Chapter 8: Beacons

In this chapter, you will learn about the different uses for a Bluetooth® LE advertising packet. Furthermore, you will learn about Bluetooth® LE beacons which are devices that use advertising packets to broadcast useful information without requiring (or even allowing) a connection.

Table of contents

Document conventions

Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

8.1 Advertising Packets

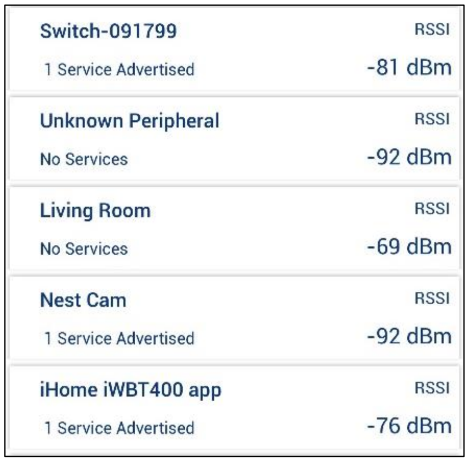
There are three main uses for advertising packets:

- Identifying a peripheral with some recognizable data so that a central knows it can connect to it.
- Sending out data (e.g. beacon data).
- Implementing a Bluetooth mesh network (which uses advertising packets to send and receive data).

Up until now, we have only been using advertising to allow a central to identify a peripheral. In this chapter, we will use advertising to send out beacon data. The third use - Bluetooth® mesh - is covered in a separate class.

8.1.1 Using the Advertising Packet to Get Connected

If you turn on the CySmart scanner, you will find that there are likely a bunch of unknown devices that are advertising around you. For instance, in the figure below you can see that there are quite a few Bluetooth LE devices around me.



Switch-091799 RSSI
1 Service Advertised -81 dBm

Unknown Peripheral RSSI
No Services -92 dBm

Living Room RSSI
No Services -69 dBm

Nest Cam RSSI
1 Service Advertised -92 dBm

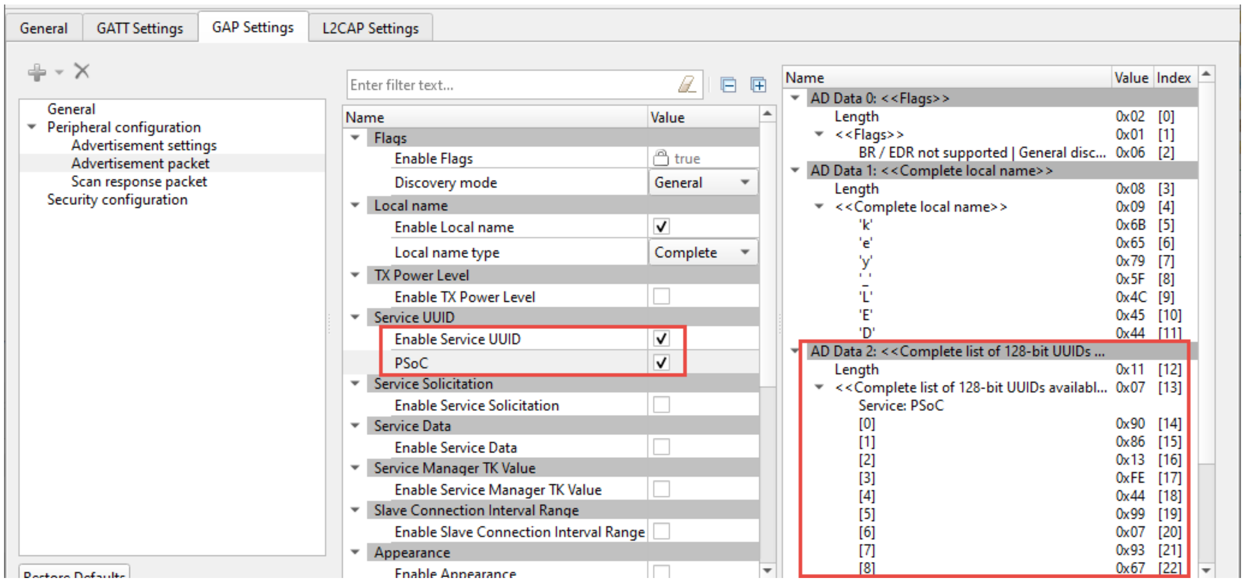
iHome iWBT400 app RSSI
1 Service Advertised -76 dBm

When a central wants to connect to a peripheral, how does it know which peripheral to talk to? There are two common ways to address that question.

