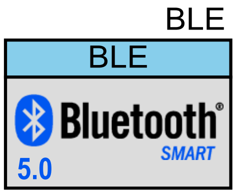
PSoC® Creator™ Component Datasheet

Bluetooth Low Energy (BLE_PDL)

2.0



Features

- Multi-link support for up to four simultaneous connections in any combination of roles
- Bluetooth v5.0 compliant protocol stack
- Generic Access Profile (GAP) and Generic Attribute Profile (GATT)
- Security Manager
- Logical Link Adaption Protocol (L2CAP) connection-oriented channel
- Link Layer (LL)

General Description

The Bluetooth Low Energy (BLE) Peripheral Driver Library (PDL) Component provides a comprehensive GUI-based configuration window to facilitate designing applications requiring BLE connectivity. The BLE_PDL Component incorporates a Bluetooth Core Specification v5.0 compliant protocol stack and provides APIs to enable user applications to access the underlying hardware via the stack.

The BLE_PDL Component is a hybrid: graphical configuration entity with a set of Component-specific API built on top of the BLE middleware available in the PDL. It allows schematic-based connections and hardware configuration as defined by the Component Configure dialog.

When to use the BLE_PDL Component

BLE is used in very low-power network and Internet of Things (IoT) solutions aimed for low-cost battery operated devices that can quickly connect and form simple wireless links. Target applications include HID, remote controls, sports and fitness monitors, portable medical devices and smart phone accessories, among many others that are being added to a long list of BLE supporting solutions.

SIG adopted Profiles and Services

The BLE_PDL Component supports numerous SIG-adopted GATT-based Profiles and Services. Each of these can be configured for either a GATT Client or GATT Server. The Component generates all the necessary code for a particular Profile/Service operation, as configured in the Component Configure dialog.

Comprehensive APIs

The BLE_PDL Component together with the BLE Middleware Library provide application-level APIs to design solutions without requiring manual stack level configuration. The BLE Middleware Library API documentation is provided in separate HTML-based files.

Custom Profiles

You can create custom Profiles that use existing Services, and you can create custom Services with custom Characteristics and Descriptors.

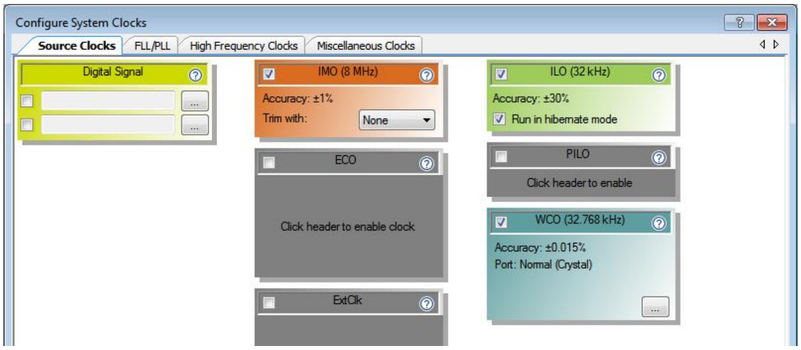
Debug Support

For testing and debugging, the Component can be configured to HCI mode through a Component-embedded UART. See General Tab – BLE Controller only (HCI over UART).

For over-the-air verification, the Cypress CySmart Central Emulation Tool can be used for generic Bluetooth host stack emulation. To launch this tool, right-click on the Component and select **Launch CySmart**.

Quick Start

1. Drag a BLE_PDL Component from the Cypress/Communications folder in the Component Catalog onto your schematic.
2. In the Workspace Explorer, under the Design-Wide Resources (<*project*>*.cydwr*) file, double-click the **Clocks** item in the tree to open the Clock Editor. Then double-click on the WCO clock in the table to open the Configure System Clocks dialog.
   a. Under the **Source Clocks** tab, enable the **WCO** clock.



