*n*Blue™ *Bluetooth*® 4.0
SensorBug Interface Specification
v1.3a

AT HOME. AT WORK. ON THE ROAD. USING BLUETOOTH WIRELESS TECHNOLOGY MEANS TOTAL FREEDOM FROM THE CONSTRAINTS AND CLUTTER OF WIRES IN YOUR LIFE.

Table of Contents

TABLE OF CONTENTS 2
REVISION HISTORY 4
1 IMPORTANT NOTES – PLEASE READ PRIOR TO CONTINUING 5
1.1 IMPORTANT NOTES 5
1.2 KNOWN ISSUES IN THIS VERSION 5
2 HARDWARE 6
2.1 HARDWARE SUMMARY 6
2.2 BATTERY 6
2.2.1 Changing the Battery 6
2.2.2 Maximizing Battery Life 6
3 BASIC OPERATION 7
3.1 POWERING ON 7
3.2 POWERING OFF 7
3.3 PAIRING 7
3.4 CLEARING PAIRED DEVICES 7
3.5 FACTORY RESET 7
3.6 USING THE DEVICE 7
4 ADVERTISING (UNCONNECTED LE INTERFACE) 8
4.1 ADVERTISING DATA STRUCTURES 8
4.1.1 Flags 8
4.1.2 16-Bit Incomplete Service List 8
4.1.3 Manufacturer Specific Data 8
4.1.4 Data 11
4.2 SCAN RESPONSE DATA STRUCTURES 12
4.2.1 TX Power Level 12
4.2.2 Complete Name 12
5 GATT SERVICES (CONNECTED LE INTERFACE) 13
5.1 DEVICE INFORMATION SERVICE 14
5.2 BATTERY SERVICE 14
5.3 LINK LOSS SERVICE 14
5.4 IMMEDIATE ALERT SERVICE 14
5.5 TX POWER SERVICE 14
5.6 BR ACCELEROMETER SERVICE 15
5.6.1 Config Characteristic 15
5.6.2 Data Characteristic 20
5.6.3 Alert Characteristic 21

5.6.4 Service Specific Errors 22
5.7 BR LIGHT SERVICE 23
5.7.1 Config Characteristic 23
5.7.2 Data Characteristic 25
5.7.3 Alert Characteristic 26
5.7.4 Stats Characteristic 27
5.7.5 Service Specific Errors 27
5.8 BR TEMPERATURE SERVICE 27
5.8.1 Config Characteristic 27
5.8.2 Data Characteristic 29
5.8.3 Alert Characteristic 29
5.8.4 Stats Characteristic 30
5.8.5 Service Specific Errors 30
5.9 BR OTA FIRMWARE UPDATE SERVICE 31
5.9.1 Control / Status Characteristic 31
5.9.2 Data Characteristic 32
5.9.3 Client Firmware Update Procedure 33
5.10 BR PAIRING MANAGEMENT SERVICE 39
5.10.1 Control / Status Characteristic 39
5.10.2 Data Characteristic 41
5.10.3 Config Characteristic 43
5.10.4 Ad Encryption Key Characteristic 44
5.10.5 Service Specific Errors 44
5.11 BR UTILITIES SERVICE 45
5.11.1 Device Name Characteristic 45
5.11.2 Default Connection Parameters Characteristic 45
5.11.3 Current Connection Parameters Characteristic 47
5.11.4 Advertising Parameters Characteristic 48
5.11.5 RF Power Levels Characteristic 53
5.11.6 Disconnect Characteristic 54
5.11.7 Public Device Address Characteristic 54
5.11.8 Config Counter Characteristic 54
5.11.9 Blink LED Characteristic 55
5.11.10 Factory Value Characteristic 55

Revision History

Rev #	Date	Description
1.0.0.0	12/8/2014	**Initial Document**
1.1.0.0	5/14/2015	**New Features** ▪ Added ledOnConnect flag to advertising parameters characteristic for testing
1.2.0.0	1/11/2016	**New Features** ▪ Added Blink LED characteristic in BRU Utilities Service
1.3.0.0	5/15/2017	**New Features** ▪ Added Factory Value characteristic ▪ Added enableAlertsIndividually, enableAlertLow and enableAlertHigh flags to Temperature and Light Config flags allowing low and high alerts to be enabled/disabled individually ▪ Added alertResetCount and alertResetDiff to Temperature and Light Config to allow better control over how consecutive alerts will be triggered. **Changes** ▪ Temperature and Light Config flags.enableAlertClear has been deprecated. A new alert will now only be generated if the data value goes back past the threshold by alertResetDiff for alertResetCount readings. ▪ To lower power consumption during advertising, the scan response data (Name and TX Power) will only be populated when the device is pairable or the button has been pressed to force the device to advertise. ▪ Improved BROTA reliability. **Bug Fixes** ▪ Fixed an issue causing light and temp ad data not to be populated when an accel alert occurred. ▪ Fixed an issue causing BRU Service Advertising Parameters and Device Name (if 20 bytes long) reads to fail if read by UUID. Now just 19 bytes will be returned and a GATT Read Long will be needed to read the full value. ▪ Fixed an issue causing the device to advertise as connected if the connection was dropped during an OTA update ▪ Fixed an issue causing a paired devices name in the Pairing Management Service to sometimes get corrupted if the device was re-paired. ▪ Fixed a bug causing config counter to not increment on a name change.

1 Important Notes – Please Read Prior To Continuing

1.1 Important Notes

- To provide the best firmware architecture, design, and future profile support there will not be 100% backwards compatibility between releases.
- Before proceeding, make sure the first two digits of the device's firmware version matches the first two digits of the version of this document.
- All structures are packed.
- :1 indicates a bit field of 1, a single bit.
- uint8 = unsigned 8-bit integer, int8 = signed 8-bit integer
- uint16 = unsigned 16-bit integer, int16 = signed 16-bit integer
- uint32 = unsigned 32-bit integer, int32 = signed 32-bit integer
- Multibyte values are in **little endian** order – **the first byte will be the LSB**.

1.2 Known Issues in This Version

- None

2 Hardware

This section is meant to provide a summary of the hardware specifications.

2.1 Hardware Summary

- BR-LE4.0-S3A Bluetooth LE Single Mode Module (Max TX Power = 0dB)
- 3-axis accelerometer (±2g, ±4g, ±8g, ±16g)
- 16-bit light sensor (0-64000 lux)
- 13-bit temp sensor (-40 – 85C)
- 2Mb flash for firmware updates and data storage

2.2 Battery

The SensorBug requires 1 CR2032 3V lithium battery.

2.2.1 Changing the Battery

2.2.1.1 SensorBugs With Plastics

To change the battery on the SensorBug, first unscrew and remove the lower (blue) part of the plastics using a quarter in the slot on the bottom of the device. Next, remove the board from the top (clear) part of the plastics by pulling up on the metal battery holder in front of the + symbol. Remove the battery by pushing it from the side near the + symbol and the pulling it out from the opposite side. A small plastic or wooden tool (such as a pencil or golf tee) can help in pushing the battery out, but DO NOT USE A METAL TOOL. Replace the battery with a new CR2032, making sure the + symbol is facing up towards the + symbol on the battery holder.

Place the board back into the clear part of the plastics by angling the board into the plastics button side first. The battery should be facing out the open end of the plastics as it was when the board was removed. Make sure the buttons on the board line up with the rubber buttons on the plastics and the flat edge on the opposite side of the board lies up evenly with the flat edge on the plastics. When the board is lined up push down on the board near the flat edge to seat the board in the plastics. Last, screw the bottom (blue) part of the plastics back into the top piece.

2.2.1.2 SensorBugs Without Plastics (Low Profile – LP)

Remove the battery by pushing it from the side near the + symbol and the pulling it out from the opposite side. A small plastic or wooden tool (such as a pencil or golf tee) can help in pushing the battery out, but DO NOT USE A METAL TOOL. Replace the battery with a new CR2032, making sure the + symbol is facing up towards the + symbol on the battery holder.

2.2.2 Maximizing Battery Life

Connect Only When Necessary

Connected devices will consume more energy, so only connect to a device when necessary and stay connected for as short a time as possible to extend battery life.

Disable Unused Sensors

If data from a specific sensor is not needed, disable the sensor. When a sensor is disabled it will be powered down on the device, conserving energy and extending battery life.

Longer Advertising Interval

A longer advertising interval will make the device less responsive when not connected (connection establishment will be slower and the device will advertise slower), but will increase battery life. A shorter advertising interval will

make the device more responsive when not connected (connection establishment will be faster and the device will advertise faster), but will decrease battery life.

3 Basic Operation

Note: For SensorBugs with plastics, not the low profile (LP) version where the board is visible, the buttons will need to be pressed firmly to activate.

3.1 Powering On

The SensorBug will automatically power on when a battery is inserted. If the SensorBug is off, it can be turned on by momentarily pressing both buttons at the same time. Both LEDs will blink twice to indicate that the SensorBug has turned on. When powered on, pressing either button individually will turn on the corresponding LED, indicating that the device is powered on.

3.2 Powering Off

The SensorBug can be powered off by momentarily pressing both buttons at the same time. Both LEDs will remain on briefly and then turn off, indicating the SensorBug has powered off. When powered off, pressing either button individually will not turn on the corresponding LED, indicating that the device is powered off.

3.3 Pairing

To put the SensorBug into pairing mode, hold either button for ~1 second until the blue LED start blinking. It will remain in pairing mode for 30 seconds or until a device connects. Either switch can be pressed to exit pairing mode early. The blue LED will stop blinking when pairing mode has been exited.

3.4 Clearing Paired Devices

To clear all paired devices from the SensorBug, hold either button for ~5 seconds. After ~1 second the blue LED will start blinking – continue to hold the button until the blue LED stops blinking. At this point the pairing has been cleared and the button can be released.

3.5 Factory Reset

If the SensorBug needs to be returned to its factory settings, first completely remove the battery. Next, reinsert the battery while holding down button 1 (labeled S1). The blue and green LEDs will blink 4 times to indicate that the SensorBug has been factory reset. Factory resetting the SensorBug will also clear all paired devices.

3.6 Using the Device

The SensorBug ships in the powered off state. Once powered on it will need to be paired with a central device, until it is paired it will not start advertising. Once paired, unless configured otherwise, it will advertise the temperature data at an interval of approximately 1 second. The temperature data can then be collected from the ad data by any device without having to connect. If a higher sensor data rate is needed a paired central device can connect to the SensorBug and enable notifications to receive streaming sensor data. While connected the central device can also enable other sensors, configure alerts, change settings and update the firmware. If alerts are enabled the central device should monitor the ad data for alert flags while disconnected. If an alert flag is set, it can be cleared by connecting to the SensorBug and writing to the appropriate alert characteristic.

4 Advertising (Unconnected LE Interface)

By default the SensorBug will not advertise, but once it has been paired with a device it will always advertise connectable so paired devices can connect to it. If an unpaired device connects, the SensorBug will automatically disconnect with it. Unless the SensorBug is in pairing mode, then it will allow the unpaired device to stay connected so it may pair.

4.1 Advertising Data Structures

SensorBug advertising data will consist of the following three advertising data structures: Flags (0x01), 16-Bit Incomplete Service List (0x03) and Manufacturer Specific Data (0xFF).

Definitions and formats for the different advertising data structure types can be found here:
https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=291904

4.1.1 Flags

This data structure is mandatory for LE devices. The BR/EDR not supported flag will always be set as this is not a dual mode device. The LE limited discoverable flag will only be set when the device is pairable, allowing this flag to be used to filter out pairable devices. When not pairable the LE general discoverable flag will be set instead.

4.1.2 16-Bit Incomplete Service List

The service list will only contain the UUID for the Device Information Service 0x180A. It may seem unnecessary to advertise this service since all LE devices should support the service, but it is included in order to provide iOS apps a service to filter by for background discoveries. Without advertising a specific service iOS will not discover devices while scanning in the background.

4.1.3 Manufacturer Specific Data

The manufacturer specific data structure is used to transmit device specific information such as device PID, sensor data, battery level and alert flags. The manufacturer specific data consists of the five sections shown in the table below. It can range from 6 – 31 bytes in length. The Header, Template ID Byte and Static Data will always be included, the Encryption LSB and Dynamic Data are optional fields.