The first way is to advertise a service that the central knows about (because it is defined by the Bluetooth SIG or is custom to your company). In the picture above, you can see that some of the devices are advertising that they support 1 service.

You can advertise a complete list of services (if it fits in the packet) or just a subset of services. The services themselves are specified by including the service UUID which can be 128 bits, 32 bits, or 16 bits depending on the service.

The Bluetooth® configurator can be used to advertise one or more services – just check the box for **Enable Service UUID** and then check the box next to each service that you want to advertise. Remember that the packet can only be 31 bytes total so you can't advertise many services before you run out of space.



Name	Value
Flags	
Enable Flags	true
Discovery mode	General
Local name	
Enable Local name	✓
Local name type	Complete
TX Power Level	
Enable TX Power Level	
Service UUID	
Enable Service UUID	✓
PSoC	✓
Service Solicitation	
Enable Service Solicitation	
Service Data	
Enable Service Data	
Service Manager TK Value	
Enable Service Manager TK Value	
Slave Connection Interval Range	
Enable Slave Connection Interval Range	
Appearance	
Enable Appearance	

Name	Value	Index
AD Data 0: <<Flags>>		
Length	0x02	[0]
<<Flags>>	0x01	[1]
BR / EDR not supported | General disc...	0x06	[2]
AD Data 1: <<Complete local name>>		
Length	0x08	[3]
<<Complete local name>>	0x09	[4]
'k'	0x6B	[5]
'e'	0x65	[6]
'y'	0x79	[7]
'_'	0x5F	[8]
'L'	0x4C	[9]
'E'	0x45	[10]
'D'	0x44	[11]
AD Data 2: <<Complete list of 128-bit UUIDs ...		
Length	0x11	[12]
<<Complete list of 128-bit UUIDs availabl...	0x07	[13]
Service: PSoC		
[0]	0x90	[14]
[1]	0x86	[15]
[2]	0x13	[16]
[3]	0xFE	[17]
[4]	0x44	[18]
[5]	0x99	[19]
[6]	0x07	[20]
[7]	0x93	[21]
[8]	0x67	[22]

The advertising flag will depend on how many and what type of services you advertise and it indicates whether all services are being advertised or not. The possibilities are:

```
BTM_BLE_ADVERT_TYPE_16SRV_PARTIAL
BTM_BLE_ADVERT_TYPE_16SRV_COMPLETE
BTM_BLE_ADVERT_TYPE_32SRV_PARTIAL
BTM_BLE_ADVERT_TYPE_32SRV_COMPLETE
BTM_BLE_ADVERT_TYPE_128SRV_PARTIAL
BTM_BLE_ADVERT_TYPE_128SRV_COMPLETE
```