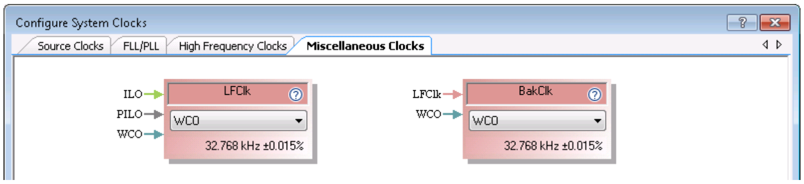
IMO (8 MHz) ☑
Accuracy: ±1%
Trim with: None

ILO (32 kHz) ☑
Accuracy: ±30%
☑ Run in hibernate mode

PILO □

WCO (32.768 kHz) ☑
Accuracy: ±0.015%
Port: Normal (Crystal)

b. Then under the **Miscellaneous Clocks** tab, set “WCO” as the clock source for the Low Frequency (LFClk) clock and for the Backup(BakClk) clock.



c. Close the Configure System Clocks dialog.

3. In the *<project>.cydwr* file, select the **Interrupts** tab.

   a. Assign the BLE interrupt (BLE_bless_isr) to appropriate ARM core where the BLE controller is running. It depends on the option chosen in the **CPU Core** field (see General Tab).

      - for **Single core (Complete Component on CM0+)** option – CM0+
      - for **Single core (Complete Component on CM4)** option – CM4
      - for **Dual core (Controller on CM0+, Host and Profiles on CM4)** option – BLE_blless_interrupt on **CM0+**, other peripheral interrupts on **CM4**

   b. Set the **Priority** level to the highest value.

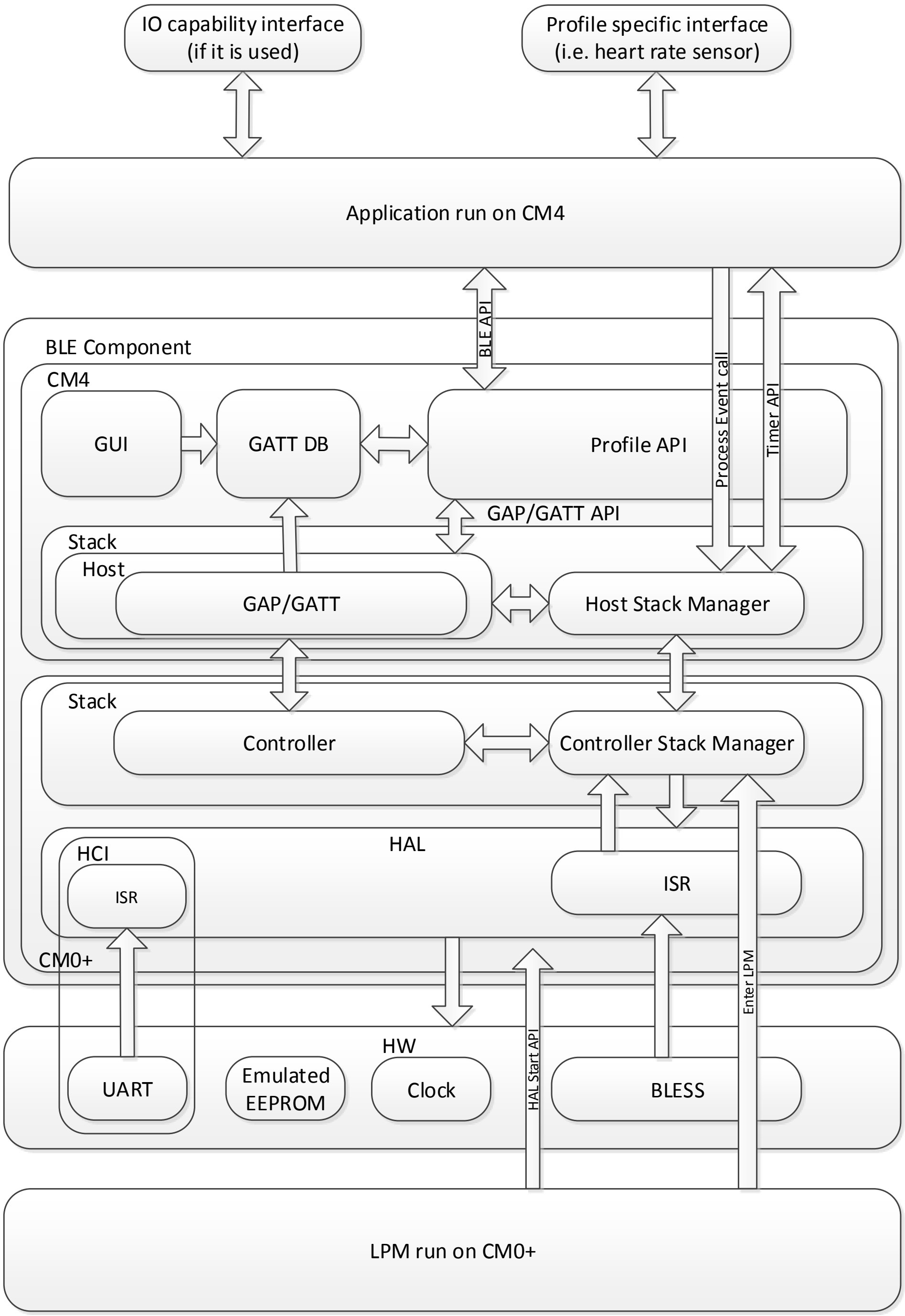
4. Click on **Build** to generate your APIs. Once the clock and interrupt configurations meet the requirements, no warnings or errors should appear.