All multibyte fields within the sections are in Little Endian order.

	Header	Template ID Byte	Encryption Key LSB	Static Data	Dynamic Data
Required	Yes	Yes	No	Yes	No
Length (Bytes)	6	1	1	Variable	Variable

4.1.3.1 Header

The Header defines the length of the manufacturer specific data, as well as identifies the device.

	MSD Length	MSD Ad Type	BlueRadios CID	Major PID	Minor PID
Length (Bytes)	1	1	2	1	1
Possible Values	5-27	0xFF (MSD)	0x0085	0x02	0x00

MSD Length
The total number of bytes in the manufacturer specific data, not including the length byte.

MSD Ad Type
Identifies this data structure as manufacturer specific data.

BlueRadios CID (Company Identifier)
Identifies this data structure as being defined by BlueRadios.

Major/Minor PID (Product Identifier)
Identifies the specific BlueRadios device, for the SensorBug the Major PID is 0x02 (Sensor) and the Minor PID is 0x00 (SensorBug).

4.1.3.2 Template ID Byte

The Template ID Byte defines whether encryption is enabled, whether the device is pairable and the structure of the static data.

Bit	7	6	5	4	3	2	1	0
Value	Encrypted[1]	Pairable[1]	Template ID (TID) = 0x3C					

Encrypted
If this bit is set then the Encryption LSB will follow the Template ID Byte and all data after the Encryption LSB will be encrypted.

Pairable
If this bit is set the device is currently in pairable mode.

Template ID (TID)
The Template ID (TID) defines the structure of the Static Data, which for the SensorBug is TID 0x3C.

4.1.3.3 Encryption Key LSB

This byte will only be present if the Encrypted bit is set in the Template ID Byte. If set then all data after this byte will be encrypted with using AES-128 with the key stored in the Pairing Management Service Ad Encryption Key Characteristic. If necessary, padding bytes will be added via the dynamic data to make sure the encrypted value is 128 bits in length.

When encrypted this byte will be the LSB of the encryption key and will be incremented if the encryption key is changed, so paired devices should compare the value of this byte to the LSB of the ad encryption key they have stored and connect and refresh their local key if different.

4.1.3.4 Static Data (TID 0x3C)

	Battery Level	Config Counter
Length (Bytes)	1	1
Possible Values	0-100, 0xE0,0xEE	0-255

4.1.3.4.1 Battery Level

Current battery level: 0-100% (0x00-0x64), 0xE0=Unknown, 0xEE=External Power.

4.1.3.4.2 Config Counter

Prevents the need for a connect to check for configuration value changes. Value will increments each time the device's configuration is changed and rolls over automatically from 255 to 1. It will only be set to 0 after a reset. This values matches the value in the BR Utilities Service Config Counter Characteristic. For the SensorBug, if the Accel Config, Light Config, or Temp Config characteristic values change, this value will be incremented.

4.1.3.5 Dynamic Data

Dynamic Data may or may not be present depending on the device configuration. If present the Dynamic Data field will consist of 1 or more dynamic data structures. The data structures are never guaranteed to be in the same order. Each data structure consists of the following fields:

	Dynamic Data Structure ID	Alert Data	Data
Length (Bytes)	1	0-1	0-Variable

4.1.3.5.1 Dynamic Data Structure ID

Bit	7	6	5	4	3	2	1	0
Value	Alert Enabled Flag	Data Flag	Data Type					

Alert Enabled Flag
If this flag is set, alerts are enabled for this data type and the *Alert Data* field will be present. If the flag is not set the Alert Data field will not be present. Alerts can be enabled or disabled by writing the enableAlert flag in the modeFlags field of the sensor service config characteristic.

Data Flag
If this flag is set, the *Data* field will be present. If it is not set the Data field will not be present and this data structure is just being used for alerts. Data can be enabled or disabled by writing the enableAdData flag in the modeFlags field of the sensor service config characteristic.

Data Type

Data Type Value	Data Type	Data Structure Data Len (Bytes)
0x01	Accelerometer	2
0x02	Light	2-3
0x03	Temperature	2
0x2F	Pairing	1
0x3F	Encryption Padding	Variable

4.1.3.5.2 Alert Data

This field will only be present when the Alert Enabled Flag is set in the Dynamic Data Structure ID.

Bit	7	6	5	4	3	2	1	0
Value	Alert Flag	Reserved	Alert Counter – 0-63					

Alert Flag
If this flag is set, an alert has occurred for this data type. The alert can be cleared by writing 0x00 to the alert characteristic of the service that the alert occurred on.

Alert Counter (Alert ID)
Increments each time the alert flag is set from 0 to 1, will roll over automatically from 63 to 1. Will only be set to 0 on a reset.

4.1.4 Data

4.1.4.1 Accelerometer Data

If accelerometer ad data is enabled (enableAdData in accelerometer config modeFlags) then the accelerometer app type and alert characteristic are advertised so position or axis alerts can be monitored. Raw accelerometer axis data cannot be advertised.

	Accel App Type	Accel Alert Characteristic Data Byte
Length (Bytes)	1	1
Value	appType	data

4.1.4.2 Light Data

If light ad data is enabled (enableAdData in light config modeFlags) then the value of the light data characteristic will be advertised.

	Light Data Characteristic
Length (Bytes)	2-3
Value	data characteristic

4.1.4.3 Temperature Data

If temperature ad data is enabled (enableAdData in temp config modeFlags) then the value of the temperature data characteristic will be advertised.

	Temperature Data Characteristic
Length (Bytes)	2
Value	data characteristic

4.1.4.4 Pairing Data

In the Pairing Data structure the alert/data flags of the Data Structure ID are repurposed:

Bit	7	6	5	4	3	2	1	0
Value	N/A	New Device Paired	Pairing Data Type = 0x2F					

The New Device Paired flag will be set when a new device is paired (except on the first device). This flag can be cleared using the Pairing Management Service – by either reading the paired device list or sending the Clear New Paired command.

4.1.4.5 Encryption Padding Data

If encryption is enabled, encryption padding bytes may be added to the ad data. These will just be randomly generated bytes that can be ignored.

4.2 Scan Response Data Structures

SensorBug scan response data will consist of the following three advertising data structures: TX Power Level (0x0A) and Complete Name (0x09).

Definitions and formats for the different scan response data structure types can be found here: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=291904

4.2.1 TX Power Level

The TX Power Level byte is included so a remote device can estimate the distance to the SensorBug using path loss.

4.2.2 Complete Name

This data structure contains the complete name of the device. The SensorBug name will default to SensorBug###### where ###### denotes the Bluetooth Device Address Lower Address Part (LAP). For example a SensorBug with a Bluetooth Address of ECFE7E123456 would have a default name of SensorBug123456.

5 GATT Services (Connected LE Interface)

The following is a complete list of all the GATT services supported by the SensorBug.

Service	UUID
Bluetooth Defined Services (16-bit UUID)	
Device Information	180A
Battery	180F
Link Loss	1803
Immediate Alert	1802
TX Power	1804
BlueRadios Proprietary Services (128-bit UUID)	
BR Accelerometer	9DC84838-7619-4F09-A1CE-DDCF63225B10
BR Light	9DC84838-7619-4F09-A1CE-DDCF63225B20
BR Temperature	9DC84838-7619-4F09-A1CE-DDCF63225B30
BR OTA Firmware Update	EDB10B81-1F98-45EA-B403-E27A1A39A790
BR Pairing Management	3188AC28-72D4-4006-BD96-C6C4BC6153A0
BR Utils	4216378B-2073-47C4-83D6-A7DF9A61EC30

5.1 Device Information Service

This service provides various information about the device. The full specification can be found here:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=244369

The SensorBug exposes the following Device Information Service characteristics, most importantly the firmware revision string:

- Manufacturer Name String = “BlueRadios, Inc.”
- Model Number String = “BR-BUTTON-S3A”
- Hardware Revision String = “E”
- Firmware Revision String = “1.3.0.0”
- Software Revision String = “1”
- Vendor ID = 0x0085 (BlueRadios)
- Product ID = 0x0002 (SensorBug)
- PNP ID = 0x0100

5.2 Battery Service

This service provides the device’s battery level. The full specification can be found here:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=245138

5.3 Link Loss Service

This service can be used to make the SensorBug flash its leds if it becomes disconnected from a specific device. The full specification can be found here:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239391

5.4 Immediate Alert Service

This service can be used to make the SensorBug flash its leds to help locate the device. The full specification can be found here:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239390

5.5 TX Power Service

This service exposes the device’s current transmit power level. The full specification can be found here:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239393

5.6 BR Accelerometer Service

The BR Accelerometer Service allows the accelerometer to be configured and read.

Characteristic	UUID
Config	9DC84838-7619-4F09-A1CE-DDCF63225B11
Data	9DC84838-7619-4F09-A1CE-DDCF63225B12
Alert	9DC84838-7619-4F09-A1CE-DDCF63225B13

5.6.1 Config Characteristic

This characteristic is used to configure the service. Can write all or part of the config structure – allowing just enable/disable if needed.

5.6.1.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.6.1.2 Value Structure

```
typedef struct
{
 uint8  enableSensor;
 accelSvcModeFlags_t modeFlags;
 accelSvcAxisFlags_t axisFlags;
 uint8  dataRate;
 uint8  notiDataRate;
 uint8  range;
 uint8  hpfCutOffFreq;
 uint8  alertMode;
 uint8  alertThreshold;
 uint8  alertDuration;
 uint16 reserved;
 uint8  appType;
}accelSvcConfig_t;
```

5.6.1.2.1 **enableSensor**

0 = Sensor Disabled (Powered Off), 1 = Sensor Enabled (Powered On) DEFAULT = 0

5.6.1.2.2 **modeFlags**

```
typedef struct
{
  uint8 enableAdData:1;
  uint8 enableAlert:1;
  uint8 enableLowPowerMode:1;
  uint8 enableHpfOnData:1;
} accelSvcModeFlags_t;
```

enableAdData
Enables accelerometer data in ad data. DEFAULT = 1

enableAlert
Enables accelerometer alerts in ad data. DEFAULT = 0

enableLowPowerMode
Enables accelerometer low power mode, ~half the current consumption, but 1/16th the resolution. DEFAULT = 1

enableHfpOnData
Enables a high pass filter on the acceleration data (removes the effect of gravity). DEFAULT = 1

5.6.1.2.3 axisFlags

```
typedef struct
{
  uint8 enableX:1;
  uint8 enableXAlert:1;
  uint8 enableY:1;
  uint8 enableYAlert:1;
  uint8 enableZ:1;
  uint8 enableZAlert:1;
} accelSvcAxisFlags_t;
```

enable
Enables the axis on the accelerometer. DEFAULT = 1

enableAlert
Enables alerts on the axis on the accelerometer, enable must bet set to 1. DEFAULT = 1

5.6.1.2.4 dataRate / notiDataRate

Sets the data rate (sampling rate) of the sensor. dataRate is the rate that will be used normally, notiDataRate is the rate that will be used when notifications are enabled on the data characteristic. DEFAULT = 0x02 (10Hz) for both dataRate and notiDataRate.