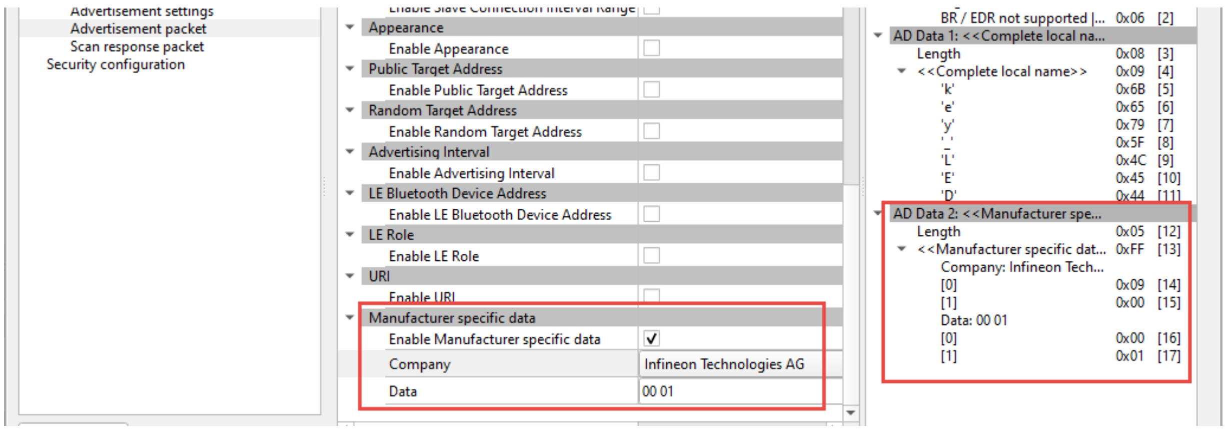
Note: *The advertising data types can be found in btstack/version/wiced_include/wiced_bt_ble.h.*

The second way that is commonly used is to advertise "Manufacturer's Specific Data". The data has two parts:

- A two-byte manufacturer code as specified by the Bluetooth SIG (e.g. Infineon = 0x0009).
- The actual data which is typically a product ID that is unique for each product that the company makes.

The way that this works is that you would write a central application that has a table of known peripheral product IDs that it knows how to talk to. Each peripheral advertises its manufacturer code and product ID in the manufacturers data field. When a central sees something that it knows how to talk to, it can make a connection.

The Bluetooth® configurator can also be used to setup the manufacturer specific data.



Advertisement settings
Advertisement packet
Scan response packet
Security configuration

Enable Slave Connection Interval Range	☐
Appearance	
Enable Appearance	☐
Public Target Address	
Enable Public Target Address	☐
Random Target Address	
Enable Random Target Address	☐
Advertising Interval	
Enable Advertising Interval	☐
LE Bluetooth Device Address	
Enable LE Bluetooth Device Address	☐
LE Role	
Enable LE Role	☐
URI	
Enable URI	☐
Manufacturer specific data	
Enable Manufacturer specific data	✓
Company	Infineon Technologies AG
Data	00 01

BR / EDR not supported |...	0x06	[2]
AD Data 1: <<Complete local na...		
Length	0x08	[3]
<<Complete local name>>	0x09	[4]
'k'	0x6B	[5]
'e'	0x65	[6]
'y'	0x79	[7]
'_'	0x5F	[8]
'L'	0x4C	[9]
'E'	0x45	[10]
'D'	0x44	[11]
AD Data 2: <<Manufacturer spe...		
Length	0x05	[12]
<<Manufacturer specific dat...	0xFF	[13]
Company: Infineon Tech...		
[0]	0x09	[14]
[1]	0x00	[15]
Data: 00 01		
[0]	0x00	[16]
[1]	0x01	[17]

Note: *Multiple bytes in the **Data** field must be separated with spaces.*

8.1.2 Beacons

Bluetooth LE Beacons are devices that are intended to send out data using advertising packets. Often, they will allow connections for configuration of the beacon but not for normal use.

Beacons can be used for lots of different purposes such as providing location (especially in large indoor spaces without GPS coverage (like Shinjuku station in Tokyo - https://allabout-japan.com/en/article/2074), or links to web sites with geographically relevant information (like a website with sale information for a store that you are currently standing in).

There are (of course) two popular types of beacon: iBeacon, which is defined by Apple, and Eddystone which is defined by Google. Each of these will be discussed later in this chapter.

Typically beacons should not send out an advertising packet more often than every 100 ms.

8.1.3 Bluetooth® LE mesh

The third and final common use of advertising packets is for Bluetooth® LE mesh networks. In those networks, advertising packets are used to both send and receive information. Some devices will even provide a bridge between a mesh network and a traditional GATT connection. Bluetooth® LE mesh is a very detailed topic that is covered in a separate training class.