Note If the **CPU Core** parameter in the BLE Configure dialog is set to Dual core (Controller on CM0+, Host and Profiles on CM4), the IPC configuration file (*cy_ipc_config.h*) (located in the Shared Files folder of the workspace) must be modified. The define CY_IPC_INTR_CYPIPE_MUX_EP0 should be changed to Deep Sleep wake-up compatible interrupt vectors. The valid range: 0..7.

Note The BLE_PDL Component uses the cy_em_eeprom section to store bond data. To use the Em_EEPROM Component/Middleware in pair with the BLE_PDL, modify the linker scripts. For more information, refer to the Middleware/Cypress Em_EEPROM Middleware section of the PDL documentation. To access this document, go to the PSoC Creator **Help menu > Documentation > Peripheral Driver Library**.

BLE_PDL Component Architecture

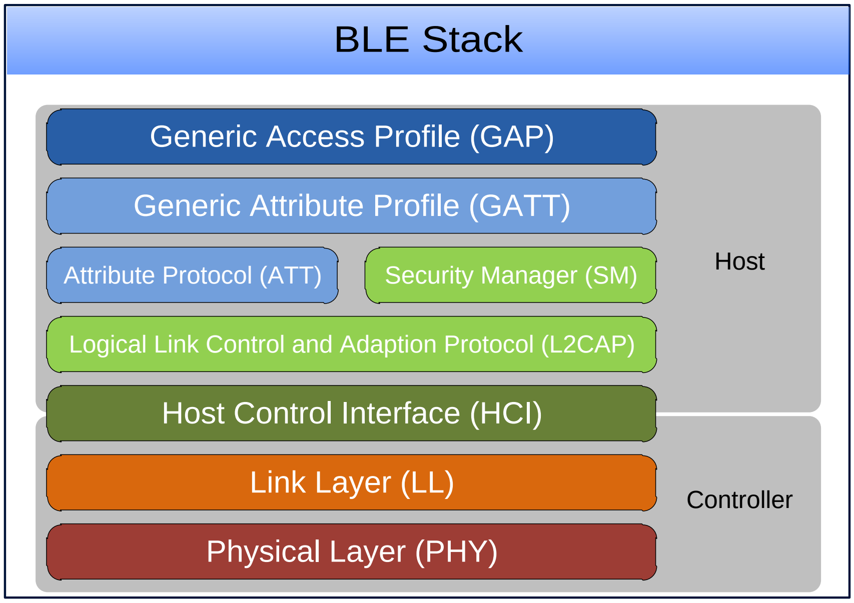
The BLE_PDL Component consists of the BLE Stack, BLE Profile, BLE_PDL Component Hardware Abstraction Layer (HAL), and the Link Layer. The following figure shows a high-level architecture of the BLE_PDL Component, illustrating the relationship between each of the layers and the route in which the application interacts with the Component. Note that the application is informed of the BLE events through the use of callback functions. You may build your state machine using these. Refer to the Callback Functions section for more details.



The following sub-sections give an overview of each of these layers.