Value	Data Rate
0x01	1 Hz
0x02	10 Hz
0x03	25 Hz
0x04	50 Hz
0x05	100 Hz

5.6.1.2.5 range

Sets the range of the accelerometer. DEFAULT = 0x00 (±2g)

Value	Range
0x00	±2g
0x01	±4g
0x02	±8g
0x03	±16g

5.6.1.2.6 **hpfCutoffFreq**

Sets the cutoff frequency for the high pass filter. The frequency value depends on dataRate. DEFAULT = 0x00

$$fc \cong (\text{dataRate (Hz)} / 50) / (2^{\text{hpfCutoffFreq}})$$

Value	dataRate = 0x01 (1 Hz)	dataRate = 0x02 (10 Hz)	dataRate = 0x03 (25Hz)	dataRate = 0x04 (50 Hz)	dataRate = 0x05 (100 Hz)
0x00	.02 Hz	.2 Hz	.5 Hz	1 Hz	2 Hz
0x01	.008 Hz	.08 Hz	.2 Hz	.5 Hz	1 Hz
0x02	.004 Hz	.04 Hz	.1 Hz	.2 Hz	.5 Hz
0x03	.002 Hz	.02 Hz	.05 Hz	.1 Hz	.2 Hz

5.6.1.2.7 **alertMode**

Sets the alert mode of the accelerometer. DEFAULT = 0x00 (Position Change)

Value	Alert Mode
0x00	Position Change
0x01	Acceleration Start
0x02	Acceleration Stop
0x03	Reserved
0x04	Free Fall

Position Change

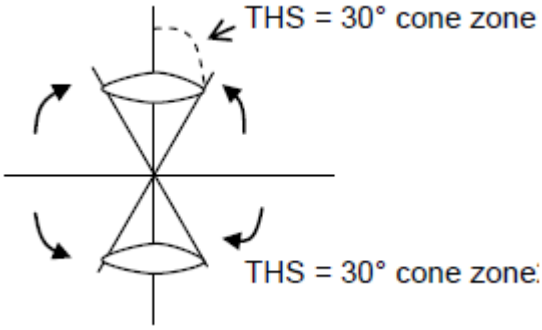
This mode is used to detect device position changes and is used for the BlueSense garage sensor and vermin alert. This mode generates an alert when an alert enabled axis enters a virtual cone around the axis for alertDuration. The angle of the cone is set by alertThreshold. An alert will be generated each time the device changes positions.

$$\text{ConeAngle} = \sin\text{-}1(\text{alertThreshold (G)})$$

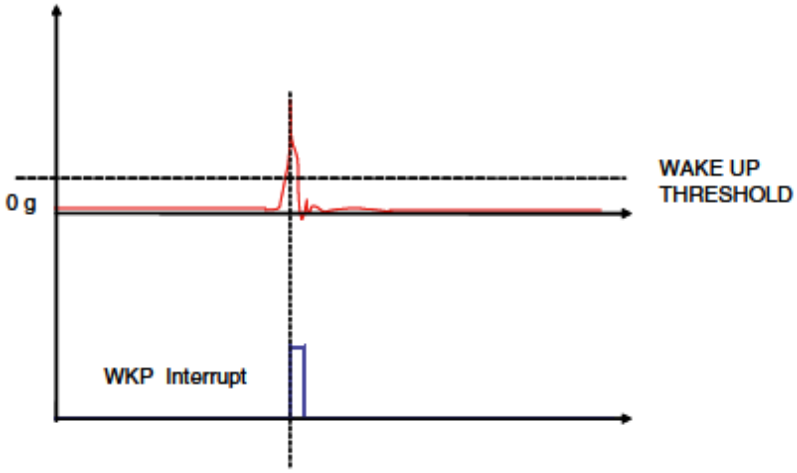
When using this mode *range* should be set to 2G since the position is detected based on gravity which is 1G. Also, at a 2G range the alert threshold is approximately equal to the angle of the cone.

Example: At *range* = 0, alertThreshold = 32, cone angle = 30 degrees

$$\sin\text{-}1(32 * .015625) = 30$$

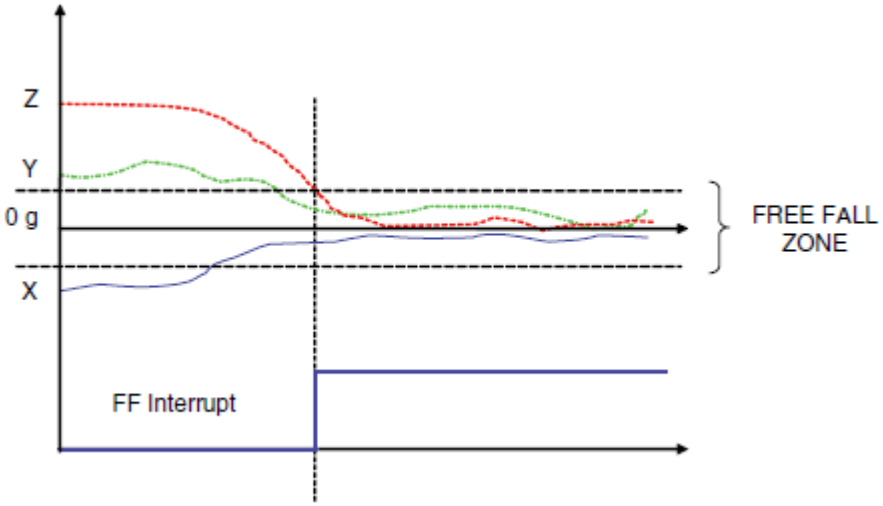


Acceleration Start
This mode alerts when acceleration starts on any alert enabled axis. It is used for the BlueSense vibration and glass break modes.



Acceleration Stop
This mode alerts when acceleration stops on any alert enabled axis.

Free Fall
This mode alerts when the free fall state is detected – when all axes approach zero g.



5.6.1.2.8 **alertThreshold**

This value works together with alertDuration. An alert will be generated when the magnitude of the acceleration on any axis enabled in axisFlags exceeds the alertThreshold for alertDuration. DEFAULT = 45

Value = 0-127

The units of alertThreshold depend on the range setting:

Alert Threshold Units (mg) = Range (mg) */ 128*

range	Threshold Units
0x00 (±2g)	15.625 mg
0x01 (±4g)	31.25 mg
0x02 (±8g)	62.5 mg
0x03 (±16g)	125 mg

Alert Threshold (mg) = Threshold Units (mg) ** alertThreshold*

5.6.1.2.9 **alertDuration**

This value works together with alertThreshold. An alert will be generated when the magnitude of the acceleration on any axis enabled in axisFlags exceeds the alertThreshold for alertDuration. DEFAULT = 0

Value = 0-127

The units of alertDuration depend on the dataRate setting:

Alert Duration Units (ms) = 1000 / dataRate (Hz)

dataRate	Duration Units
0x01 (1 Hz)	1000 ms

0x02 (10 Hz)	100 ms
0x03 (25 Hz)	40 ms
0x04 (50 Hz)	20 ms
0x05 (100 Hz)	10 ms

Alert Duration (s) = alertDuration / dataRate (Hz)

5.6.1.2.10 **reserved**

Set to 0.

5.6.1.2.11 **appType**

This value will be advertised with the accel data to let other devices know what application the accelerometer is currently being used for. Any value can be written from 0-255, the values used by BlueSense/BLE Cloud are shown below. DEFAULT = 0x05

Value	App Type
0x00	Vibration
0x01	Glass Break
0x02	DEPRECATED
0x03	Vermin Alert
0x04	Position Change
0x05	Garage Door
0x06	Sliding Door Security Bar

5.6.2 Data Characteristic

This characteristic is to read acceleration data. The data values are already in milligrav and do not need to be converted.

5.6.2.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.6.2.2 Value Structure

```
typedef struct {
 int16 xData;
 int16 yData;
 int16 zData;
} accelSvcData_t;
```

Signed 16-bit values for each axis in mg (milligrav).

5.6.3 Alert Characteristic

This characteristic is to read/clear alert data. Write 0x00 to clear the count and the alert flag in the advertising. If position sensing is enabled the data itself will not be cleared, so you can read the alert value at any time to get the current position.

5.6.3.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.6.3.2 Value Structure

```
typedef struct {
 uint16 count;
 accelSvcAlertData_t data;
} accelSvcAlert_t;
```

5.6.3.2.1 **count**

Count is the number of alerts that have occurred since the last time alerts were cleared.

5.6.3.2.2 **data**

```
typedef struct {
 uint8 value:6;
 uint8 type:2;
} accelSvcAlertData_t;
```

Type is a 2 bit field that identifies the type of data contained in the value field.

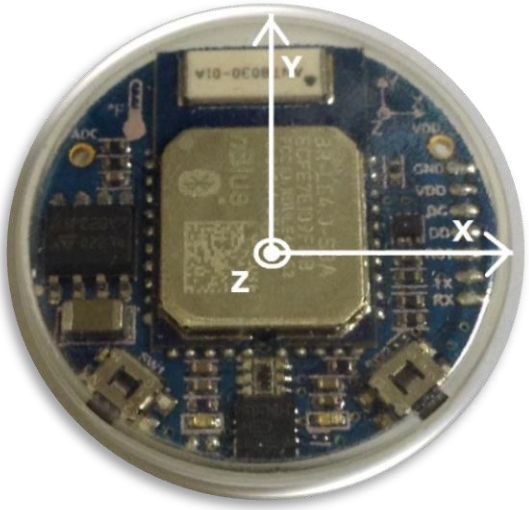
Value	Data Type
0x00	Position Change
0x01	Acceleration Start
0x02	Acceleration Stop
0x03	Free Fall

Value is a 6-bit field that identifies the current position of the device or what axis generated an alert.

Position Change Values (Type == 0x00)

In position change mode, value will be set to one of the values in the table below. The value will indicate which position the device is currently in. When alerts are cleared in this mode the data itself will not be cleared, so you can read the alert value at any time to get the current position.

Value	Position
0x01	X- (standing on right edge)
0x02	X+ (standing on left edge)
0x04	Y- (standing on top edge)
0x08	Y+ (standing on bottom edge)
0x10	Z- (laying flat, upside down)
0x20	Z+ (laying flat)
0x80	Position Unknown



Acceleration Start/Stop/Free Fall Values (Type == 0x01,0x02,0x03)

In acceleration start/stop or freefall mode, value will be set to any combination of the values below, indicating which axes the alert occurred on.

Value	Alert Axis
0x01	X Axis Alert
0x02	Y Axis Alert
0x04	X Axis Alert

5.6.4 Service Specific Errors

Value	Error
0x81	Invalid State – Returned if the data or alert characteristics is read when the sensor is disabled.

5.7 BR Light Service

The BR Light Service allows the light sensor to be configured and read.

Characteristic	UUID
Config	9DC84838-7619-4F09-A1CE-DDCF63225B21
Data	9DC84838-7619-4F09-A1CE-DDCF63225B22
Alert	9DC84838-7619-4F09-A1CE-DDCF63225B23
Stats	9DC84838-7619-4F09-A1CE-DDCF63225B24

5.7.1 Config Characteristic

This characteristic is used to configure the service. Can write all or part of the config structure – allowing just enable/disable if needed.

5.7.1.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.7.1.2 Value Structure

```
typedef struct
{
 uint8 enableSensor;
 lightSvcModeFlags_t modeFlags;
 uint16 dataRate;
 uint16 notiDataRate;
 uint16 alertLow;
 uint16 alertHigh;
 uint16 alertFaults;
 uint16 reserved;
 uint8   range;
 uint8   resolution;
 uint16 alertResetCount;
 uint16 alertResetDiff;
}lightSvcConfig_t;
```

5.7.1.2.1 enableSensor

0 = Sensor Disabled (Powered Off), 1 = Sensor Enabled (Powered On). DEFAULT = 0

5.7.1.2.2 modeFlags

```
typedef struct
{
 uint8 enableAdData:1;
 uint8 enableAlert:1;
 uint8 reserved:1;
 uint8 enableIR:1;
 uint8 enableAlertsIndividually:1;
 uint8 enableAlertLow:1;
 uint8 enableAlertHigh:1;
} lightSvcModeFlags_t;
```

enableAdData
Enables light data in ad data. DEFAULT = 1

enableAlert
Enables light alerts in ad data. DEFAULT = 0

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings.

enableIR
Enables IR sensing instead of ambient light sensing mode. DEFAULT = 0

enableAlertsIndividually
When enabled, the enableAlertLow and enableAlertHigh flags will be functional.
DEFAULT = 1

enableAlertLow
When enabled, low temperature alerts will be enabled.
DEFAULT = 1

enableAlertHigh
When enabled, high temperature alerts will be enabled.
DEFAULT = 1

5.7.1.2.3 dataRate / notiDataRate

Sets the data rate (sampling rate) of the sensor. dataRate is the rate that will be used normally, notiDataRate is the rate that will be used when notifications are enabled on the data characteristic. DEFAULT = 10000 (dataRate), 250 (notiDataRate)

0 – 65535 ms

5.7.1.2.4 alertLow / alertHigh

Sets the low and high alert levels. An alert will be generated if the level is less than or equal to alertLow or greater than or equal to alertHigh. DEFAULT = 10 (alertLow), 1000 (alertHigh)

The alert levels must be set as a raw sensor value. A lux value can be converted to a raw value with the following code:

```
const static uint16 rangeVal[] =  { 1000, 4000, 16000, 64000 };
const static uint16 resolutionMaxVal[] = { 65535, 4095, 255, 15 };

rawValue = lux / (rangeVal[range] / resolutionMaxValue[resolution])
```

5.7.1.2.5 alertFaults

Sets the number of consecutive out of range values that must be measured before an alert will be generated. For example if alert faults is set to 2, then 2 consecutive out of range values must be measured before an alert will be triggered. DEFAULT = 1

5.7.1.2.6 reserved

Set to 0.