8.2 Creating Advertising Packets Manually

So far, we have been using the Bluetooth® configurator to create the data arrays for advertising packets which is by far the easiest way to do it. However, the configurator does not allow the data to be changed on the fly. Since beacons commonly want to periodically change their advertising data, we will setup the advertising arrays manually in the firmware.

As you learned earlier, an advertising packet is made up of fields where each field has up to 3 parts:

- Length in bytes (not including the Length byte)
- Type
- Optional Data

There is a datatype called `wiced_bt_ble_advert_elem_t` which is a structure that holds the information for one advertising field. It is defined as:

```
/** Advertisement element */
typedef struct
{
    uint8_t                     *p_data;        /**< Advertisement data */
    uint16_t                    len;            /**< Advertisement length */
    wiced_bt_ble_advert_type_t  advert_type;    /**< Advertisement data type */
}wiced_bt_ble_advert_elem_t;
```

Note: *The `len` parameter used here is the length of just the data. It does NOT include the 1-byte for the advertising field type so it is not the same value that will go in the advertising packet itself.*

To create the complete advertising packet, you just create an array of the advertising field structures. For example, the following will setup the advertising packet for a BLE only device that is discoverable and that advertises a name of "my_ble".

```
#define NUM_FIELDS (2)
uint8_t adv_data_0[] = { BTM_BLE_GENERAL_DISCOVERABLE_FLAG |
      BTM_BLE_BREDR_NOT_SUPPORTED };
uint8_t adv_data _1[] = { "my_ble" };
wiced_bt_ble_advert_elem_t adv_packet_data[] =
{
    /* Flags */
    {
        .advert_type = BTM_BLE_ADVERT_TYPE_FLAG,
        .len = sizeof(adv_data_0),
        .p_data = adv_data_0
    },
    /* Complete local name */
    {
        .advert_type = BTM_BLE_ADVERT_TYPE_NAME_COMPLETE,
        .len = strlen(adv_data_1),
        .p_data = adv_data_1
    },

};
```

Once the packet is created in that format, the advertising data can be set using this function:

```
wiced_bt_ble_set_raw_advertisement_data(NUM_FIELDS,
      adv_packet_data);
```

8.3 Multi-advertisement

Beacons can send out multiple advertisement packets to provide different types of data simultaneously. For example, a beacon may send out both iBeacon and Eddystone advertisement packets so that it will appear as both types of beacon. Each advertising instance can have unique parameters if desired.

8.3.1 Multi-advertisement API

There are three functions we will use to setup and use multi-advertising packets (there are two others that we won't use here but are available if needed). Each one uses a parameter called `adv_instance`. This is just an integer from 1 to 16 to uniquely identify each advertising instance that you want to have. The functions are:

wiced_set_multi_advertisment_params

This function sets advertisement parameters for each instance such as the advertising type, advertising interval, advertising address (if a unique address is desired), etc. Its arguments are:

• `advertising_interval_min`	Same as for standard advertising
• `advertising_interval_max`	Same as for standard advertising
• `advertising_type`	Same as for standard advertising
• `own_address_type`	Type of address for this advertising instance – see `wiced_bt_ble_address_type_t`
• `own_address`	Address if unique for this instance (otherwise use `0`)
• `peer_address_type`	Type of address for the peer (only for directed adv)
• `peer_address`	Address if unique for this instance (otherwise use `0`)
• `advertising_channel_map`	List of advertising channels to use (can use 37, 38, 39, or a combination)
• `advertising_filter_policy`	Filter policy – see `wiced_bt_ble_advert_filter_policy_e`
• `adv_instance`	A unique number for this advertisement instance
• `transmit_power`	Transmit power in dB - can use one of `MULTI_ADV_TX_POWER_MIN_INDEX` `MULTI_ADV_TX_POWER_MAX_INDEX`