BLE Stack

The BLE stack implements the core BLE functionality as defined in the Bluetooth Core Specification 5.0. The stack is included as a precompiled library and is included in the BLE Middleware library. The BLE Stack implements a layered architecture of the BLE protocol stack as shown in the following figure.


BLE Stack
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager (SM)
Logical Link Control and Adaption Protocol (L2CAP)
Host Control Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
Host
Controller

Generic Access Profile (GAP)

The Generic Access Profile defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. In addition, this profile includes common format requirements for parameters accessible on the user interface level.

The Generic Access Profile defines the following roles when operating over the LE physical channel:

- **Broadcaster role:** A device operating in the Broadcaster role can send advertising events. It is referred to as a Broadcaster. It has a transmitter and may have a receiver.
- **Observer role:** A device operating in the Observer role is a device that receives advertising events. It is referred to as an Observer. It has a receiver and may have a transmitter.
- **Peripheral role:** A device that accepts the establishment of an LE physical link using any of the connection establishment procedures is termed to be in a "Peripheral role." A device operating in the Peripheral role will be in the "Slave role" in the Link Layer Connection State. A device operating in the Peripheral role is referred to as a Peripheral. A Peripheral has both a transmitter and a receiver.

- **Central role:** A device that supports the Central role initiates the establishment of a physical connection. A device operating in the "Central role" will be in the "Master role" in the Link Layer Connection. A device operating in the Central role is referred to as a Central. A Central has a transmitter and a receiver.

Generic Attribute Profile (GATT)

The Generic Attribute Profile defines a generic service framework using the ATT protocol layer. This framework defines the procedures and formats of services and their Characteristics. It defines the procedures for Service, Characteristic, and Descriptor discovery, reading, writing, notifying, and indicating Characteristics, as well as configuring the broadcast of Characteristics.

GATT Roles

- GATT Client: This is the device that wants data. It initiates commands and requests towards the GATT Server. It can receive responses, indications, and notifications data sent by the GATT Server.
- GATT Server: This is the device that has the data and accepts incoming commands and requests from the GATT Client and sends responses, indications, and notifications to a GATT Client.

The BLE Stack can support both roles simultaneously.

Attribute Protocol (ATT)

The Attribute Protocol layer defines a Client/Server architecture above the BLE logical transport channel. The attribute protocol allows a device referred to as the GATT Server to expose a set of attributes and their associated values to a peer device referred to as the GATT Client. These attributes exposed by the GATT Server can be discovered, read, and written by a GATT Client, and can be indicated and notified by the GATT Server. All the transactions on attributes are atomic.

Security Manager Protocol (SMP)

Security Manager Protocol defines the procedures and behavior to manage pairing, authentication, and encryption between the devices. These include:

- Encryption and Authentication
- Pairing and Bonding
  - Pass Key and Out of band bonding
- Key Generation for a device identity resolution, data signing and encryption
- Pairing method selection based on the IO capability of the GAP central and GAP peripheral device

Logical Link Control Adaptation Protocol (L2CAP)

L2CAP provides a connectionless data channel. LE L2CAP provides the following features:

- Channel multiplexing, which manages three fixed channels. Two channels are dedicated for higher protocol layers like ATT, SMP. One channel is used for the LE-L2CAP protocol signaling channel for its own use.
- Segmentation and reassembly of packets whose size is up to the BLE Controller managed maximum packet size.
- Connection-oriented channel over a specific application registered using the PSM (protocol service multiplexer) channel. It implements credit-based flow control between two LE L2CAP entities. This feature can be used for BLE applications that require transferring large chunks of data.

Host Controller Interface (HCI)

The HCI layer implements a command, event, and data interface to allow link layer access from upper layers such as GAP, L2CAP, and SMP.

Link Layer (LL)

The LL protocol manages the physical BLE connections between devices. It supports all LL states such as Advertising, Scanning, Initiating, and Connecting (Master and Slave). It implements all the key link control procedures such as LE Encryption, LE Connection Update, LE Channel Update, and LE Ping. The Link Layer is a hardware-firmware co-implementation, where the key time critical LL functions are implemented in the LL hardware. The LL firmware maintains and controls the key LL procedure state machines. It supports all the BLE chip specific low-power modes.

