Application Report
SWRA475A–January 2015–Revised October 2016

Bluetooth® low energy Beacons

Joakim Lindh

ABSTRACT

This application report presents the concept of beacons by using Bluetooth low energy technology. The important parameters when designing beacon solutions are elaborated on a detailed level throughout the document. With the use of the TI Bluetooth low energy-Stack, beacons can be done in a simple and intuitive way.

Contents

List of Figures

List of Tables

Trademarks

1 What is a Beacon?

A beacon in wireless technology is the concept of broadcasting small pieces of information. The information may be anything, ranging from ambient data (temperature, air pressure, humidity, and so forth) to micro-location data (asset tracking, retail, and so forth) or orientation data (acceleration, rotation, and so forth).

The transmitted data is typically static but can also be dynamic and change over time. With the use of Bluetooth low energy, beacons can be designed to run for years on a single coin cell battery. This application report introduces the concept of beacons and how to get started with implementing a beacon solution. Naming conventions throughout this document can be summarized as Beacons that broadcast information by using advertisements with Bluetooth low energy technology that could be branded as Bluetooth low energy.

2 Bluetooth low energy and Bluetooth Smart

A Bluetooth low energy device can operate in four different device roles. Depending on the role, the devices behave differently. The first two roles are connection-based:

- A Peripheral device is an advertiser that is connectable and can operate as a slave in a connection. This could, for example, be a health thermometer or a heart rate sensor.
- A Central device scans for advertisers and can initiate connections. It operates as a master in one or more connections. Good examples are Smartphones and computers.

This means that the device roles used for established connections are the Peripheral and the Central roles. The other two device roles are used for one-directional communication:

- A Broadcaster is a non-connectable advertiser, for example, a temperature sensor that broadcasts the current temperature, or an electronic tag for asset tracking.
- An Observer scans for advertisements, but cannot initiate connections. This could be a remote display that receives the temperature data and presents it, or tracking the electronic tag.

The two obvious device roles for beacon applications are Peripheral and Broadcaster. Both of them send the same type of advertisements with the exception of one specific flag that indicates if it is connectable or non-connectable. A Peripheral device that implements a GATT Server (GATT is an architecture for how data is stored and exchanged between two or more devices) can be branded as a Bluetooth low energy device. So a Bluetooth low energy branding indicates that the device is a connectable Peripheral device that has data, which could be interacted with.

A Bluetooth low energy solution is ideal for beacons because it is low power and the eco-system is already deployed in the majority of smartphones or other Bluetooth low energy enabled devices on the market. The low-power consumption is achieved by keeping the transmission time as short as possible and allowing the device to go into sleep mode between the transmissions.

2.1 Non-Connectable Beacons

The non-connectable beacon is a Bluetooth low energy device in broadcasting mode. It simply transmits information that is stored internally. Because the non-connectable broadcasting does not activate any receiving capabilities, it achieves the lowest possible power consumption by simply waking up, transmit data and going back to sleep. This comes with the drawback of dynamic data being restricted to what is only known to the device, or data being available through external input from example serial protocols (universal asynchronous receiver/transmitter (UART), serial peripheral interface (SPI), universal serial bus (USB), and so forth).

2.2 Connectable Beacons

The connectable beacon is a Bluetooth low energy device in peripheral mode, which means that it can not only transmit, but receive as well. This allows a central device (for example, a smartphone) to connect and interact with services implemented on the beacon device. Services provide one or more characteristics that could be modified by a peer device. One example of these characteristics could be a string of data that represents the broadcasted information. This way, it is possible to have a configurable Beacon that is easily updated over the air.

2.3 Data Packet

The transmitted data from a Bluetooth low energy device is formatted according to the Bluetooth Core Specification and is comprised of the parts shown in Figure 1.