Note: *You can look in the GeneratedSource/cycfg_gap.h file to see the definitions that the Bluetooth® configurator creates for the parameters above. You can use these definitions when you setup multi-advertisement parameters if appropriate.*

wiced_set_multi_advertisement_data

This function sets advertisement data for multi-advertisement packets. It is analogous to `wiced_bt_ble_set_raw_advertisment_data` but the advertising data is sent as a flat array instead of a structure of advertising elements. Therefore, the procedure to setup the advertising packet will be a bit different as you will see in the exercises. Its arguments are:

• `p_data`	Pointer to an advertising data array
• `data_len`	Length of the advertising data array
• `adv_instance`	Number of the instance specified above

wiced_start_multi_advertisements

This function starts advertisements using the parameters specified above. It is analogous to `wiced_bt_start_advertisments`. Its arguments are:

• `advertising_enable`	`MULTI_ADVERT_START` or `MULTI_ADVERT_STOP`
• `adv_instance`	Number of the instance specified above

8.3.2 Multi-advertisement callback events

The Bluetooth® stack callback function will be called for multi-advertisement events. The event name is `BTM_MULTI_ADVERT_RESP_EVENT`. The event data will indicate the reason for the event. The possible values are:

`SET_ADVT_PARAM_MULTI`	`caused by calling wiced_set_multi_advertisement_params`
`SET_ADVT_DATA_MULTI`	`caused by calling wiced_set_multi_advertisement_data`
`SET_SCAN_RESP_DATA_MULTI`	`caused by calling wiced_set_multi_advertisement_scan_response_data`
`SET_RANDOM_ADDR_MULTI`	`caused by calling wiced_set_multi_advertisements_random_address`
`SET_ADVT_ENABLE_MULTI`	`caused by calling wiced_start_multi_advertisements`

8.4 iBeacon

iBeacon is an Advertising Packet format defined by Apple. The iBeacon information is embedded in the Manufacturer section of the advertising packet. It simply contains:

- Apple's manufacturing ID
- Beacon type (2-bytes)
- Proximity UUID (16-bytes)
- Major number (2-bytes)
- Minor number (2-bytes)
- Measured Power (1-bytes)

Because the packet uses the Apple company ID you need to register with Apple to use iBeacon.

The measured power allows you to calibrate each iBeacon as you install it so that it can be used for indoor location measurement.

8.5 Eddystone

Eddystone is a Google protocol specification that defines a Bluetooth low energy (Bluetooth LE) Advertising message format for proximity beacon messages. It describes several different frame types that may be used individually or in combinations to create beacons that can be used for a variety of applications.

There are four types of Eddystone Frames:

- UID – A unique beacon ID for use in mapping functions

- URL – An HTTP URL in a compressed format
- TLM – Telemetry information about the beacon such as battery voltage, device temperature, counts of packet broadcasts
- EID – Ephemeral ID packets which broadcast a randomly changing number

TLM frames do not show up as a separate beacon but rather are associated with other frames from the same device. For example, you may have a beacon that broadcasts UID frames, URL frames, and TLM frames. In a beacon scanner, that will appear as a UID&TLM beacon and a URL&TLM beacon.

The Eddystone Advertising Packet has the following fields:

- Flags
  - Type: BTM_BLE_ADVERT_TYPE_FLAG (0x01)
  - Value: BTM_BLE_GENERAL_DISCOVERABLE_FLAG | BTM_BLE_BREDR_NOT_SUPPORTED
- 16-bit Eddystone Service UUID
  - Type: BTM_BLE_ADVERT_TYPE_16SRV_COMPLETE (0x03)
  - Value: 0xFEAA
- Service Data
  - Type: BTM_BLE_ADVERT_TYPE_SERVICE_DATA (0x16)
  - Value: The Eddystone Service UUID (0xFEAA), the Eddystone frame type, then the actual data. Frame types are:
    - UID – 0x00
    - URL – 0x10
    - TLM – 0x20
    - EID – 0x30

The data required depends on the frame type. For example, a UID frame has: 1 byte of Tx power of the beacon at 0 m and a 16-byte beacon ID consisting of 10-byte namespace, and 6-byte instance.

In the application BLE-20819EVB02.beacon there is an example of a beacon that acts as both an Eddystone and iBeacon at the same time. The example uses 5 advertising instances - one for each of the four Eddystone frame types and one for iBeacon.