The BLE Stack is a pre-compiled library in the BLE_PDL Component. The appropriate configuration of the BLE Stack library is linked during a build process based on application. The BLE Stack libraries are ARM Embedded Application Binary Interface (EABI) compliant and they are compiled using ARM compiler version 5.03.

The following table shows the mapping between the BLE Stack library to the user-configured Complete BLE Protocol mode or HCI mode. Refer to the General Tab section for selection of stack configuration.

BLE_PDL Component Configuration	CPU Core	BLE Stack Libraries
Complete BLE Protocol	Single core (Complete Component on CM4)	cy_ble_stack_gcc_radio_max_cm4.a cy_ble_stack_gcc_soc_cm4.a

BLE_PDL Component Configuration	CPU Core	BLE Stack Libraries
Host and Controller on single core with software interface	Single core (Complete Component on CM0+)	cy_ble_stack_gcc_radio_max_cm0p.a cy_ble_stack_gcc_soc_cm0p.a
Complete BLE Protocol Host and Controller on dual core with IPC interface	Dual core (Controller on CM0+, Host and Profile on CM4)	cy_ble_stack_gcc_radio_max_cm0p.a cy_ble_stack_gcc_controller_ipc_cm0p.a cy_ble_stack_gcc_host_ipc_cm4.a
HCI Mode	CM4	cy_ble_stack_gcc_radio_max_cm4.a cy_ble_stack_gcc_controller_uart_cm4.a
	CM0+	cy_ble_stack_gcc_radio_max_cm0p.a cy_ble_stack_gcc_controller_uart_cm0p.a

Stack mode and Core parameters can be configured in the **General** tab. For a successful build, make sure that BLE_bless_isr is assigned to the same core with the controller in the Design-Wide Resources Interrupt Editor (*project.cydwr*).

There are two sets of libraries:

- "_gcc_" libraries are built with wchar_t typedef set to 32-bit.
- "_mdk_" libraries are built with wchar_t set to 16-bit.

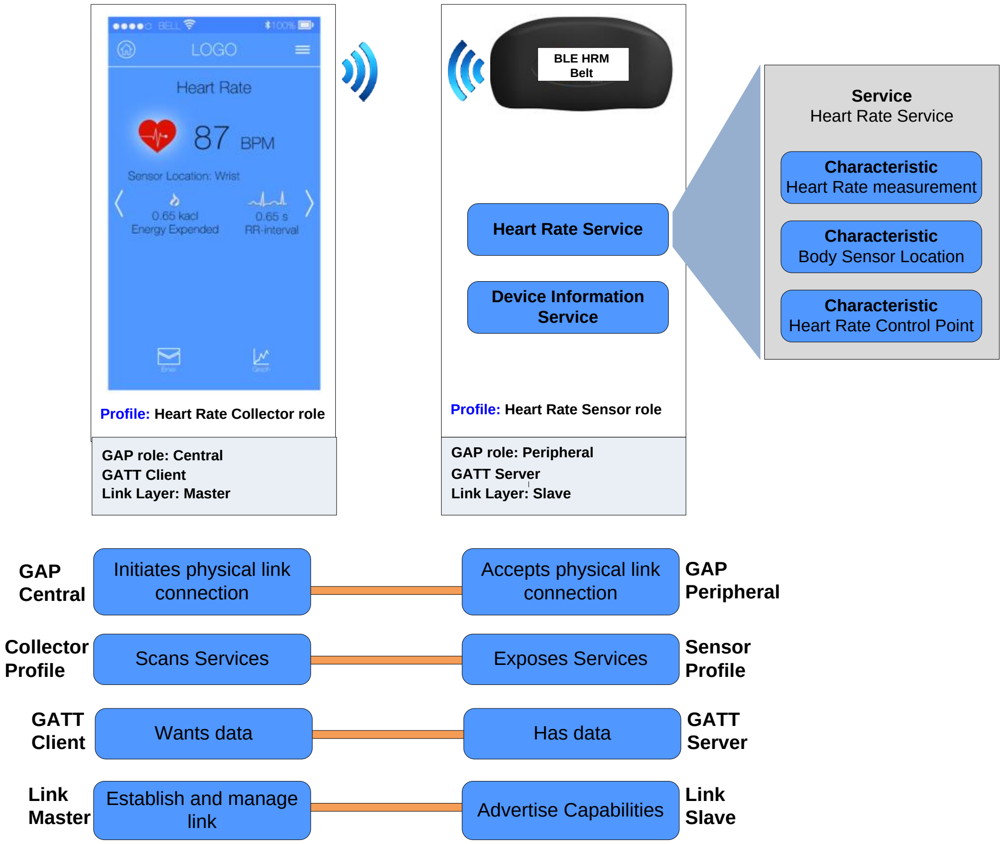
Note The "_gcc_" libraries are used for the IAR compiler.