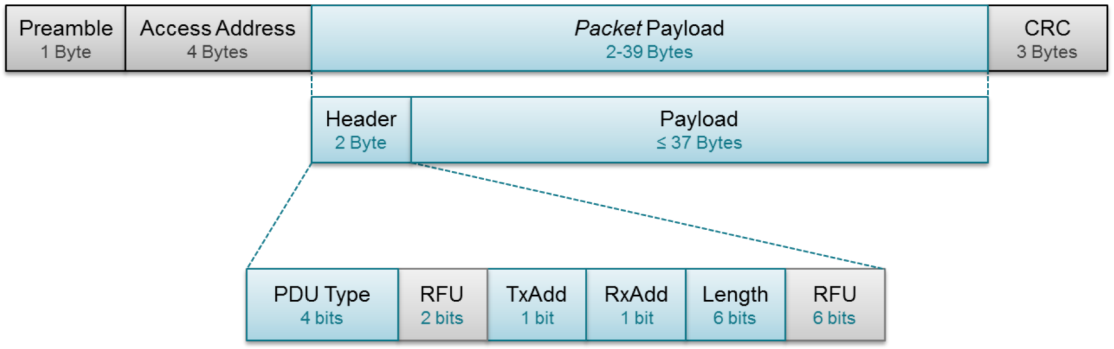

Figure 1. Bluetooth low energy Data Packet

The Preamble is a 1 byte value used for synchronization and timing estimation at the receiver. It will always be 0xAA for broadcasted packets. The Access Address is also fixed for broadcasted packets, set to 0x8E89BED6. The packet payload consists of a header and payload. The header describes the packet type and the PDU Type defines the purpose of the device. For broadcasting applications, there are three different PDU Types, as shown in Table 1. ADV_IND and ADV_NONCONN_IND have been described previously (as connectable and non-connectable) while ADV_SCAN_IND is simply a non-connectable broadcaster that can provide additional information by scan responses.

Table 1. Advertising PDU Types for Broadcasting Data

PDU Type	Packet Name	Description
0000	ADV_IND	Connectable undirected advertising event
0010	ADV_NONCONN_IND	Non-connectable undirected advertising event
0110	ADV_SCAN_IND	Scannable undirected advertising event

The TxAdd bit indicates whether the advertiser's address (contained in the Payload) is public (TxAdd = 0) or random (TxAdd = 1). RxAdd is reserved for other types of packets not covered in this application note, as they do not apply to beacons.

The final part of the transmitted packet is the Cyclic Redundancy Check (CRC). CRC is an error-detecting code used to validate the packet for unwanted alterations. It ensures data integrity for all transmitted packets over the air.

The Payload of the packet includes the advertiser's address along with the user defined advertised data as shown in Figure 2. These fields represent the beacon's broadcasted address and data.

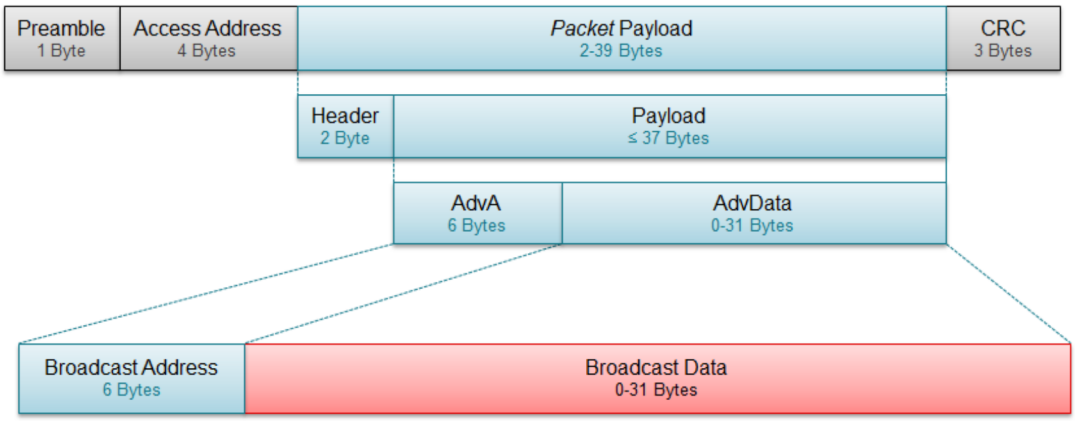

Figure 2. Bluetooth low energy Broadcast Data

2.4 Device Address

The broadcast address can be either public or random. A public address [1] (Vol.6.C.1.3 page 2500) is an IEEE 802-2001 standard and uses an Organizationally Unique Identifier (OUI) obtained from the IEEE Registration Authority. Texas Instruments provides IEEE addresses for all Bluetooth Smart devices. Random addresses can be directly generated by the beacon and can be of three different types: static, non-resolvable private and resolvable private. Static addresses are not allowed to be changed unless the device power cycles. A private address can change over time and a resolvable address can be used to derive the true address. A non-resolvable address can also change over time, which is the differentiation compared to a static address. The random address is a privacy feature that prevents tracking of a specific device. There are specific rules when generating random addresses and the details can be found in the Core Specification [1] (Vol3.C.10.8 page 2020).

