Chapter 7: 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**.

7.1 Advertising Packets

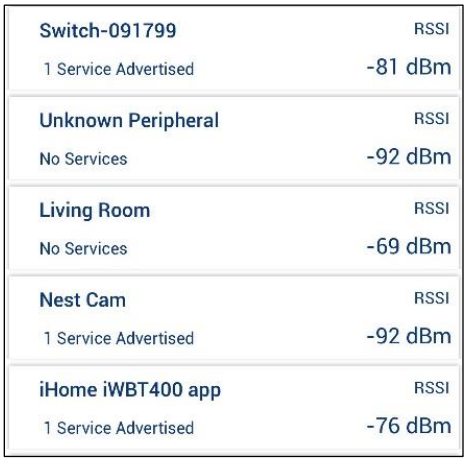
There are three main uses for advertising packets:

- 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.

7.1.1 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.



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 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 mtb_shared/wiced_btsdk/dev-kit/baselib/<device>/version/COMPONENT_<device>/ 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 device sees something that it knows how to talk to, it can make a connection.

The advertising flag for manufacturer specific data is `BTM_BLE_ADVERT_TYPE_MANUFACTURER`.

7.1.2 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.

7.1.3 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.

7.2 Beacon Library

The BTSDK includes a library called *Bluetooth Low Energy* (*btsdk-ble*) that contains support for a host of profiles, beacons, and GATT utilities. In fact, we used the GATT utilities in the chapter that covers Central devices. The beacon support includes a number of functions to help with both Eddystone and iBeacon applications.

To use the library, there are 3 steps:

1. Open the library manager and include the "Bluetooth® Low Energy" library. Click **Update**. Once the update is finished, click **Close**.
2. In the *makefile*, add a component entry to include the beacon utilities portion of the library:

   ```
   COMPONENTS += beacon_lib
   ```

3. In the source code, include the header file for the beacon utilities:

   ```
   #include "wiced_bt_beacon.h"
   ```

The library source code is in *mtb_shared/wiced_btsdk/dev-kit/libraries/btsdk-ble/<version>/COMOPNENT_beacon_lib*. The header file is in *mtb_shared/wiced_btsdk/dev-kit/btsdk-include/<version>.*

7.3 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.

7.3.1 Multi-advertisement API

There are three functions we will use to set up 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`

The function prototype is:

```
wiced_bt_dev_status_t wiced_set_multi_advertisment_params (
uint8_t adv_instance, wiced_bt_ble_multi_adv_params_t* p_param)
```

This function specifies the advertising instance as its first argument. The second argument is a pointer to a structure that sets the advertisement parameters for the specified instance. The structure has the following elements:

- `uint16_t adv_int_min`
- `uint16_t adv_int_max`
  Both are the same as for standard advertising
- `wiced_bt_ble_multi_advert_type_t adv_type`
  Same as for standard advertising
- `wiced_bt_ble_advert_chnl_map_t channel_map`
  List of advertising channels to use (can use 37, 38, 39, or a combination)
- `wiced_bt_ble_advert_filter_policy_t adv_filter_policy`
  Advertising filter policy
- `wiced_bt_ble_adv_tx_power_t adv_tx_power`
  Advertising Tx power as index into power table (`MULTI_ADV_TX_POWER_MIN` - `MULTI_ADV_TX_POWER_MAX`).
- `wiced_bt_device_address_t peer_bd_addr`
  Address for the peer (only for directed advertising, otherwise use `{0}`)
- `wiced_bt_ble_address_type_t peer_addr_type`
  Type of address for the peer (only for directed advertising, otherwise use `BLE_ADDR_PUBLIC`)
- `wiced_bt_device_address_t own_bd_addr`
  Bluetooth® address if it is unique for this instance (otherwise use `{0}`)
- `wiced_bt_ble_address_type_t own_addr_type`
  Type of address if it is unique for this instance (otherwise, use `BLE_ADDR_PUBLIC`)

See the documentation for each data type for possible settings for each element. As an example, the following will configure a non-connectable advertising instance that advertises on all channels using a random address with no filtering and max power.

```
wiced_bt_ble_multi_adv_params_t params =
{
  .adv_int_min = 160,
  .adv_int_max = 160,
  .adv_type = MULTI_ADVERT_NONCONNECTABLE_EVENT,
  .channel_map = BTM_BLE_DEFAULT_ADVERT_CHNL_MAP,
  .adv_filter_policy = MULTI_ADVERT_FILTER_POLICY_FILTER_ACCEPT_LIST_NOT_USED,
  .adv_tx_power = MULTI_ADV_TX_POWER_MAX,
  .peer_addr_type = 0,              //valid only for directed
  .peer_bd_addr = BLE_ADDR_PUBLIC,  //valid only for directed
  .own_bd_addr = 0;                 //use the same address as the device
  .own_addr_type = BLE_ADDR_PUBLIC  //use the same address as the device
};
```

wiced_set_multi_advertisement_data

The function prototype is:

```
wiced_bt_dev_status_t wiced_set_multi_advertisement_data (
uint8_t * p_data, uint8_t data_len, uint8_t adv_instance )
```

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 set up 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` Same as specified in `wiced_set_multi_advertisment_params`

wiced_start_multi_advertisements

The function prototype is:

```
wiced_bt_dev_status_t wiced_start_multi_advertisements (
uint8_t advertising_enable, uint8_t adv_instance)
```

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` Same as specified in `wiced_set_multi_advertisment_params`

7.4 Advertising packet interval units

One important note about the advertising packet interval settings is that they are specified in units of 0.625ms. Therefore, if you want an interval of 100ms, you need to enter a value of 160. This applies to both the settings in *app_bt_cfg.c* and in the multi-advertisement functions. The units can be found in the file *mtb_shared/wiced_btsdk/dev-kit/baselib/<device>/<version>/COMPONENT_<device>/include/wiced_bt_cfg.h*. See the comment next to each parameter for a description of the units.