Profile Layer

In BLE, data is organized into concepts called Profiles, Services, and Characteristics.

- A **Profile** describes how devices connect to each other to find and use Services. It is a definition used by Bluetooth devices to describe the type of application and the general expected behavior of that device. See the Component parameters section for how to configure the BLE_PDL Component.
- A **Service** is a collection of data entities called Characteristics. A Service is used to define a certain function in a Profile. A Service may also define its relationship to other Services. A Service is assigned a Universally Unique Identifier (UUID). This is 16 bits for SIG adopted Services and 128 bits for custom Services. See the Toolbar section for information about adding Services to a Profile.
- A **Characteristic** contains a Value and the Descriptor that describes a Characteristic Value. It is an attribute type for a specific piece of information within a Service. Like a Service, each Characteristic is designated with a UUID; 16 bits for SIG adopted Characteristics and 128 bits for custom Characteristics. See the Toolbar section for information about adding Characteristics and Descriptors.

The following diagram shows the relationship between Profiles, Services, and Characteristics in a sample BLE heart rate monitor application using a Heart Rate Profile.



The Heart Rate Profile contains a Heart Rate Service and a Device Information Service. Within the Heart Rate Service, there are three Characteristics, each containing different information. The device in the diagram is configured as a Sensor role, meaning that in the context of the Heart Rate Profile, the device is a GAP Peripheral and a GATT Server. These concepts are explained in the BLE Stack description.

The Profile layer is generated by PSoC Creator using the parameter configurations specified in the GUI. The Profile implements the Profile specific attribute database and APIs required for the application. You can choose to configure the standard SIG adopted Profile and generate a design or define a Custom Profile required by an application. The GUI also allows import/export of a Profile design in XML format for Profile design reuse. In addition, the Bluetooth Developer Studio compliant XML format is available.