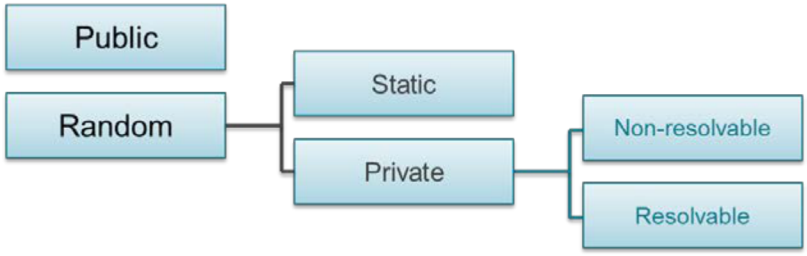

Figure 3. Beacon Address Types

The broadcasted data can be formatted according to Bluetooth SIG specified data formats, with some examples shown in Table 2 [2], [3]. For the purpose of this application report, the focus is on the Flags and Manufacturer-Specific Data.

Table 2. Advertisement Data Types

AD Data Type	Data Type Value	Description
Flags	0x01	Device discovery capabilities
Service UUID	0x02 - 0x07	Device GATT services
Local Name	0x08 - 0x09	Device name
TX Power Level	0x0A	Device output power
Manufacturer Specific Data	0xFF	User defined

2.4.1 Flags

The first three bytes of the broadcasted data defines the capabilities of the device. It is a requirement from the Core Specification [1] (Vol 3.C.13.1.1 page 2029) and the format is defined as shown in Table 3.

Table 3. Advertisement Data Type, Flags

Byte	Bit	Flag/Value	Description
0		0x02	Length of this data
1		0x01	GAP AD Type Flags
2	0	LE Limited Discoverable Mode	180 s advertising
	1	LE General Discoverable Mode	Indefinite advertising time
	2	BR/EDR Not Supported	
	3	Simultaneous LE and BR/EDR (Controller)	
	4	Simultaneous LE and BR/EDR (Host)	
	5-7		Reserved

Discovery mode flags are bit masked and the various settings are presented in Table 3. If no bit flags are to be set, the Flags Data type can be omitted [2]. It would for example not be required for a non-connectable advertisement packet (ADV_NONCONN_IND).

2.4.2 Manufacturer Specific Data

When using Manufacturer-Specific Data, the 0xFF flag is used to indicate so. The first two bytes of the data itself should be a company identifier code.

Table 4. Advertisement Data Types, Manufacturer Specific Data Format

Byte	Value	Description
0	0x03-0x1F	Length of this data
1	0xFF	Manufacturer-Specific Data Flag
2	0x0D	Company ID
3	0x00	(Example, 0x000D – Texas Instruments)
4 - 31	-	User Defined Data (optional)

The Bluetooth low energy packet format allows a device to broadcast 25 Bytes of Manufacturer-Specific Data if the advertisement is of the type Connectable undirected advertising (ADV_IND) or Scannable undirected advertising event (ADV_SCAN_IND) as Discoverable Mode flags are required. For non-connectable undirected advertising (ADV_NONCONN_IND) maximum length of the Manufacturer-Specific Data is 28 bytes. The Manufacturer-Specific Data can contain any user-defined information.

The broadcasted data can also be formatted in a standardized way. At the time of writing this application report there are a couple of standards, such as iBeacon from Apple and AltBeacon, from Radius Networks. iBeacon is protected by the MFi license and is interoperable with all iOS devices. AltBeacon is an open standard and the specification can be downloaded at http://altbeacon.org/. This application note describes how to use the TI Bluetooth low energy stack for an iBeacon. Eddystone, from Google, is an open source beacon standard, and is covered in an application note on the Bluetooth low energy wiki page.

2.5 Broadcast Interval

A beacon achieves low-power consumption by residing in sleep most of the operating time, only waking up briefly to broadcast data. The time between these broadcast events are referred to as advertising interval, which is illustrated in Figure 4. For non-connectable beacons, the interval cannot be smaller than 100 ms. For connectable beacons, it cannot be smaller than 20 ms. To this interval, a 0-10 ms pseudo-random delay is added to ensure that beacons can coexist, even if they might start broadcasting at the same time.