5.7.1.2.7 range

Sets the range of the light sensor.

Value	Range
0x00	1000 lux
0x01	4,000 lux
0x02	16,000 lux
0x03	64,000 lux

5.7.1.2.8 resolution

Sets the resolution of the light sensor.

Value	Resolution	Sample Time
0x00	16-bit	90.39 ms
0x01	12-bit	5.65 ms
0x02	8-bit	.35 ms
0x03	4-bit	.022 ms

5.7.1.2.9 alertResetCount

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings. DEFAULT = 3

5.7.1.2.10 alertResetDiff

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings. DEFAULT = 100

5.7.2 Data Characteristic

This characteristic is used to read raw light sensor data. The data is raw sensor data and needs to be converted to lux by the client device.

5.7.2.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.7.2.2 Value Structure

The light value structure format will depend on the resolution setting. If the resolution is 4/8 bit the data field will be 1 byte, but if the resolution is 12/16 bit then the data field will be 2 bytes. The dataLen field of info will specify the size of the data field.

```
typedef struct
{
```

```
 uint8  info;
 uint8 data;
}lightSvcData1_t;
```

or

```
typedef struct
{
 uint8  info;
 uint16 data;
}lightSvcData2_t;
```

5.7.2.2.1 **info**

Bit	7	6	5	4	3	2	1	0
Sensor	IR	-	*resolution*		*range*		dataLen	

dataLen
Specifies whether the data field is 1 or 2 bytes in length.

IR
If set, the data is infrared light instead of ambient light.

5.7.2.2.2 **data**

The data is raw sensor data and needs to be converted to lux by the client device. The data value can be converted to lux with the following code, where range and resolution are the values from the info byte:

```
const static uint16 rangeVal[] =  { 1000, 4000, 16000, 64000 };
const static uint16 resolutionMaxVal[] = { 65535, 4095, 255, 15 };

lux = data * (rangeVal[range] / resolutionMaxValue[resolution])
```

5.7.3 Alert Characteristic

This characteristic is to read/clear the alert counter. Write 0x00 to clear the count and the alert flag in the advertising.

5.7.3.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.7.3.2 Value Structure

```
uint16 alertCount;
```

alertCount is the number of alerts that have occurred since the last time alerts were cleared.

5.7.4 Stats Characteristic

This characteristic is to track min, max and average light readings. Count and sum can be used to calculate an average. Writing any value to this characteristic will reset the stats.

5.7.4.1 Permissions

Property	Required Auth Level
Read	User
Write	User

5.7.4.2 Value Structure

```
typedef struct
{
 uint16 count;
 uint16 min;
 uint16 max;
 uint32 sum;
}lightSvcStats_t;
```

5.7.5 Service Specific Errors

Value	Description
0x81	Invalid State – Returned if the data or alert characteristics is read when the sensor is disabled.

5.8 BR Temperature Service

The BR Temperature Service allows the temperature sensor to be configured and read.

Characteristic	UUID
Config	9DC84838-7619-4F09-A1CE-DDCF63225B31
Data	9DC84838-7619-4F09-A1CE-DDCF63225B32
Alert	9DC84838-7619-4F09-A1CE-DDCF63225B33
Stats	9DC84838-7619-4F09-A1CE-DDCF63225B34

5.8.1 Config Characteristic

This characteristic is used to configure the service. Can write all or part of the config structure – allowing just enable/disable if needed.

5.8.1.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.8.1.2 Value Structure

```
typedef struct
{
 uint8  enableSensor;
 tempSvcModeFlags_t modeFlags;
```

```
 uint16 dataRate;
 uint16 notiDataRate;
 int16  alertLow;
 int16  alertHigh;
 uint16 alertFaults;
 int8   reserved;
 uint16 alertResetCount;
 uint16 alertResetDiff;
}tempSvcConfig_t;
```

5.8.1.2.1 enableSensor

0 = Sensor Disabled (Powered Off), 1 = Sensor Enabled (Powered On). DEFAULT = 0

5.8.1.2.2 modeFlags

```
typedef struct
{
 uint8 enableAdData:1;
 uint8 enableAlert:1;
 uint8 reserved:1;
 uint8 enableAlertsIndividually:1;
 uint8 enableAlertLow:1;
 uint8 enableAlertHigh:1;
} tempSvcModeFlags_t;
```

enableAdData
Enables temp data in ad data. DEFAULT = 1

enableAlert
Enables temp alerts in ad data. DEFAULT = 0

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings.

enableAlertsIndividually
When enabled, the enableAlertLow and enableAlertHigh flags will be functional.
DEFAULT = 1

enableAlertLow
When enabled, low temperature alerts will be enabled.
DEFAULT = 1

enableAlertHigh
When enabled, high temperature alerts will be enabled.
DEFAULT = 1

5.8.1.2.3 dataRate / notiDataRate

Sets the data rate (sampling rate) of the sensor. dataRate is the rate that will be used normally, notiDataRate is the rate that will be used when notifications are enabled on the data characteristic. DEFAULT = 10000 (dataRate), 1000 (notiDataRate)

0 – 65535 ms

5.8.1.2.4 alertLow / alertHigh

Sets the low and high alert levels. An alert will be generated if the level is less than or equal to alertLow or greater than or equal to alertHigh. DEFAULT = 0 (32F alertLow), 604 (100F alertHigh)

The alert levels must be set as a raw sensor value. A Celsius value can be converted to a raw value with the following equation:

$$rawValue = tempCelsius / .0625$$

5.8.1.2.5 alertFaults

Sets the number of consecutive out of range values that must be measured before an alert will be generated. For example if alert faults is set to 3, then 3 consecutive out of range values must be measured before an alert will be triggered. DEFAULT = 3

5.8.1.2.6 reserved

Set to 0.

5.8.1.2.7 alertResetCount

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings. DEFAULT = 3

5.8.1.2.8 alertResetDiff

When alerts are enabled, if an alert is triggered another alert will not be issued until the data value goes back past the threshold by alertResetDiff for alertResetCount readings. DEFAULT = 8

5.8.2 Data Characteristic

This characteristic is used to read raw temperature sensor data. The data is raw sensor data and needs to be converted to Celsius by the client device.

5.8.2.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.8.2.2 Value Structure

int16 data;

The data value can be converted to Celsius using the following equation:

$$tempCelsius = data * .0625$$

5.8.3 Alert Characteristic

This characteristic is to read/clear the alert counter. Write 0x00 to clear the count and the alert flag in the advertising.

5.8.3.1 Permissions

Property	Required Auth Level
Read	User
Write	User
Notify	User

5.8.3.2 Value Structure

uint16 alertCount;

alertCount is the number of alerts that have occurred since the last time alerts were cleared.

5.8.4 Stats Characteristic

This characteristic is to track min, max and average temperature readings. Count and sum can be used to calculate an average. Writing any value to this characteristic will reset the stats.

5.8.4.1 Permissions

Property	Required Auth Level
Read	User
Write	User

5.8.4.2 Value Structure

```
typedef struct
{
 uint16 count;
 uint16 min;
 uint16 max;
 uint32 sum;
}lightSvcStats_t;
```

5.8.5 Service Specific Errors

Value	Description
0x81	Invalid State – Returned if the data or alert characteristics is read when the sensor is disabled.

5.9 BR OTA Firmware Update Service

The BR OTA Firmware Update Service allows a device's firmware to be updated over the air.

Characteristic	UUID
Control / Status	EDB10B81-1F98-45EA-B403-E27A1A39A791
Data	EDB10B81-1F98-45EA-B403-E27A1A39A792

5.9.1 Control / Status Characteristic

This characteristic is used to configure and control a firmware update, as well as monitor the status of the update. Only control, blockSize and flags can be written too – state and blocksFlashed are read only. Control can be written by itself.

5.9.1.1 Permissions

Property	Required Auth Level
Read	Admin
Write	Admin

5.9.1.2 Value Structure

```
typedef struct
{
 uint8  control;
 uint8  blockSize;
 brotaCtrlFlags_t  flags;
 uint8  state;
 uint16 blocksFlashed;
}brotaSvcCtrlStat_t;
```

5.9.1.2.1 control

Value	Command Name	Description
0x00	Disable Update	This command will cancel an update and free all related resource.
0x01	Start Update	This command will start a new update.
0x02	Halt Update	This command will halt an in progress update.
0x03	Resume Update	This command will resume a halted update.
0x04	Factory Reset	This command can be used to perform an over the air factory reset. Once the Factory Reset state has been entered, disconnect to apply the factory reset.

5.9.1.2.2 blockSize

Data is sent in blocks of blockSize (units of 256 bytes). When downloading the client will write a block of image data to the Data characteristic 20 bytes at a time using GATT write without response, then wait for a notification from the Control/Status characteristic indicating the block has been received, then send the next block. DEFAULT = 4 (1024 bytes), but a blockSize of 2 (512 bytes) is recommended.

5.9.1.2.3 flags

```
typedef struct
{
 uint8 enableFwuFactoryReset:1;
```

```
uint8 enableNBootLeds:1;
uint8 enableNBootDBG:1;
uint8 reserved:1;
}brotaCtrlFlags_t;
```

enableFwuFactoryReset
If enabled, the device will be factory reset after the update has completed. DEFAULT = 0

enableNBootLeds
Enables blinking the leds when the bootloader (nBoot) is flashing the new firmware. DEFAULT = 0

enableNBootDBG
Enabled UART debug output when the bootloader (nBoot) is flashing the new firmware. DEFAULT = 1

reserved
Set to 0.

5.9.1.2.4 **state**

Value	State	Description
0x00	Disabled	No firmware update in progress.
0x01	Initializing V1	Firmware update initializing, waiting for header BROTA Header V1 to be sent to the data characteristic.
0x02	Downloading	Firmware update download in progress, image data should be written to the data characteristic in blockSize blocks.
0x03	Halted	A firmware update has been halted.
0x04	Complete	Firmware has been downloaded successfully, disconnect to apply the update.
0x05	Factory Reset	Device is ready to be factory reset, disconnect to apply.
0x07	Initializing V2	Firmware update initializing, waiting for header BROTA Header V2 to be sent to the data characteristic.
0xFE	Firmware Invalid	This state will be entered if the firmware is detected to be incompatible with the device.
0xFF	CRC Error	This state will be entered if after downloading the firmware a CRC error is detected. Re-try the update.

5.9.2 Data Characteristic

This characteristic is used to read/write the header of the firmware image being programmed as well as the firmware image data.

Writing is only supported in the following state: Initializing V1, Initializing V2 and Downloading. For maximum throughput write without response should be used when writing the firmware image data in the Downloading state.

Reading is only supported in the following states: Downloading, Halted or Complete. Reading is used to read out the header of the image currently being programmed.

5.9.2.1 Permissions

Property	Required Auth Level

Read	Admin
Write	Admin

5.9.2.2 Value Structure

When in the Downloading, Halted or Complete state, reading data will return either the brotaHdrV1_t or brotaHdrV2_t structure (the structures can be told apart by the length). The structure gives information about the firmware that is being programmed, allowing an update to be resumed in the case of a disconnect in the middle of an update.