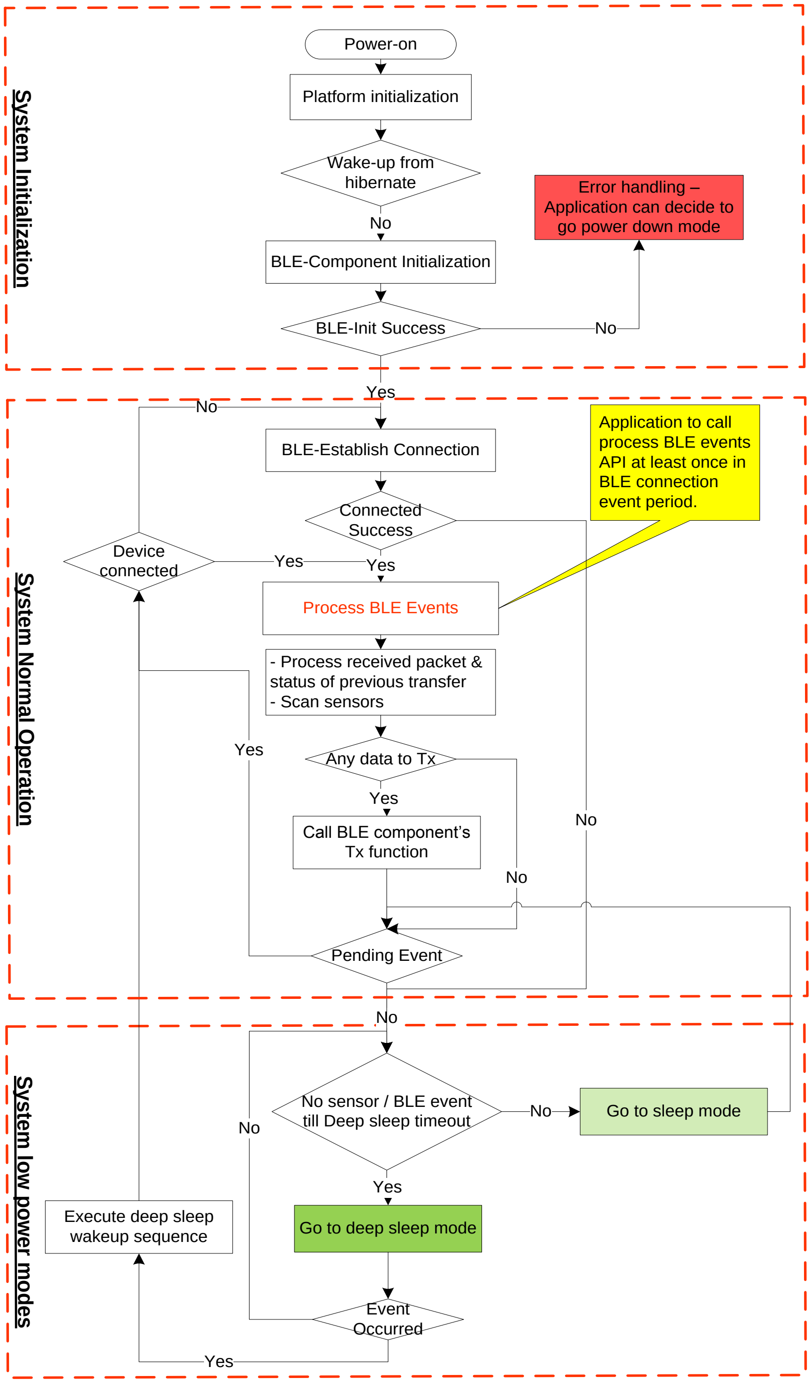
Hardware Abstraction Layer (HAL)

The HAL implements the interface between the BLE stack and the underlying hardware. This layer is meant for the stack only and it is not advisable to modify it.

Functional Description

Operation Flow

A typical application code consists of three separate stages: Initialization, Normal operation, and Low-power operation.


System Initialization
Power-on
Platform initialization
Wake-up from hibernate
No
BLE-Component Initialization
BLE-Init Success
No
Error handling – Application can decide to go power down mode
Yes
System Normal Operation
No
BLE-Establish Connection
Connected Success
Device connected
Yes
Yes
Application to call process BLE events API at least once in BLE connection event period.
Process BLE Events
- Process received packet & status of previous transfer
- Scan sensors
Yes
Any data to Tx
Yes
Call BLE component's Tx function
No
No
Pending Event
No
System low power modes
No sensor / BLE event till Deep sleep timeout
No
Go to sleep mode
No
Yes
Go to deep sleep mode
Execute deep sleep wakeup sequence
Event Occurred
Yes

After initialization, the Component enters normal operation and periodically enters various degrees of low-power operation to conserve power. Therefore, initialization should only happen at system power-up, and the Component should operate between normal mode and low-power mode afterwards.

System Initialization

The initialization stage happens at system power-up or when waking from system hibernation. This stage sets up the platform and the Component parameters. The application code should also start the Component and set up the callback functions for the event callbacks that will happen in the other modes of operation.

System Normal Operation

Upon successful initialization of the BLE_PDL Component or hibernate wakeup sequence, the Component enters normal mode. Normal operation first establishes a BLE connection if it is not already connected. It should then process all pending BLE events by checking the stack status. This is accomplished by calling Cy_BLE_ProcessEvents(). When all events have been processed, it can transmit any data that need to be communicated and enters low-power operation unless there is another pending event. In such a case, it should execute the normal operation flow again. Processing of BLE events should be performed at least once in a BLE connection event period. The BLE connection event is configured by the Central device while establishing a connection.

System Low-Power Operation

When there are no pending interrupts in normal operation, the Component should be placed in low-power mode. It should first enter Sleep mode. The Component can enter either Sleep or Deep Sleep mode depending on the state of the BLE interface hardware. If an event happens at any time in low-power mode, it should re-enter normal operation.