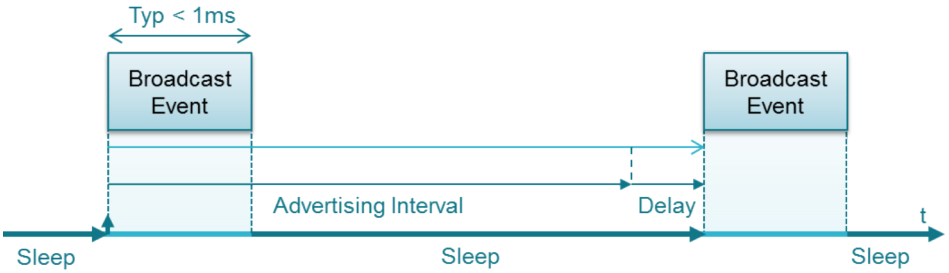

Figure 4. Advertising Interval

The advertising interval is chosen based on requirements of power consumption vs latency. Higher interval allows more time in sleep mode but increases the time it might take for an observer to obtain a broadcasted packet.

An observer typically uses a scanning with a duty cycles less than 100% to conserve energy or allow other wireless protocols a time slot. One good example is Smartphones, which in most cases has a one chip solution for Bluetooth and Wi-Fi®. If an audio headset is connected via Bluetooth and Wi-Fi has a connection to a local access point, Bluetooth low energy scanning will probably have a very low-duty cycle. The time slots are essentially split between the 2.4 GHz protocols on the device.

An observer can be either scanning in passive or active mode. If active mode is used and a beacon supports it, a Scan Request will be sent, which the beacon must respond to with a Scan Response. The request itself is an empty packet (no data allowed), whereas, the response is typically static information such as name or model of device. It is fully proprietary so it could also be calibration data for sensors or any other useful information. When an observer is scanning in passive mode, it will not send a Scan Request.

2.6 Power

A beacon can be powered using several methods. There are three different main approaches:

- DC power supply (USB, Mains, and so forth)
- Batteries (CR2032, AAA, Lithium, and so forth)
- Energy harvesting (Solar, kinetics, and so forth)

Typically, the primary choice is batteries that allow small and low-cost product designs while still maintaining a lifetime suitable for most applications. It is also possible to use re-chargeable batteries and in some applications together with wireless charging. The choice of battery type is important because some batteries might be sensitive to high peak currents. The amount of capacity needed is based on how often broadcasting is necessary, and if any further processing is required (sensor interaction, algorithms, and so forth). Sensor interaction typically involves serial communication using UART, SPI or inter-integrated circuit (I2C) that adds a power consumption overhead, which may be greater than the Bluetooth low energy protocol alone.

Figure 5. Power Options

If the design is powered by DC power supply, the assumption would be that the power consumption is no critical parameter. If it is still critical, the approach would be the same as for a battery powered design.

Energy harvesting is being introduced more into low-power wireless solutions and a beacon can be powered by energy harvested sources. Mechanical pressure, fresh fruit and solar power are commonly known sources and even indoor light [4] can be used to power a beacon.

2.7 Range

In Radio Frequency (RF) theory, the range is given by a number of factors:

- Sensitivity of the radio receiver
- Output power of the radio transmitter
- Environment and coexistence
- Antenna performance

A beacon application may require ranges from centimeters up several hundred meters. The maximum output power allowed by the core specification is 10 dBm, which should get distances up to several hundred meters if all above mentioned factors are considered. The mathematical and physical derivation of the theoretical range is not covered by this application report.

2.8 Coexistence

The open 2.4 GHz ISM frequency band that is used by Bluetooth low energy is filled with many other wireless protocols, such as Wi-Fi, as well as potential interference from home appliances, such as microwave ovens. These radio activities may interfere with the Bluetooth low energy activity. The broadcasting of advertisements occurs on three different channels sequentially as demonstrated in Figure 6.

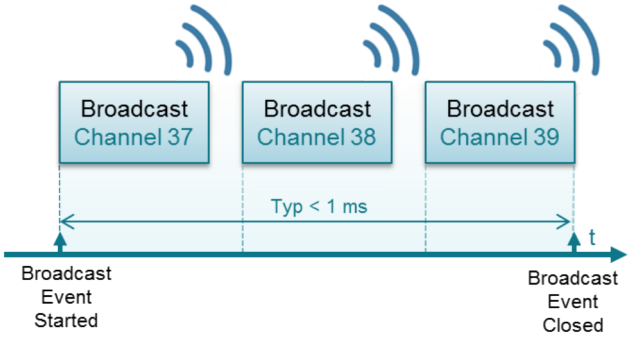

Figure 6. Broadcasting Advertisements

The channels 37, 38 and 39 have been chosen to not collide with the three most commonly used Wi-Fi channels; 1, 6 and 11, as shown in Figure 7.