5.9.2.2.1 BROTA Header V1

Length = 16

```
typedef struct
{
 uint16 cid;
 uint16 pid;
 uint16 fwCrc;
 uint32 fwAddr;
 uint32 fwLen;
 uint16 hdrCrc;
} brotaHdrV1_t;
```

5.9.2.2.2 BROTA Header V2

Length = 18, same as V1 with mid (Module ID) added to prevent firmware intended to run on a specific module from being written to another module.

```
typedef struct
{
 uint16 cid;
 uint16 pid;
 uint16 mid;
 uint16 fwCrc;
 uint32 fwAddr;
 uint32 fwLen;
 uint16 hdrCrc;
} brotaHdrV2_t;
```

5.9.3 Client Firmware Update Procedure

Updates can be done using both .bru and .brz file types, but the update procedure is slightly different for each file type.

5.9.3.1 BRU Files

.bru - BlueRadios Update File.

BRU files are the latest BlueRadios firmware image file type, replacing the .brx and .brz types. BRU files consist of a section of image tags describing the image, followed by image data after the #IMAGE tag. Image data is organized in lines of 256 bytes (512 characters). Each line in the file is separated by a carriage return / line feed (0x0D0A).

5.9.3.1.1 BRU File Example

```
#BRU_VER=1
```

```
#FILE_NAME=3.4.0.0-D2.bru
#FW_VER=000304000000452B
#HDR_V1=850001005DA40018000000680E009746
#HDR_V2=8500010003005DA40018000000680E005D07
#IMAGE=
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3140005CB3134……..
173CB3130E7D0C9313240F3CDFC30000D2D301520A3CE……..
```

5.9.3.1.2 BRU Image Tags

- **BRU_VER** - ASCII string containing the version of the .bru file spec. Currently version 1 is the only version.
- **FILE_NAME** - ASCII string containing the file name. Should match the name of the file if not modified.
- **FW_VER** - Binary structure containing the firmware version of the image in the form Major.Minor.Bug.Dev along with the FW Id and a CRC of the structure for validation. Can be used to validate the firmware version against the filename.

```
typedef struct
{
   uint8 fwId;
   uint8  major;
   uint8 minor;
   uint8  bug;
   uint8  dev;
   uint16 crc;
} bruFwVer_t;
```

- **HDR_V1** - Binary structure containing BROTA Header V1. Sent to the BROTA service during the initialization phase and can also be used to validate the firmware image. The hdrCrc value can be used to validate the structure and the fwCrc can be used to validate the image.
- **HDR_V2** - Binary structure containing BROTA Header V2. Sent to the BROTA service during the initialization phase and can also be used to validate the firmware image. The hdrCrc value can be used to validate the structure and the fwCrc can be used to validate the image.
- **IMAGE** - Indicates the end of the BRU Image Tags, the next line will be image data.

5.9.3.1.3 BRU File Update Procedure

1. Parse the image tags and validate the file, storing off the #HDR_V1= and #HDR_V2= values. All lines after the #IMAGE= tag line are image data.
2. Write control and block size to the Control/Status characteristic. Set control to Start Update (0x01) and the block size to the client's preferred block size. The block size is in units of 256 bytes and defaults to 4 (1024 bytes).
3. Wait for a notification from Control/Status indicating the state has changed to either Initialize V1 or Initialize V2.
4. If the state has changed to Initializing V1, write the value of the #HDR_V1 tag to the Data characteristic. If the state has changed to Initializing V2, write the value of the #HDR_V2 tag to the Data characteristic.

5. Wait for a notification from Control/Status indicating the state has changed to Downloading.

6. Data is sent in blocks of blockSize. Write a block of image data (or less if at the end of the file) to the Data characteristic 20 bytes at a time using GATT write without response.

7. Wait for a notification from the Control/Status characteristic indicating the block has been received.

8. Repeat steps 6 and 7 until the entire file has been sent. After all data has been sent the state will change to Complete in the last notification from Control/Status.

9. Disconnect to apply the update.

5.9.3.2 BRZ Files

.brz - BlueRadios OTA Update File

BRZ were the original BlueRadios OTA firmware image file type, but have been replaced by .bru files. BRZ files consist of a single header line, followed by image data. Image data is organized in lines of 256 bytes (512 characters). Each line in the file is separated by a carriage return / line feed (0x0D0A).

5.9.3.2.1 BRZ File Example

850001005DA40018000000680E009746
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3140005CB3134……..
173CB3130E7D0C9313240F3CDFC30000D2D301520A3CE……..

5.9.3.2.2 BRZ Header

Binary structure containing the BROTA Header V1. Sent to the BROTA service during the initialization phase and can also be used to validate the firmware image. The hdrCrc value can be used to validate the structure and the fwCrc can be used to validate the image.

5.9.3.2.3 BRZ File Update Procedure

1. Write control and block size to the Control/Status characteristic. Set control to Start Update (0x01) and the block size to the client's preferred block size. The block size is in units of 256 bytes and defaults to 4 (1024 bytes).

2. Wait for a notification from Control/Status indicating the state has changed to Initializing V1. If the state changes to Initializing V2 the device cannot be updated with a .brz file and a .bru file will need to be used instead.

3. Write the first line of the .brz file to the Data characteristic. All lines after this are image data.

4. Wait for a notification from Control/Status indicating the state has changed to Downloading.

5. Data is sent in blocks of blockSize. Write a block of image data (or less if at the end of the file) to the Data characteristic 20 bytes at a time using GATT write without response.

6. Wait for a notification from the Control/Status characteristic indicating the block has been received.

7. Repeat steps 5 and 6 until the entire file has been sent. After all data has been sent the state will change to Complete in the last notification from Control/Status.

8. Disconnect to apply the update.

5.9.3.3 Halt/Resume Procedure

1. While in the Downloading state, setting control in the Control/Status characteristic to Halt Update (0x02) will halt the update process and put it in the Halted state. If the connection is lost in the middle of an update, the Halted state (0x03) will automatically be entered.

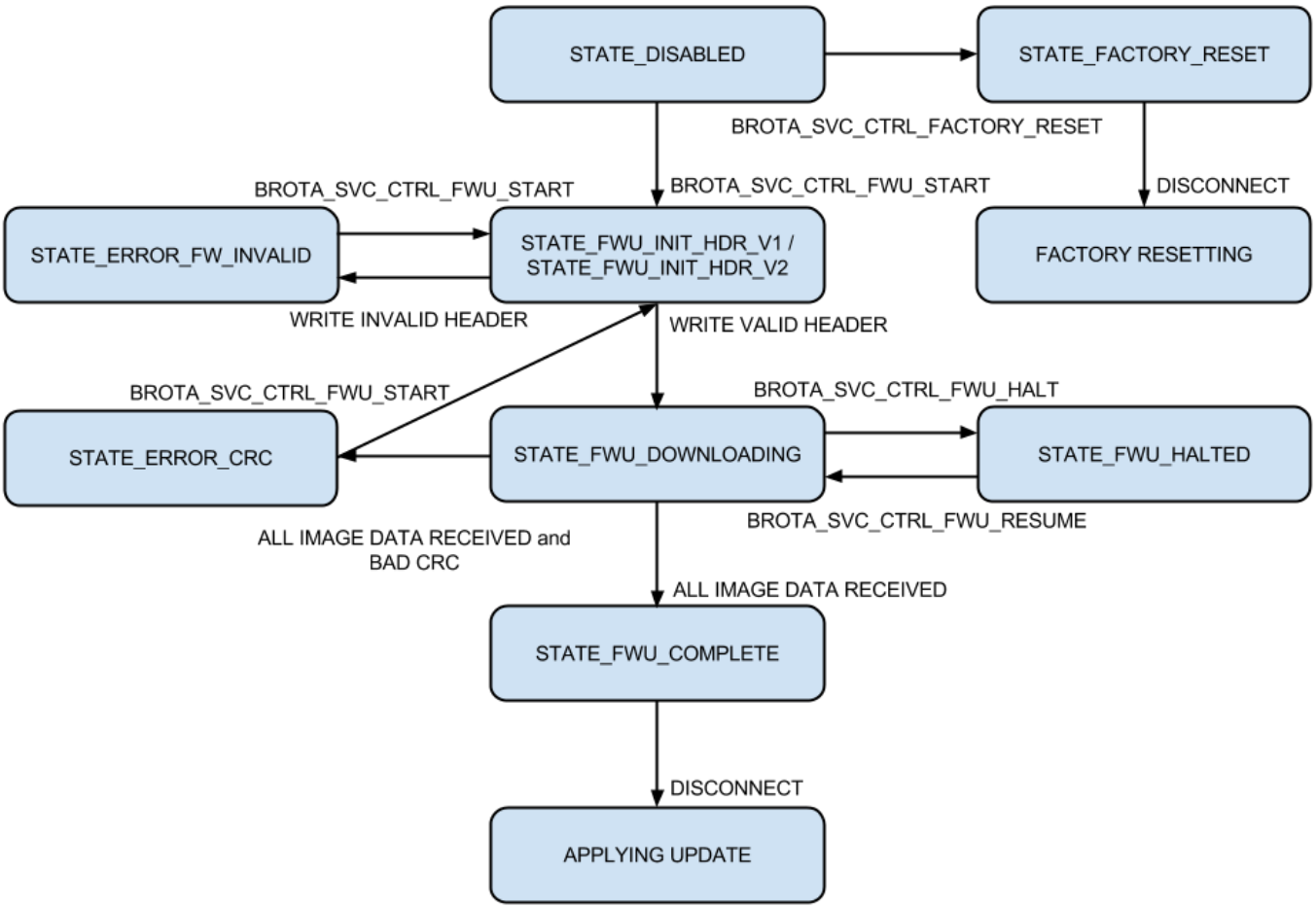
2. Upon reconnection from a disconnect, or when ready to resume read the Control/Status register to make sure the service is in the Halted state (0x03) and to find out how many blocks have been flashed. Then read the Data characteristic to find out what image was being written.

3. When ready to resume write Resume Update (0x03) to Control/Status control.

4. Wait for a notification from Control/Status indicating the state has changed to Downloading.

5. Resume sending image data. Start sending from when the update was halted by indexing into the image data by the number of blocksFlashed.

5.9.3.4 BROTA State Diagram



5.9.3.5 CRC Algorithm

```
uint8 temp;
uint16 i,j;

 // CRC the read data
 for ( i = 0; i < EXT_FLASH_PAGE_SIZE; i++ )
 {
   temp = fu->pageBuf[i];

   if ( temp != 0xFF )
   {
     fu->crc ^= temp;

     for ( j=0;j<8;++j )
     {
       if ( fu->crc & 0x0001 )
       {
         fu->crc >>= 1;
         fu->crc ^= 0xA001;
       }
       else
         fu->crc >>= 1;
     }
   }
 }
```

5.10 BR Pairing Management Service

The BR Pairing Management Service allows a device's pairing list to be managed, as well as security and authorization to be configured for the device.

Characteristic	UUID
Control / Status	3188AC28-72D4-4006-BD96-C6C4BC6153A1
Data	3188AC28-72D4-4006-BD96-C6C4BC6153A2
Config	3188AC28-72D4-4006-BD96-C6C4BC6153A3
Ad Encryption Key	3188AC28-72D4-4006-BD96-C6C4BC6153A4

5.10.1 Control / Status Characteristic

This characteristic is used to control and get status about the paired device list. Only the control portion of the structure can be written (control, param and addr), the rest of the structure is for status and is read only. The characteristic can be configured for notifications upon changes to encryption/device count. This is useful for getting a notification on the central device when the pairing process is complete and the connection is encrypted.

5.10.1.1 Permissions

Property	Required Auth Level
Read	None
Write	Admin (User if control = 0x00 (None), 0x07 (Write Own Name) or 0x0A (Write Own ASU Enable))
Notify	None

5.10.1.2 Value Structure

```
typedef struct
{
 uint8 control;
 uint8 param;
 uint8 addr[B_ADDR_LEN];  // B_ADDR_LEN = 6
 pmSvcStatFlags_t flags;
 uint8 authLevel;
 uint8 numPaired;
 uint8 maxPaired;
 uint8 newPairCounter;
} pmSvcCtrlStat_t;
```

5.10.1.2.1 control / param / addr

Control specifies a command to perform, param and addr are arguments used with some commands. For commands that require an address, this can be obtained by reading the paired device list.

command	Command Name	param	addr	Auth	Description
0x00	Read Paired Device List	N/A	N/A	User	Writing will reset the device list index for admin and user devices. For unpaired devices writing can be used to trigger pairing on unpaired iOS devices.
0x01	Delete Device	N/A	Device Address	Admin	Allows an admin to delete a paired device. Cannot delete self.

