | --- |
| **Byte Offset** | **Size (Bytes)** | **Description** |
| 0 | 2 | CRC-16 Value (calculated starting at offset 8) |
| 2 | 2 | Unused (reserved for future use) |
| 2 | 4 | CRC-16 Length (num. Bytes to calculate CRC16 for) |
| 8 | 4 | App Identifier = 0x41505030 (“APP0”) or 0x41505031(“APP1”) |
| 12 | 2 | Application Start Function Offset (relative to AppInfo address) |
| 14 | 2 | Application Version – Major (0-65535, MSB first) |
| 16 | 2 | Application Version – Minor (0-65535, MSB first) |
| 18 | 2 | Application Version – Build (0-65535, MSB first) |
| 20 | 2 | Application Version – Branch (0-65535, MSB first) |

Table 5‑1: HxM2 Firmware Image Application Info Header

## Example Firmware Image File

Figure 5‑1 gives details of an actual Binary MSP430 Firmware Image File. The total file length is 31744 Bytes.

The App Identifier at offset 0x0008 into the file is “APP0” which confirms the file is an MSP430

Firmware Image.

It can be seen that the file contains some unused code space (all 0xFF’s).

0x7C00

0x7BDE

0x3021

0x0000

Used Code Space

Used Code Space

Unused Code Space

(All 0xFF’s)

0x0000 – 0x3020 (12321 Bytes)

0x3021 – 0x7BDD (19389 Bytes)

0x7BDE – 0x7BFF (34 Bytes)

**Binary Image File**

Figure 5‑1 Binary Image File

Table 5‑2 shows how the image file in Figure 5‑1 is transferred over BLE by writing to the Firmware Image Data Characteristic of the Firmware Update Service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DLC** | **Image Type** | **Seq No.** | **Data Type** | **Data** |
| 04 00 (4) | 01 (MSP430) | 00 00 (0) | 01 (Prepare for new Firmware Image) | No Data |
| 08 00 (8) | 01 (MSP430) | 01 00 (1) | 02 (Firmware Image Address) | 00 00 00 00 (0x0000) |
| 44 00 (68) | 01 (MSP430) | 02 00 (2) | 03 (Firmware Image Data) | Bytes 0-63 |
| 44 00 (68) | 01 (MSP430) | 03 00 (3) | 03 (Firmware Image Data) | Bytes 64-127 |
| : | : | : | : | : |
| 44 00 (68) | 01 (MSP430) | C1 00 (193) | 03 (Firmware Image Data) | Bytes 12224-12287 |
| 25 00 (37) | 01 (MSP430) | C2 00 (194) | 03 (Firmware Image Data) | Bytes 12288-12320 |
| 08 00 (8) | 01 (MSP430) | C3 00 (195) | 02 (Firmware Image Address) | DE 7B 00 00 (0x7BDE) |
| 26 00 (38) | 01 (MSP430) | C4 00 (196) | 03 (Firmware Image Data) | Bytes 31710-31743 |
| 04 00 (4) | 01 (MSP430) | C5 00 (197) | 04 (Firmware Image Complete) | No Data |

Table 5‑2 Firmware Image Data Transfer