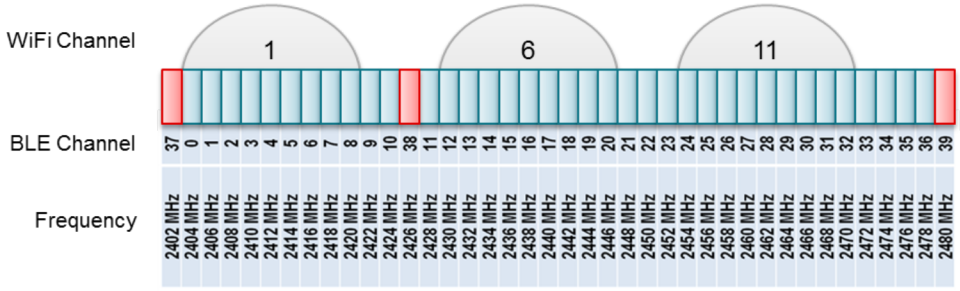

Figure 7. Frequency Band and Channels, Bluetooth low energy

However, Wi-Fi has significantly higher output power, up to 23 dBm compared to maximum allowed 10 dBm for Bluetooth low energy. This means that placing a beacon very close to a Wi-Fi source will probably distort the transmitted data as spurious emissions on side channels of the Wi-Fi unit will almost always occur on a non-ideal RF product, as show in Figure 8. Figure 8 shows the three broadcast output power peaks on advertising channels 37, 38 and 39 versus the Wi-Fi output peak for streaming data at 24 Mbit/s.

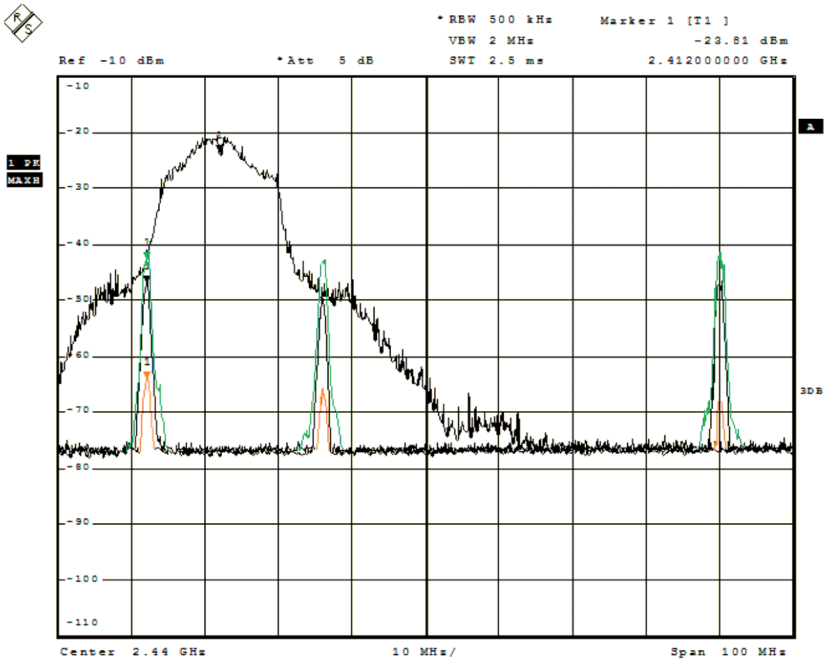

Figure 8. Spectrum for Wi-Fi Channel 1 versus Beacon Advertisements Channels

If Wi-Fi runs on other channels that overlap with the broadcasting channels, it is dependent on the received signal strength (RSSI) at the receiving device. Even though the advertising channels for Bluetooth low energy has been strategically placed in the 2.4 GHz ISM band to not interfere with the most common Wi-Fi channels, Figure 8 shows that interference and coexistence issues with these particular Wi-Fi channels could be present anyway. However, it should be noted that the frequency spectrum was measured with a beacon residing directly on top of a Wi-Fi device.

Depending on the Beacon application, there are different requirements on coexistence. Even a small amount of active beacons tend to interfere with each other and cause packets to be lost. As already seen, Wi-Fi has in general higher TX-power as well as wider occupancy in the 2.4 GHz ISM band than Bluetooth low energy devices.