If you are using Eddystone to send a URL, it is limited to 15 characters excluding a prefix (http://, https://, http://www., or https://www.) and a suffix (.com, .com/, .org, .org/, .edu, .edu/, etc.). If you need to create a shorter URL for a site, there are sites that will allow you to create a short URL alias such as www.tinyurl.com.

You can find the detailed Eddystone spec at https://github.com/google/eddystone.

There is also an Eddystone GATT configuration service that can be used to configure and register Eddystone-EID beacons and enable interoperability between hardware manufacturers and developers. That service is not covered in this chapter.

8.6 Exercises

Exercise 1: Eddystone URL beacon

In this exercise, you will create an Eddystone beacon that will advertise the URL for https://www.infineon.com. From your phone you will be able to scan for the beacon (using a beacon scanner app) and then directly connect to the advertised website.

Application Creation

☐ 1. Create a new ModusToolbox™ application for the CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator to use the template application from the class files under *Templates/ch08_ex01_eddy.*

Note: *This is a very simple application with no GATT support. All it does is advertise.*

☐ 2. Open the Bluetooth® configurator.

☐ 3. On the **General** tab, disable the GATT database.

☐ 4. On the **GAP Settings > General** tab, set the device name to **<inits>_eddy**. Check the box for **MAC-based 'Company assigned' part of address**.

☐ 5. On the **GAP Settings > Advertisement settings** tab, uncheck **Enable high duty advertising timeout** so that advertising will continue forever. Change the **High duty advertising interval minimum** and **High duty advertising interval maximum** values to 100.

☐ 6. Save edits and close the configurator.

☐ 7. Complete the `app_set_advertisement_data` function provided in the template to create an advertising packet with the following 3 fields:

a. `BTM_BLE_ADVERT_TYPE_FLAG`

Set this to the same value used previously. That is:
`BTM_BLE_GENERAL_DISCOVERABLE_FLAG | BTM_BLE_BREDR_NOT_SUPPORTED`

b. `BTM_BLE_ADVERT_TYPE_16_SRV_COMPLETE`

This is the 2-byte Eddystone service UUID of 0xFEAA. Note that Bluetooth® uses little-endian so the LSB must be the first element in the array.

c. `BTM_BLE_ADVERT_TYPE_SERVICE_DATA`

This is the Eddystone URL frame it must contain:

- The Eddystone service UUID again (in little-endian)
- The Eddystone frame type for a URL frame (0x10)
- Transmit power (use 0xF0)
- The Eddystone URL scheme prefix for *https:/www.*
- The URL as a list of characters – 'i', 'n', 'f', 'i', 'n', 'e', 'o', 'n'
- The Eddystone URL suffix for *.com*

Note: *See the Eddystone website for a list of frame types, URL prefixes, and URL suffixes along with other useful information about Eddystone.*

Note: *Look for TODO in main.c to find the places that need to be completed.*

Testing

☐ 1. Program the application to your kit.

☐ 2. Open a beacon scanner app on your phone and start scanning.

Note: *Most beacon scanner apps don't show the device address or the device name so if there is more than one beacon running you may not be able to tell which one is yours from the beacon scanner. If this is the case, you can change the URL to something other than https://infineon.com.*

☐ 3. Open the URL for https://www.infineon.com from the beacon app by tapping on the link.

Note: *If you don't see your device in the beacon app it most likely means your packet isn't correct, so it isn't identifying your device as an Eddystone beacon. If so, use the CySmart PC application to scan for your device and look at its advertising packet to determine what's wrong.*

Exercise 2: Eddystone URL, UID, and TLM beacon using multi-advertising

In this exercise you will use multi-advertising to send UID, URL and TLM frames to the listening devices.

☐ 1. Create a new ModusToolbox™ application for the CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator to use the template application from the class files under *Templates/ch08_ex01_eddy_multi.*

☐ 2. Open the Bluetooth® configurator.

☐ 3. On the **General** tab, disable the GATT database.