7.5 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)

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

Because the packet uses the Apple company ID, you need to register with Apple to use iBeacon. For that reason, we will not cover iBeacon in detail in this class, but there is a code example available if you want to explore further. In fact, it demonstrates iBeacon and Eddystone beacons simultaneously. Just enter "beacon" in the search box when selecting the template application in Project Creator.

7.6 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.

As mentioned above, there is a code example demonstrates both an Eddystone and iBeacon at the same time. It 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 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.

The beacon utility library has the following functions to help in setting up Eddystone frames. In each case, the functions fills in `adv_data` with the correct Eddystone frame data and `adv_len` with the length of the frame.

```
void wiced_bt_eddystone_set_data_for_url(
      uint8_t tx_power,
      uint8_t urlscheme,
      uint8_t encoded_url[EDDYSTONE_URL_VALUE_MAX_LEN],
      uint8_t adv_data[WICED_BT_BEACON_ADV_DATA_MAX],
      uint8_t *adv_len);

void wiced_bt_eddystone_set_data_for_uid(
      uint8_t eddystone_ranging_data,
      uint8_t eddystone_namespace[EDDYSTONE_UID_NAMESPACE_LEN],
      uint8_t eddystone_instance[EDDYSTONE_UID_INSTANCE_ID_LEN],
      uint8_t adv_data[WICED_BT_BEACON_ADV_DATA_MAX],
      uint8_t *adv_len);

void wiced_bt_eddystone_set_data_for_tlm_unencrypted(
      uint16_t vbatt,
      uint16_t temp,
      uint32_t adv_cnt,
      uint32_t sec_cnt,
      uint8_t  adv_data[WICED_BT_BEACON_ADV_DATA_MAX],
      uint8_t  *adv_len);

void wiced_bt_eddystone_set_data_for_tlm_encrypted(
      uint8_t  etlm[EDDYSTONE_ETLM_LEN],
      uint16_t salt,
      uint16_t mic,
      uint8_t  adv_data[WICED_BT_BEACON_ADV_DATA_MAX],
      uint8_t  *adv_len);

void wiced_bt_eddystone_set_data_for_eid(
      uint8_t eddystone_ranging_data,
      uint8_t eid[EDDYSTONE_EID_LEN],
      uint8_t adv_data[WICED_BT_BEACON_ADV_DATA_MAX],
      uint8_t *adv_len);
```

After calling one of these functions, you can pass `adv_data` and `adv_len` to `wiced_set_multi_advertisement_data` instead of having to create an array for the packet manually.

7.7 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 by clicking on the link in the scanner app.

Note: *In this application, you will set up the Eddystone packet manually so that you understand how it is done. In the next exercise, you will get a chance to use the beacon library functions with multi advertising to simplify the process.*

Application Creation

☐ 1. Create a new ModusToolbox™ application for the CYW920835M2EVB-01 BSP.

Use the Import functionality in project creator to use the template application from the class files under *Templates/ch07_ex01_eddy.*

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

☐ 2. Open the Bluetooth® Configurator and change the device name to <inits>_eddy.

☐ 3. Save edits and close the configurator.

☐ 4. In the *app_bt_cfg.c* file, change the `ble_advert_cfg` settings as follows:

```
high_duty_min_interval = 160
high_duty_max_interval = 160
high_duty_duration = 0
```

Note: *This will result in an advertising packet every 100 ms (since the units are 0.625ms) with no timeout.*

☐ 5. In *app.c*, 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 (just 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 app.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 CYW920835M2EVB-01 BSP.

   Use the Import functionality in project creator to use the template application from the class files under *Templates/ch07_ex01_eddy_multi.*

2. Open the Bluetooth® Configurator and change the device name to <inits>_eddy_multi.
3. Save edits and close the configurator.
4. Open the library manager and add the *Bluetooth Low Energy* library (*btsdk-ble)*.
5. In the m*akefile*, edit the `COMPONENTS` variable to include `beacon_lib`:

   ```
   COMPONENTS += bsp_design_modus beacon_lib
   ```

6. Open *app.c and look for TODO* to find places that need to be completed. These are detailed in the steps below.
7. Add the include for the beacon library: *wiced_bt_beacon.h*.
8. Enter the advertising parameters in the `adv_parameters` structure to be used for multi-advertising packets. The same values will be used for all advertising instances.

   For the advertising min and max, use 160 so that we will get a packet every 100ms.

   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_DEFAULT_ADVERT_CHNL_MAP`

   For the filter policy, specify that a filter accept list is not used.

   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}` and the address types are `BLE_ADDR_PUBLIC`.

9. In the `app_multi_advertise` function, the code to set up and start URL frames is already done for you using advertising instance 1. Use that as an example to do the same for UID frames. Use advertising instance 2.

   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.*

☐ 10. In the `app_multi_advertise` function, set up TLM frames using advertising instance 3.

Note: *Set up an unencrypted TLM packet (`wiced_bt_eddystone_set_data_for_tlm_unencrypted`).*

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.*

☐ 11. In the `timer_cback` function, set up the TLM packet so that it uses the updated time and set the multi advertisement data.

Note: *The timer is configured so that it executes the callback function `timer_cback` every 100ms.*

Note: *You just need to copy the 2 appropriate lines from the `app_multi_advertise` function to the `timer_cback` function. You do not need to configure or start multi advertisements again, just creating the new packet and updating the multi advertisement data is sufficient since the advertising instance is already running.*

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.*

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by

Infineon Technologies AG

81726 Munich, Germany

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.