0x02	Delete Auth Level	Auth Level	N/A	Admin	Allows an admin to delete all devices with a specific auth level. Cannot delete self.
0x04	Set Device Auth Level	Auth Level	Device Address	Admin	Allows an admin to change the auth level of a paired device.
0x05	Enable Pairing	N/A	N/A	Admin	Allows an admin to make the device pairable over the air. Once this command has been set, disconnect to make the device pairable.
0x06	Read/Write Device Name	N/A	Device Address	Admin	Allows an admin to read/write the stored name of a paired device. Once the command has been set, write the name to the data characteristic.
0x07	Read/Write Own Name	N/A	N/A	User	Allows a paired device to read/write its own stored name. Once the command has been set, write the name to the data characteristic.
0x08	Clear New Paired	N/A	N/A	Admin	Allows an admin to clear the new paired flag from the advertising data.
0x09	Read/Write Device ASU Enable	0 = Disable, 1 = Enable	Device Address	Admin	Allows an admin to enable/disable active state updates for a paired device. Not supported by all devices.
0x0A	Read/Write Own ASU Enable	0 = Disable, 1 = Enable	N/A	User	Allows a paired device to enable/disable active state updates for itself. Not supported by all devices.

5.10.1.2.2 **flags**

The flags provide information about the connected client device, allowing the client to find out its current status.

```
typedef struct
{
 uint8 paired:1;
 uint8 encrypted:1;
 uint8 authenticated:1;
 uint8 admin:1;
 uint8 asuEnabled:1;
} pmSvcStatFlags_t;
```

paired
The connected client is paired (bonded). This doesn't imply that the connection is encrypted yet, just that the device is in the pairing list. (On AT.s modules a PAIRED event has not yet occurred)

encrypted
The connection is encrypted. (On AT.s modules a PAIRED event has occurred)

authenticated
MITM (Man in the Middle) protection was used during pairing to authenticate the connected client.

admin
The connected client is an administrator.

asuEnabled
Active state updates are enabled for the connected client.

5.10.1.2.3 authLevel

The authorization level of the connected client.

Value	Auth Level	Description
0x00	None	No permissions.
0x01	Guest	Temporary device with limited permissions.
0x02	User	Normal user with permissions to use the device but not configure it.
0xAA	Admin	Device administrator with global permissions.

5.10.1.2.4 numPaired

The number of devices currently in the paired list.

5.10.1.2.5 maxPaired

The maximum number of devices that can be paired with this device.

5.10.1.2.6 newPairCounter

The number of new devices paired since the last device reset, paired list read or the Clear New Paired command was issued.

5.10.2 Data Characteristic

This characteristic is used to read the paired device list as well as read/write device names.

5.10.2.1 Permissions

Property	Required Auth Level
Read	User (None if control = 0x07 (Read/Write own name))
Write	Admin (User if control = 0x07 (Read/Write own name))

5.10.2.2 Value Structure

If control is set to 0x06 (Read/Write Device Name) or 0x07 (Read/Write Own Name), reading from the data characteristic will return the name of the device selected by addr or the device's own name. The name will be the exact length of the name and will not be null terminated. The device name can be changed by writing a string up to 20 bytes with no null terminator.

If control is set to 0x00 (Read Paired Device List), reading this characteristic will return the paired device list. A long read should be used to read the list. The list will be returned as a list of the following structures, with length numPaired. If the paired device list is read by a user, the returned information will be limited to the device ID, device name and appearance.

```
typedef struct
{
 pmDevRecFlags_t flags;
```

```
 uint8  devId;
 uint8  addr[B_ADDR_LEN];
 uint8  addrType;
 uint8  authLevel;
 uint16 appearance;
 uint8  name[GAP_DEVICE_NAME_LEN];  // GAP_DEVICE_NAME_LEN = 21, null terminated string
} pmDevRec_t;
```

5.10.2.2.1 flags

```
typedef struct
{
 uint8 authenticated:1;
 uint8 connectedDevice:1;
 uint8 newDevice:1;
 uint8 limitedInfo:1;
 uint8 asuEnabled:1;
} pmDevRecFlags_t;
```

authenticated
MITM (Man in the Middle) protection was used during pairing to authenticate the device.

connectedDevice
This flag indicates that this device record is for the currently connected device.

newDevice
This device was newly paired since the last device reset, paired list read or the Clear New Paired command was issued.

limitedInfo
If the device list is read by a user this flag will be set. In this case the returned information will be limited to the device ID, device name and appearance. For security purposes the addr, addrType and authLevel will all be set to 0xFF.

asuEnabled
This flag indicates that active state updates are enabled for this device.

5.10.2.2.2 devId

Index of device in the pairing list. Some devices will keep an activity log that contains a devId with each entry, the devId can be used to map to a specific device in the pairing list.

5.10.2.2.3 addr

This is the public *Bluetooth* address of the paired device. If the device was using a Resolvable Private Random Address, this will be the resolved public address, not the random address.

5.10.2.2.4 addrType

This is the Bluetooth address type that the paired device is using.

Value	Address Type
0x00	Public Address
0x01	Static Random Address
0x02	Non-Resolvable Private Random Address

0xAA	Resolvable Private Random Address

5.10.2.2.5 authLevel

This is the authLevel of the paired device.

5.10.2.2.6 appearance

The external appearance of this device. Appearance values are defined by the *Bluetooth* SIG here: http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.gap.appearance.xml

5.10.2.2.7 name

The stored name of the device. The maximum name length is 20 and the string is null terminated.

5.10.3 Config Characteristic

This characteristic is used to configure security level, default authorization levels and advertising data encryption. All or part of the structure can be written, but if a passkey is written the full 4 bytes of the uint32 must be written. If security level is set to 0x02 and the passkey is not written a random passkey will be generated which can be read back after the write is complete. A new random passkey can be generated by writing securityLevel to 0 or 1 and then back to 2.

5.10.3.1 Permissions

Property	Required Auth Level
Read	Admin
Write	Admin

5.10.3.2 Value Structure

```
typedef struct
{
 pmSvcUserCfgFlags_t flags;
 uint8  securityLevel;
 uint8  defAuthLevel;
 uint32 passkey;
} pmSvcUserCfg_t;
```

5.10.3.2.1 flags

```
typedef struct
{
 uint8 enableAdEncrypt:1;
 uint8 disablePairingButton:1;
 uint8 disablePairingButtonSupported:1;
} pmSvcUserCfgFlags_t;
```

enableAdEncrypt
If set the advertising data will be encrypted with the key in the Ad Encryption Key characteristic.

disablePairingButton
If set, the hardware pairing button will be disabled and pairing can only be enabled over the air by an admin.

disablePairingButtonSupported
If set, it is possible to disable the hardware pairing button on this device. Read only – cannot be changed.

5.10.3.2.2 securityLevel

This value assigns the security level of the device, allowing an admin to disable the pairing requirement all together and allow any devices to connect or enforce authenticated pairing and require a passkey.

Changing the security level from 0x01 to 0x02 will force all currently paired devices to re-pair using the set passkey.

Value	Security Level
0x00	Pairing Not Required
0x01	Pairing Required
0x02	Authenticated Pairing Required (Passkey)

5.10.3.2.3 defAuthLevel

This value assigns the default authLevel that will be assigned to newly paired devices.

5.10.3.2.4 passkey

If securityLevel is set to 0x02 (Authenticated Pairing Required) this will be the passkey requested during pairing. If this value is not written when assigning the security level to 0x02, the passkey will be randomly generated and can be read back. The passkey is a 6-digit numeric value from 000000 – 999999. The value is written as an unsigned 32-bit value, not as a string. The passkey is always six digits, so the user would need to enter leading zeroes during the pairing sequence if the value was less than 100000.

5.10.4 Ad Encryption Key Characteristic

This characteristic is used to get the 128-bit key being used to encrypt the ad data if ad encryption is enabled. Notifications can be enabled to notify the app that the key has changed if a device is deleted. Anytime a paired device is removed from the pairing list a new key will be generated.

5.10.4.1 Permissions

Property	Required Auth Level
Read	User

5.10.4.2 Value Structure

16 byte AES-128 key.

5.10.5 Service Specific Errors

Value	Description
0x81	Invalid Addr – Returned if the device address cannot be found in the pairing list, or if an admin attempts to delete itself or change its auth level.
0x82	Invalid State – Returned if a device tires to enable pairing when the list is full, or attempts to write to data when command isn't set to a name change mode.

5.11 BR Utilities Service

The BR Utilities Service is a collection of utility characteristics that pertain to most BlueRadios devices. None of the characteristics are required for the service, so there could be only one characteristic in the service if the device didn't need any of the others.

Characteristic	UUID
Device Name	4216378B-2073-47C4-83D6-A7DF9A61EC32
Default Connection Parameters	4216378B-2073-47C4-83D6-A7DF9A61EC33
Current Connection Parameters	4216378B-2073-47C4-83D6-A7DF9A61EC34
Advertising Parameters	4216378B-2073-47C4-83D6-A7DF9A61EC3E
RF Power Levels	4216378B-2073-47C4-83D6-A7DF9A61EC36
Disconnect	4216378B-2073-47C4-83D6-A7DF9A61EC37
Public Device Address	4216378B-2073-47C4-83D6-A7DF9A61EC39
Config Counter	4216378B-2073-47C4-83D6-A7DF9A61EC3C
Blink LED	4216378B-2073-47C4-83D6-A7DF9A61EC42
Factory Value	4216378B-2073-47C4-83D6-A7DF9A61EC44

5.11.1 Device Name Characteristic

Used to get/set the device name. Behaves identically to the GAP Service Device Name characteristic (0x2A00) which is hidden by iOS devices - so this characteristic allows iOS devices to access the name.

5.11.1.1 Permissions

Property	Required Auth Level
Read	None
Write	Admin

5.11.1.2 Value Structure

Up to 20 byte device name string, not null terminated.

5.11.2 Default Connection Parameters Characteristic

Used to get/set the default connection parameters for the device. If autoUpdate is set to 0, the parameters will just be used to populate the Slave Connection Interval Range ad structure in the scan response data.

The connection parameters control how often two devices in a connection will meet up to exchange data in a connection event. They control the balance between throughput and power savings.

- **Connection Interval –** The amount of time between two connection events. A shorter connection interval will result in higher throughput, lower latency, but will use more power, and a longer connection interval will result in lower throughput, higher latency and less power consumption.

- **Slave Latency –** Allows a slave to skip a set number of connection events. This allows the slave to save power if it has no data to send. This will not affect the latency of data sent from slave to master, but will result in increased latency from master to slave. The slave latency must not make the effective connection interval greater than 32s.

- **Supervision Timeout –** Amount of time allowed since the last successful connection event. If this timeout occurs the connection will be dropped.

- **Effective Connection Interval –** Amount of time between connection events, assuming the slave always skips [slave latency] connection events. It can be calculated with the following formula:

  *Effective Connection Interval = (Connection Interval) * (1+(Slave Latency))*

5.11.2.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.11.2.2 Value Structure

```
typedef struct
{
 uint16 intervalMin;
 uint16 intervalMax;
 uint16 latency;
 uint16 supervisionTimeout;
 uint16 autoUpdate;
} defConnParams_t;
```

5.11.2.2.1 **intervalMin**

Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module. DEFAULT = 72 (90ms)

5.11.2.2.2 **intervalMax**

Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module. DEFAULT = 88 (100ms)

5.11.2.2.3 **latency**

Integer value from 0 to 499 [connection intervals]. DEFAULT = 0

5.11.2.2.4 **supervisionTimeout**

Integer value from 10 to 3200 [10ms]. Must be greater than 2 * (Max_Conn_Interval * (1+Slave_Latency)). DEFAULT = 400 (4s)

5.11.2.2.5 **autoUpdate**

0 = If connected to as a slave, the module will accept any connection parameters. The parameters will just be used to populate the Slave Connection Interval Range ad structure in the scan response data.

1-65534 = If connected to as a slave and the connection parameters don't match the module's default connection parameters, the module will automatically request a connection parameter update after waiting 1-65534 ms. 5 seconds is recommended.