☐ 4. On the **GAP Settings > General** tab, set the device name to **<inits>_eddy_multi**. Check the box for **MAC-based 'Company assigned' part of address**.

☐ 5. On the **GAP Settings > Advertisement settings** tab, uncheck **Enable high duty advertising timeout** so that advertising will continue forever. Change the **High duty advertising interval minimum** and **High duty advertising interval maximum** values to 100.

☐ 6. Save edits and close the configurator

☐ 7. Open *main.c*. Look for TODO to find places that need to be completed. These are detailed in the steps below.

☐ 8. Enter the advertising parameters in `adv_parameters` to be used for multi-advertising packets. The same values will be used for all advertising instances.

For the advertising min and max, use the values that were generated by the configurator.

Note: *The definitions can be found in GeneratedSource/cycfg_gap.h.*

Note: *The values are in units of 0.625 ms so the value should be 160 (160 * 0.625 = 100 ms)*

For the advertising type, use the macro for non-connectable multi-advertising.

Note: *You can find the possible values in wiced_bt_ble.h in the enumeration for* `wiced_bt_multi_advert_tyep_e`*.*

Advertise on all channels:

```
BTM_BLE_ADVERT_CHNL_37 | BTM_BLE_ADVERT_CHNL_38 | BTM_BLE_ADVERT_CHNL_39
```

For the filter policy, allow all requests.

Note: *You can find the possible values in wiced_bt_ble.h in the enumeration for* `wiced_bt_ble_advert_filter_policy_e`*.*

For TX power, use the max power:

```
MULTI_ADV_TX_POWER_MAX_INDEX
```

We will not use unique addresses for each advertising instance, so the peer and own addresses are 0.

☐ 9. In the `BTM_ENABLED_EVT`, the function to setup and start URL frames is already done for you. Use that as an example to do the same for UID and TLM frames.

☐ 10. The function app_set_advertisement_data_url has been done for you. Use that as a template to complete app_set_advertisement_data_uid.

Note: *Refer to the Eddystone website to find details of UID frames. Each frame type has its own subdirectory with its own README.md file.*

Note: *Use 0xF0 for the ranging data.*

Note: *Use your initials for the first few bytes of the namespace. Use any characters you wish for the remaining values in the packet.*

☐ 11. Complete the function app_set_advertisement_data_tlm.

Note: *Refer to the Eddystone website to find details of TLM frames. Each packet type has its own subdirectory with its own README.md file. Use unencrypted TLM frames.*

Note: *The template includes a timer that fires every 100 ms and increments the variable* `uptime`*. Use that as the value for SEC_CNT in the TLM packet. The value in this case is big-endian. Values for VBATT, TEMP, and ADV_CNT can be set to 0.*

Note: *The timer also calls* `the app_set_advertisement_data_tlm` *function so it will update the TLM packet's data every 100 ms.*

☐ 12. Examine the Bluetooth® stack callback hander to see the new event for `BTM_MULTI_ADVERT_RESP_EVENT`.

Testing

☐ 1. Program the application to your kit.

☐ 2. Open a beacon scanner app on your phone and start scanning.

Note: Most beacon scanner apps don't show the device address or the device name so if there is more than one beacon running you may not be able to tell which one is yours from the beacon scanner. If this is the case, you can change the URL to something other than https://infineon.com.

You should see two beacons:

- one that shows URL and TLM information – the Uptime displayed should increment
- one that shows UID and TLM information – the Uptime displayed should increment

Note: Some beacon scanner apps mistakenly show the uptime value in ms but indicate that it is seconds. This is an issue with the scanner app, not the beacon itself.

Note: If you don't see your device in the beacon app it most likely means your packet isn't correct, so it isn't identifying your device as an Eddystone beacon. If so, use the CySmart PC application to scan for your device and look at its advertising packet to determine what's wrong. Since all 3 packet types are going to the same address you will randomly see one when you start/stop scanning. You can comment out the call to start the other packets if necessary to make debugging easier.

Published by

Infineon Technologies AG

81726 Munich, Germany