3 Designing a Bluetooth low energy Beacon

As with any design, there are always parameters that can be optimized to a certain cost, by weighting the different trade-offs. One example is the broadcasting interval. Choosing a lower interval increases the probability to be observed by a peer device quicker, although the power consumption is also increased.

3.1 Development Kits

Getting started with designing a beacon first involves the decision of which development kit to use. The CC2650 LaunchPad, which is shown in Figure 9, is the recommended platform for Bluetooth low energy beacons. The sample projects mentioned in this application note run on the CC2650 LaunchPad. More information about this and other development kits can be found at ti.com/ble.

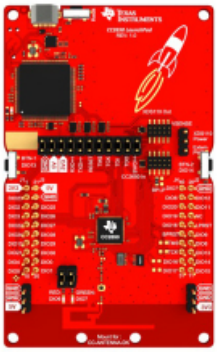

Figure 9. CC2650 LaunchPad

The development kits can be purchased online at TI store [5].

3.2 Creating a Beacon Application With TI Bluetooth low energy-Stack

The BLE-stack available from Texas Instrument for the CC26xx Wireless MCUs provides an easy and reliable implementation of connectable and non-connectable beacons. There are sample applications that can be used as templates when designing a beacon, which are described in Table 5. It is assumed that you are familiar with the IAR Embedded Workbench and the BLE-stack.

Table 5. Beacon Software Examples for CC26xx

Example Project	GAP Role	Type	Device Support
SimpleBLEPeripheral	Peripheral	Connectable	CC2640, CC2650
SimpleBLEBroadcaster	Broadcaster	Non-connectable	CC2640, CC2650

SimpleBLEPeripheral is thoroughly described in the software Development Guide [8] and is generally the best starting point if implementing a connectable beacon. The SimpleBLEBroadcaster is a simplified version of SimpleBLEPeripheral, which only supports non-connectable beacons. The API to enable beacon functionality is the same for these projects so the following code examples apply for both, although SimpleBLEBroadcaster (BLEv2.2) is used as the referenced example project.

The application is implemented in SimpleBLEBroadcaster.c where the broadcasting data is defined as advertData.

```
static uint8 advertData[] =
{
  // Flags; this sets the device to use limited discoverable
  // mode (advertises for 30 seconds at a time) instead of general
  // discoverable mode (advertises indefinitely)
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,

#ifndef BEACON_FEATURE

  // three-byte broadcast of the data "1 2 3"
  0x04,   // length of this data including the data type byte
  GAP_ADTYPE_MANUFACTURER_SPECIFIC, // manufacturer specific adv data type
  1,
  2,
  3

#else

  // 25 byte beacon advertisement data
  // Preamble: Company ID - 0x000D for TI, refer to https://www.bluetooth.org/en-
us/specification/assigned-numbers/company-identifiers
  // Data type: Beacon (0x02)
  // Data length: 0x15
  // UUID: 00000000-0000-0000-0000-000000000000 (null beacon)
  // Major: 1 (0x0001)
  // Minor: 1 (0x0001)
  // Measured Power: -59 (0xc5)
  0x1A, // length of this data including the data type byte
  GAP_ADTYPE_MANUFACTURER_SPECIFIC, // manufacturer specific adv data type
  0x0D, // Company ID - Fixed
  0x00, // Company ID - Fixed
  0x02, // Data Type - Fixed
  0x15, // Data Length - Fixed
  0x00, // UUID - Variable based on different use cases/applications
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // UUID
  0x00, // Major
  0x01, // Major
  0x00, // Minor
  0x01, // Minor
  0xc5  // Power - The 2's complement of the calibrated Tx Power

#endif // !BEACON_FEATURE
};
```

The default advertised data includes the mandatory Flags followed by 3 bytes of Manufacture-Specific Data (numbers 1, 2 and 3). The Manufacture-Specific Data can be modified to be something else, although note to update the length of the data as well, if necessary.