65535 = If connected to as a slave and the connection parameters don't match the module's default connection parameters, the module will automatically request a connection parameter update after the pairing process is complete and the link is encrypted.

DEFAULT = 0

5.11.3 Current Connection Parameters Characteristic

Used to get/set the current connection parameters of a connection. The value structure is different for reading and writing. Writing this characteristic will cause the slave device to request a connection parameter update. This should only be used if connection parameters cannot be changed through the API's of the central device, such as on iOS.

The connection parameters control how often two devices in a connection will meet up to exchange data in a connection event. They control the balance between throughput and power savings.

- **Connection Interval –** The amount of time between two connection events. A shorter connection interval will result in higher throughput, lower latency, but will use more power, and a longer connection interval will result in lower throughput, higher latency and less power consumption.
- **Slave Latency –** Allows a slave to skip a set number of connection events. This allows the slave to save power if it has no data to send. This will not affect the latency of data sent from slave to master, but will result in increased latency from master to slave. The slave latency must not make the effective connection interval greater than 32s.
- **Supervision Timeout –** Amount of time allowed since the last successful connection event. If this timeout occurs the connection will be dropped.
- **Effective Connection Interval –** Amount of time between connection events, assuming the slave always skips [slave latency] connection events. It can be calculated with the following formula:

  *Effective Connection Interval = (Connection Interval) * (1+(Slave Latency))*

5.11.3.1 Permissions

Property	Required Auth Level
Read	None
Write	User
Notify	User

5.11.3.2 Read Value Structure

```
typedef struct
{
 uint16 interval;
 uint16 latency;
 uint16 supervisiontimeout;
} connParamsRead_t;
```

5.11.3.2.1 **interval**

Integer value from 6 to 3200 [1.25ms].

5.11.3.2.2 **latency**

Integer value from 0 to 499 [connection intervals].

5.11.3.2.3 **supervisionTimeout**

Integer value from 10 to 3200 [10ms]. Must be greater than 2 * (Max_Conn_Interval * (1+Slave_Latency)).

5.11.3.3 Write Value Structure

```
typedef struct
{
 uint16 intervalMin;
 uint16 intervalMax;
 uint16 latency;
 uint16 supervisionTimeout;
} connParamsWrite_t;
```

5.11.3.3.1 **intervalMin**

Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module.

5.11.3.3.2 **intervalMax**

Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module.

5.11.3.3.3 **latency**

Integer value from 0 to 499 [connection intervals].

5.11.3.3.4 **supervisionTimeout**

Integer value from 10 to 3200 [10ms]. Must be greater than 2 * (Max_Conn_Interval * (1+Slave_Latency)).

5.11.4 Advertising Parameters Characteristic

Used to get/set the advertising parameters for the device. Can write all or part of the config structure – allowing just control to be written if needed. Control values should be written on their own without the rest of the value structure.

5.11.4.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.11.4.2 Value Structure

```
typedef struct
{
 uint8 control;
 adParamFlags_t flags;
 uint16 slowInterval;
 uint16 slowTimeout;
 uint16 slowPeriod;
 uint16 fastInterval;
 uint16 fastTimeout;
 uint16 fastPeriod;
 uint16 pairableInterval;
 uint8  pairableTimeout;
 uint16 connectedInterval;
 uint8  connectedDelay;
} adParams_t;
```

5.11.4.2.1 **control**

Used to temporarily control advertising. This value will be reset to 0 on connect. Control values should be written on their own without the rest of the value structure.

Value	Command Name	Description
0x00	None	None
0x01	Skip First Period	Skip the first advertising period after disconnect. Normally after a disconnect the device will immediately start a new advertising period, but the master can have it skip the first period to save battery . The advertising period will only be skipped if there are no alerts active and the disconnect was requested (the connection didn't timeout). Control must be set to this value at disconnect in order to take effect.
0x02	Ad First Period	Advertise the first advertising period after disconnect. If flags.alwaysSkipNextPeriod is set the device will always skip the first advertising period after the remote device disconnects, unless there is an active alert. This command can be used to temporarily disable this and have the device advertise for the first period after disconnecting. Control must be set to this value at disconnect in order to take effect.
0x02	Start Ad	Start advertising while connected.
0x03	Stop Ad	Stop advertising while connected.

5.11.4.2.2 **flags**

```
typedef struct
{
  uint8 enableChannel37:1;
  uint8 enableChannel38:1;
  uint8 enableChannel39:1;
  uint8 adOnlyOnAlert:1;
  uint8 adWhileConnected:1;
  uint8 alwaysSkipFirstPeriod:1;
  uint8 ledOnConnect:1;
} adParamFlags_t;
```

enableChannel37
If enabled will advertise on channel 37. At least one channel must be enabled, all 3 are recommended. DEFAULT = 1

enableChannel38
If enabled will advertise on channel 37. At least one channel must be enabled, all 3 are recommended. DEFAULT = 1

enableChannel39
If enabled will advertise on channel 37. At least one channel must be enabled, all 3 are recommended. DEFAULT = 1

adOnlyOnAlert
If enabled will only advertise if there is an alert or the device is pairable. DEFAULT = 0

adWhileConnected
If enabled will start advertising at the connectedInterval 1 second after connecting. DEFAULT = 0

alwaysSkipFirstPeriod
If enabled will always skip the first advertising period after disconnect. The advertising period will only be skipped if there are no alerts active and the disconnect was requested (the connection didn't timeout). DEFAULT = 0

ledOnConnect
If enabled will turn on the blue led on connect and off on is disconnect. DEFAULT = 0

5.11.4.2.3 slowInterval

Integer value from 32-6400 [.625ms] (20 – 10240 ms). This value sets the unconnected advertising interval for the device when in the slow advertising mode. DEFAULT = 1636 (1022.5ms)

5.11.4.2.4 slowTimeout

Integer value from 0-65535s, 0 = always advertise. This value sets how long the device will advertise for each slowPeriod. DEFAULT = 0 (Always advertise)

5.11.4.2.5 slowPeriod

Integer value from 1-665535s, must be greater than or equal to slowTimeout. This value sets how often the device will start advertising when in the slow advertising mode. DEFAULT = 300 (5 min)

5.11.4.2.6 fastInterval

Integer value from 32-6400 [.625ms] (20 – 10240 ms). This value sets the unconnected advertising interval for the device when in the fast advertising mode. DEFAULT = 160 (100ms)

5.11.4.2.7 fastTimeout

Integer value from 0-65535s, 0 = always advertise. This value sets how long the device will advertise for each fastPeriod. DEFAULT = 3s

5.11.4.2.8 fastPeriod

Integer value from 0-65535s, must be greater than or equal to fastTimeout unless 0, 0 = only advertise fast for one timeout on alert. This value sets how often the device will start advertising when in the fast advertising mode. DEFAULT = 0

5.11.4.2.9 pairableInterval

Integer value from 32-6400 [.625ms] (20 – 10240 ms). This value sets the advertising interval for the device when in pairing mode. DEFAULT = 160 (100ms)

5.11.4.2.10 pairableTimeout

Integer value from 1-255s. This value sets how long the device will advertise pairable for. DEFAULT = 30s

5.11.4.2.11 connectedInterval

Integer value from 160-16384 [.625ms] (100 – 10240 ms). This value sets the advertising interval used when the device is connected. DEFAULT = 3272 (2045ms)

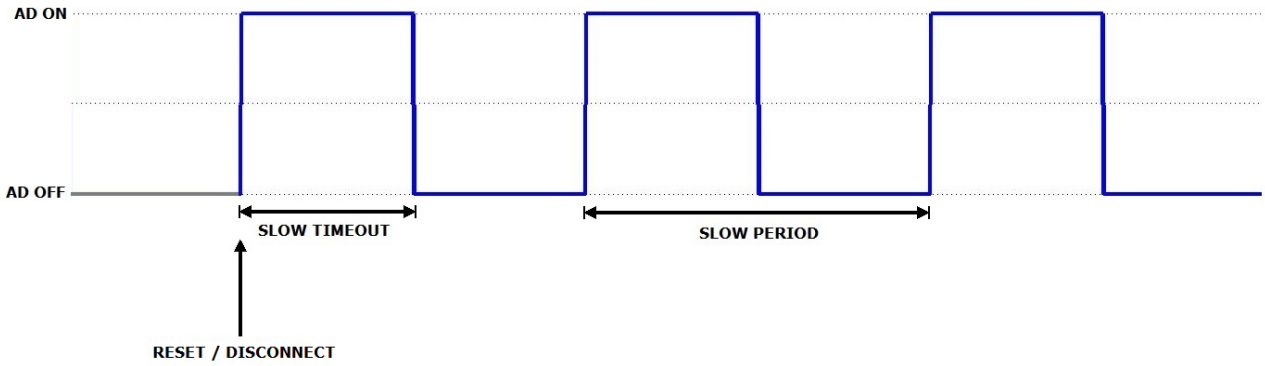
5.11.4.2.12 connectedDelay

Integer value from 1-255s. How long after connection establishment to wait before advertising connectable. DEFAULT = 1

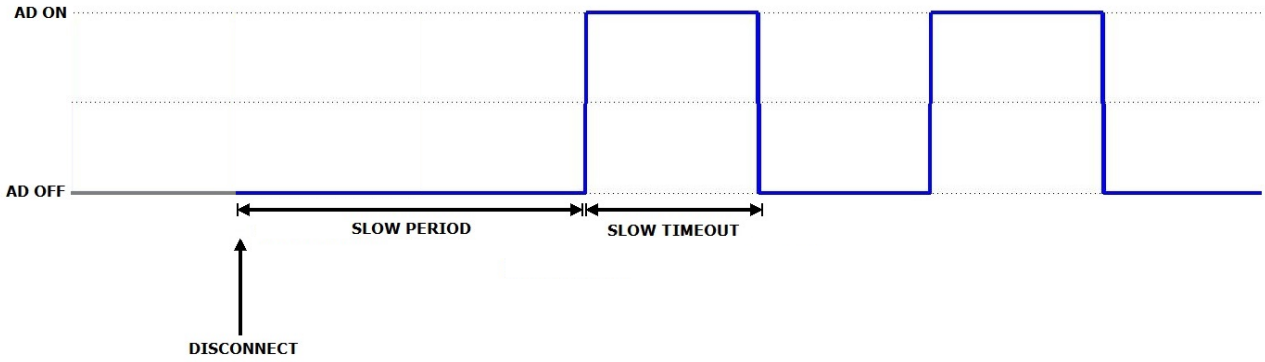
5.11.4.3 Timing Diagrams

5.11.4.3.1 Slow Advertising

After a reset or disconnect if no alerts are set the device will go into slow advertising mode.

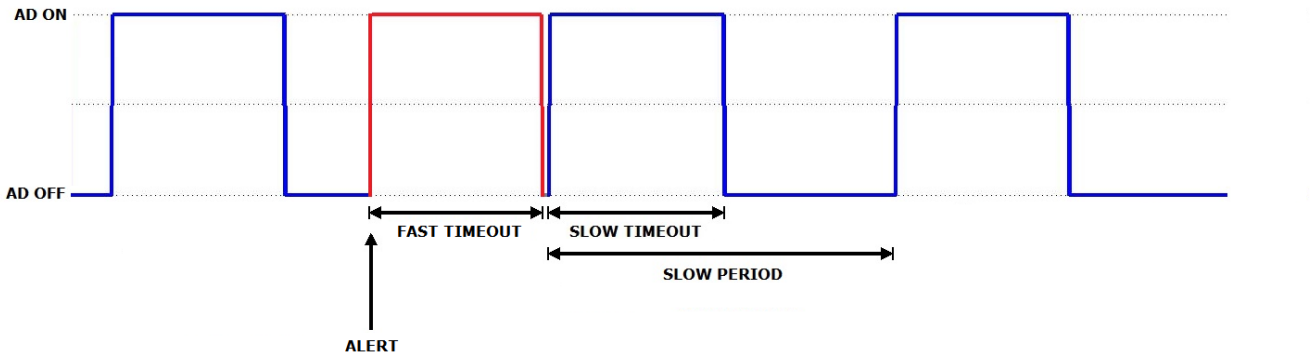


If flags.alwaysSkipFirstPeriod or control is set to 1, the first slowPeriod after a disconnect will be skipped.

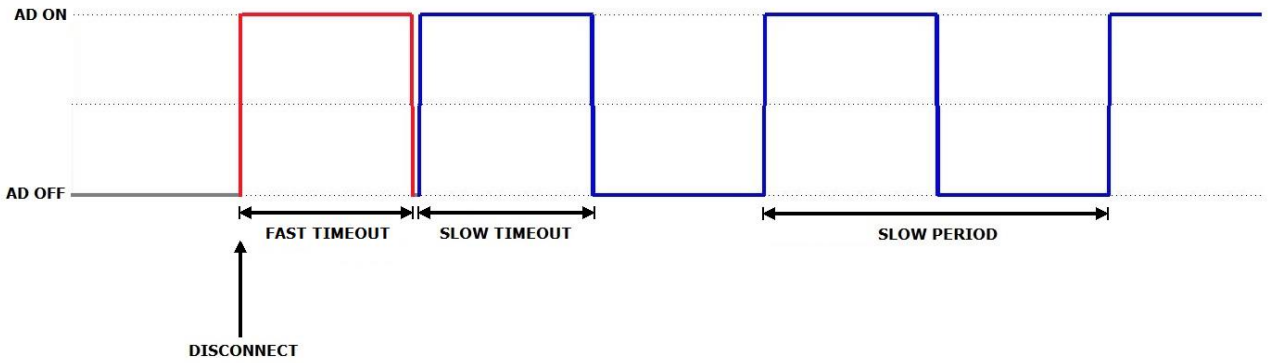


5.11.4.3.2 Fast Advertising With Fast Period of 0

If an alert occurs the device will switch to fast advertising mode and advertise at fastInterval for fastTimeout, then switch back to slow advertising mode.

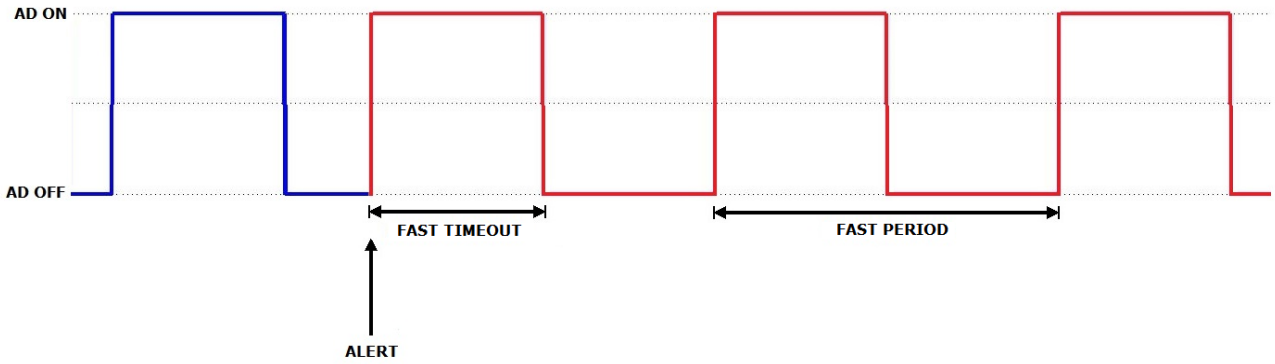


If an alert is still set after a disconnect, the device will immediately advertise in fast advertising mode at fastInterval for fastTimeout, then switch back to slow advertising mode.

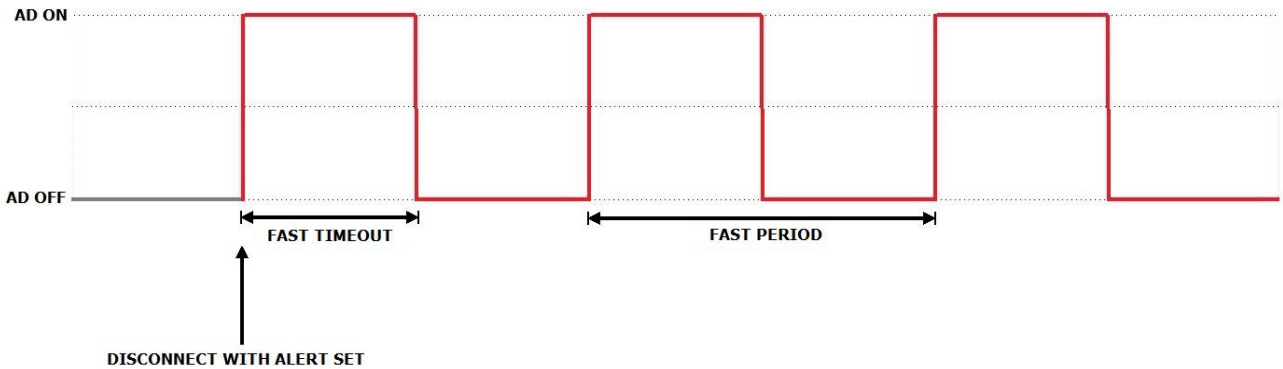


5.11.4.3.3 Fast Advertising With Non-Zero Fast Period

If an alert occurs the device will switch to fast advertising mode and advertise at fastInterval for fastTimeout every fastPeriod. It won't switch back to slow advertising mode until the alert is cleared.

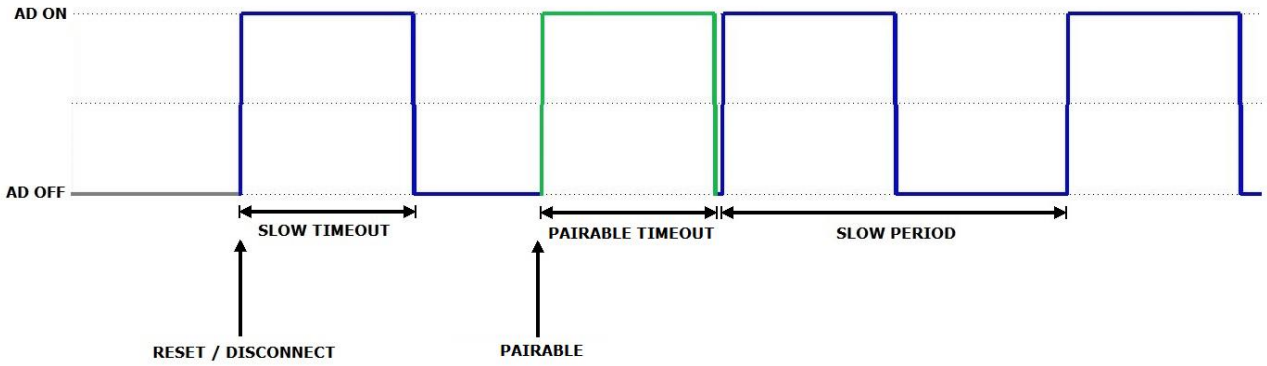


If an alert is still set after a disconnect, the device will immediately advertise in fast advertising mode at fastInterval for fastTimeout every fastPeriod. It won't switch back to slow advertising mode until the alert is cleared.



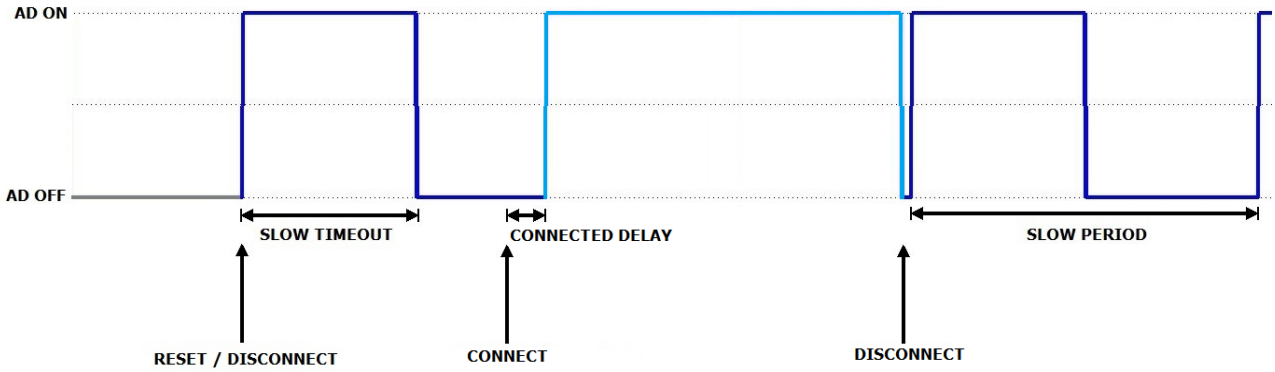
5.11.4.3.4 Pairable Advertising

If the device enters pairing mode it will advertise at pairingInterval for pairableTimeout, then switch back to the mode it was in previously.



5.11.4.3.5 Connected Advertising

If the device enters pairing mode it will start advertising at connectedInterval 1s after connecting and continue to advertise until disconnecting.



5.11.4.4 Effects on Sensor Behavior

If a sensor is enabled, but alerts aren't enabled on the sensor, then the sensor will be powered off when advertising is off to conserve power. It will then be powered on and a reading taken immediately before advertising turns back on.

If alerts are enabled the sensor won't be powered off when advertising is off so alerts can still be triggered. Immediately before advertising is turned back on a reading will be taken to make sure the data is up to date before advertising begins.

5.11.5 RF Power Levels Characteristic

Used to get/set the transmit power level and receive sensitivity.

5.11.5.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.11.5.2 Value Structure

```
typedef struct
{
 int8 txPowerLevel;
 uint8 rxSensitivity;
} powerLevels_t;
```

5.11.5.2.1 txPowerLevel

Value	TX Power Level
0	0 dBm
-6	-6 dBm
-23	-23 dBm

5.11.5.2.2 rxSensitivity

Value	RX Sensitivity
1	-96 dBm

5.11.6 Disconnect Characteristic

Writing any value will cause the remote device to disconnect. This allows iOS devices to disconnect immediately vs holding the connection for a few seconds if disconnecting through iOS.

5.11.6.1 Permissions

Property	Required Auth Level
Write	None

5.11.7 Public Device Address Characteristic

Allows a central device to read the device's public Bluetooth Device Address, since this can't be done through iOS.

5.11.7.1 Permissions

Property	Required Auth Level
Read	None

5.11.7.2 Value Structure

6 byte Public *Bluetooth* Device Address

5.11.8 Config Counter Characteristic

Allows the connected device to read the config counter, which is incremented each time the device's configuration is changed. This values matches the value in the Static Data section of the manufacturer specific data.

5.11.8.1 Permissions

Property	Required Auth Level
Read	User

5.11.8.2 Read Value Structure

1 Byte, 0-255, 0 only on a reset.

5.11.9 Blink LED Characteristic

This characteristic is used to blink the LED(s). Writing only the leds byte will blink the specified LED(s) for 10 seconds, writing the leds byte and numBlinks byte will blink the specified LED(s) for the number of blinks at a 20% duty cycle and 500ms period (2 blinks per second), writing the full blinkLEDs structure will give full control of LED(s) blinking.

5.11.9.1 Permissions

Property	Required Auth Level
Write	User

5.11.9.2 Write Value Structure

```
typedef struct
{
  uint8 leds;          // bit mask value of LED(s) to be blinked
  uint8 numBlinks;     // number of blinks
  uint8 percent;       // the percentage in each period where the LED(s) will be on
  uint16 period;       // length of each cycle in milliseconds
} blinkLEDs_t;
```

leds
Bit mask value of LED(s) to be blinked, OR together to blink multiple LEDs at same time.

Value	LED
0x01	Blue LED
0x02	Green LED

numBlinks
The number of times the LED(s) will blink.

percent
The percentage in each period where the LED(s) will be on.

period
Length of each cycle in milliseconds.

5.11.10 Factory Value Characteristic

This characteristic indicates if a device has been factory reset. After a factory reset it will be set to 0xFF. The characteristic value won't change from 0xFF on its own, a master must write the characteristic value to something other than 0xFF during the configuration process.

5.11.10.1 Permissions

Property	Required Auth Level
Read	User
Write	Admin

5.11.10.2 Value Structure

1 Byte, 0-255, 255 only on a reset.