The configured variables are then committed to the GAP layer, which sets up the Bluetooth low energy stack. Note that the *advertEnable* is not instantaneously triggered, especially not during the initiation of the application (simpleBLEBroadcaster_Init). The advertise will start when the protocol stack runs, at a later instance.

```
GAPRole_SetParameter( GAPROLE_ADVERT_ENABLED,sizeof(uint8), &advertEnable );
GAPRole_SetParameter( GAPROLE_ADVERT_DATA, sizeof(advertData), advertData );
GAPRole_SetParameter( GAPROLE_ADV_EVENT_TYPE, sizeof(uint8), &advType );
```

The advertising interval is default set to 100 ms although it can be increased up to 10.24 seconds, which is the maximum allowed by the core specification. If longer intervals are needed, it is possible to manually enable and disable the advertisement with the use of an OSAL timer, as an example.

```
// Advertising interval (units of 625us, 160=100ms)
#define DEFAULT_ADVERTISING_INTERVAL          160
```

To ensure discovery of broadcasts, there is a general rule that the advertisement Interval + 10 should be less than scan window of the observer device. This means that the beacon must be designed with the peer device capabilities in mind. Otherwise, it is possible that it will take a very long time to receive broadcasted packets. This implies that a lower broadcast interval will allow broadcasted data to be observed more quickly, although it requires more power due to the more frequent wake ups. The interval is setup with following API:

```
uint16_t advInt = DEFAULT_ADVERTISING_INTERVAL;

GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MIN, advInt );
GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MAX, advInt );
GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MIN, advInt );
GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MAX, advInt );
```

For more information regarding application architecture and API description, see the software Development Guide [6].

By using the TI Packet Sniffer [7], the broadcasted data can be verified (as shown in Figure 10). The illustration shows a packet on channel 37 (0x25) that is connectable (ADV_IND). The AdvA value is the IEEE address and advData includes Flags (0x01) and Manufacturer-Specific Data (0xFF).

Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
			Type	TxAdd	RxAdd	PDU-Length					
0x25	0x8E89BED6	ADV_DISCOVER_IND	6	0	0	14	0x90D7EBB1E02B	02 01 04 04 FF 01 02 03	0xA7C13C	-76	OK

Figure 10. Packet Sniffer v2.18.1, SimpleBLEBroadcaster Packet Over the Air

4 iBeacon Implementation

Apple has defined the iBeacon technology, available under an MFi license, for use with iOS devices. This is one of multiple protocols that have been defined by vendors to format how data is broadcasted by beacons. This section describes how to implement an example device that uses iBeacon technology, using the TI BLE-Stack V.2.2 SDK on the SimpleLink CC2640 Bluetooth low energy wireless MCU.

The necessary files for the beacon project can be found at the TI GitHub repository ble_examples.

4.1 Overview and Prerequisites

Prior to following the examples described in this section, one should have a detailed understanding of the TI BLE-Stack SDK as described in the SW Developer's Guide (SWRU393) [6], the previous sections of this application note, and the iBeacon Artwork and Specification document [8].

This sample application requires TI BLE-Stack V.2.2. Either IAR Embedded Workbench for ARM 7.50.3 or Code Composer Studio 6.1 IDEs can be used to build the project. The application runs on the CC2650 LaunchPad reference platform.

4.2 Design and Implementation

This project was created by modifying the SimplePeripheral project. It currently only supports non-connectable advertisements, but the SimplePeripheral project was used instead of the SimpleBroadcaster project to be consistent with the SimpleEddystoneBeacon project (described in a separate application note).

The main modification to the SimplePeripheral project is to the advertisement data. According to the iBeacon specifications, the data is as follows:

```
static beaconAdvData_t beaconAdv =
{
  // Flags; this sets the device to use general discoverable mode
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  0x06,

  0x1A,   // length of this data including the data type byte
  0xFF,   // type
  0x4C,   // company ID[0]
  0x00,   // company ID[1]
  0x02,   // beacon type[0]
  0x15,   // beacon type[1]
  0xA3,   // UUID LSB
  0x22,
  0x37,
  0xE7,
  0x3E,
  0xC0,
  0xC5,
  0x84,
  0x86,
  0x4B,
  0xB9,
  0x99,
  0xF9,
  0x82,
  0x03,
  0xF7,  // UUID MSB
  0x00,  //major[0] (major and minor fields can be set as desired)
  0x00,  //major[1]
  0x00,  //minor[0]
  0x00,  //minor[1]
  0x00   //measured power (can be set later)
};
```

The company ID identifies the beacon as an Apple product, and the beacon type identifies the device as a proximity beacon. Any other valid UUID could be used to replace the one that is currently there, as long as it is inserted in the order indicated. The major and minor fields are currently unset, but they can be set as desired to further identify the device. Lastly, the specifications describe a procedure for measuring the power.

Additionally, the advertisement interval is set by specifying minimum and maximum intervals for limited and general discoverable modes. Typically, all four parameters are set to the same value to get the desired interval as follows:

```
GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MAX, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MAX, advInt);
```

The specifications state that the beacon must broadcast the entire advertising packet in all three advertising frequencies, and they must use a 100-ms advertising interval.

4.3 Testing

The TI Packet Sniffer was used to test the device. Figure 11 shows an advertising packet as it was discovered by the sniffer. It can be seen that the advertising data reflects the requirements of the iBeacon specifications, as does the advertising PDU type.

Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData
			Type	TxAdd	RxAdd	PDU-Length		
0x25	0x8E89BED6	ADV_NON_CONN_IND	2	0	0	36	0xB0B448CF6A03	02 01 06 1A FF 4C 00 02 15 A3 22 37 E7 3E C0 C5 84 86 4B B9 99 F9 82 03 F7 00 00 00 00 00

Figure 11. Sniffer Packet Showing the Advertising Data of the Beacon Device

5 Proprietary Implementation

A sample application has been created using a proprietary beacon advertising format. Similar to the project that uses iBeacon technology, this application demonstrates a more generic implementation of a beacon.

5.1 Overview and Prerequisites

This project has the same requirements as the beacon project, which is described in Section 4. Instead of the iBeacon specifications, a generic example format was used, but all of the hardware and IDEs used remain the same.

The necessary files for the project can be found at the TI GitHub repository ble_examples.

5.2 Design and Implementation

This project was created by modifying the SimplePeripheral project. It currently only supports non-connectable advertisements, but the SimplePeripheral project was used instead of the SimpleBroadcaster project to be consistent with the SimpleEddystoneBeacon project (described in a separate application note) and the beacon project.

As with the other beacon projects, the main modification to the SimplePeripheral project is to the advertisement data. The data is set up as follows:

```
static propBeaconAdvData_t propBeaconAdv =
{
  // Flags; this sets the device to use general discoverable mode
  0x02,   // length of this data
  GAP_ADTYPE_FLAGS,
  GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,

0x1B,                                // length of this data including the data type    byte
  GAP_ADTYPE_MANUFACTURER_SPECIFIC,   // manufacturer specific advertising data type
  0x0D,                               // TI Company code
  0x00,                               // TI Company code
  0x00,                                // user specified data (this is the max length
  0x01,                                // - can be shorter)
  0x02,
   ...
  0x16,
  0x17,
  0x18,
};
```

The flags and company code are set the same as with the iBeacon technology (though the company code has changed to indicate that it is not a device using an Apple standard). The rest of the space is left to be configured by the user. Currently the maximum of 31 bytes is being used, but the advertising data can be shorter.

The advertising interval is set the same way as is described in Section 4 for the beacon project. Again, an interval of 100 ms is used. This can be changed, as long as the requirements discussed in Section 2.5 are followed.

5.3 Testing

The TI Packet Sniffer was used to test the device. Figure 12 shows an advertising packet as it was discovered by the sniffer. It can be seen that the advertising data and PDU type show what is expected by the device.

Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData
			Type	TxAdd	RxAdd	PDU-Length		02 01 04 1B FF 0D 00 00 01 02 03 04 05 06 07 08
0x25	0x8E89BED6	ADV_NON_CONN_IND	2	0	0	37	0xB0B448CF6A03	09 0A 0B 0C 0D 0E 10 11 12 13 14 15 16 17 18

Figure 12. Sniffer Packet Showing the Proprietary Advertising Data

6 References

1. Bluetooth.org, Core specification v4.2: https://www.bluetooth.org/en-us/specification/adopted-specifications
2. Bluetooth.org, Supplements to the Core Specification (CSS) v.4: https://www.bluetooth.org/en-us/specification/adopted-specifications
3. Bluetooth.org, Assigned Numbers, GAP, 22.10.2014 13:37: https://www.bluetooth.org/en-us/specification/adopted-specifications
4. Indoor Light Energy Harvesting Reference Design for Bluetooth low energy (BLE) Beacon Subsystem: http://www.ti.com/tool/tida-00100
5. TI Online Store: https://estore.ti.com/
6. TI Bluetooth low energy Software Developer's Guide, included with the BLE SDK Installer: www.ti.com/ble-stack
7. TI Packet Sniffer: http://www.ti.com/tool/packet-sniffer
8. iBeacon Artwork and Specification: https://developer.apple.com/ibeacon

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (January 2015) to A Revision **Page**

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265