Note The MCU and BLE Sub-System (BLESS) have separate power modes and are able to go to different power modes independent of each other. The check marks in the following table show the possible combination of power modes of MCU and BLESS.

BLESS Power Modes	MCU Power Modes				
	Active	**Sleep**	**Deep Sleep**	**Hibernate**	**Stop**
Active (idle/Tx/Rx)	✓	✓			
Sleep	✓	✓			
Deep Sleep (ECO off)	✓	✓	✓		
Off				✓	✓

Device Bonding

The BLE_PDL Component will store the link key of a connection after pairing with the remote device. If a connection is lost and re-established, the devices will use the previously stored key for the connection.

The BLE stack will update the bonding data in RAM while the devices are connected. If the bonding data is to be retained during shutdown, the application can use Cy_BLE_StoreBondingData() API to write the bonding data from RAM to the dedicated Flash location, as defined by the Component. Refer to the CE215121_BLE_HID_Keyboard code example for usage details.

LFCLK configuration

The LFCLK configuration as set in the **Clocks** tab of the Design-Wide Resources (*<project>.cydwr*) file affects the BLE_PDL Component's ability to operate in Deep Sleep mode. If the WCO is chosen, then the Component Deep Sleep mode is available for use. However, if the ILO is chosen, then the Component cannot enter Deep Sleep.

Note The LFCLK is used in the BLE_PDL Component only during Deep Sleep mode and hence the ILO inaccuracy does not affect the BLE communication.

Multi-Connection Support

The BLE_PDL supports up to four simultaneous Multi-Master Multi-Slave (MMMS) BLE connections in any combination of roles. For example, it can be a master of four slave devices (4M), a master of three slave devices and a slave of another device (3M1S), or a slave of four devices (4S), or any other combination.

To configure the maximum number of BLE connections, refer to the General Tab section (**Maximum number of BLE connections**).

The BLE_PDL Component supports a single instance of a GATT Server (single GATT database). The number of the Server instance field is always fixed to 1 in the BLE_PDL Configure dialog. You can add additional Services or a complete Server Profile to the existing Server tree and build a GATT database. This single GATT database is reused across all BLE connections.

The BLE_PDL Component manages multiple CCCD values. The CCCD value for each active connection is unique. The maximum number of CCCD storage SRAM data structures supported by the Component is determined by the number of active BLE connections/links that you select. The Component restores CCCD values in flash for each of the bonded devices while establishing a connection with a peer device.

Use the connection handle at the application level to manage the multi-connection. The connection handle (cy_stc_ble_conn_handle_t) appears when the connection is established (as the event parameter of the CY_BLE_EVT_ GATT_CONNECT_IND event).

To work with a particular connection, BLE APIs (BLE Stack APIs, BLE Profile APIs) provide the parameter connHandle (e.g., Cy_BLE_BASS_SendNotification (cy_stc_ble_conn_handle_t connHandle …).)

BLE events from the AppCallback function include a connHandle in eventParam structures to distinguish to which connection this event relates.

The following code shows how an application can manage connection handles:

```
/**
 * Allocate the connection handle array.
 * Macro CY_BLE_CONN_COUNT indicates the MAX number of supported connection.
 */

cy_stc_ble_conn_handle_t appConnHandle[CY_BLE_CONN_COUNT] =
{CY_BLE_INVALID_CONN_HANDLE_VALUE};

/* The callback event function to handle various events from the BLE stack */
void AppCallback (uint32 event, void* eventParam)
{
      switch (event)
       {
                     ...
           case CY_BLE_EVT_GATT_CONNECT_IND:
           /* Add the connected device to the connection handle array */
           appConnHandle[ (*(cy_stc_ble_conn_handle_t *)eventParam).attId ] =
                           *(cy_stc_ble_conn_handle_t *) eventParam;
           break;

           case CY_BLE_EVT_GATT_DISCONNECT_IND:
           /* Remove the connected device from the connection handle array */
           memset(&appConnHandle[(*(cy_stc_ble_conn_handle_t*)eventParam).attId],
                  CY_BLE_INVALID_CONN_HANDLE_VALUE,
                  sizeof(cy_stc_ble_conn_handle_t));
           break;
       }
}
```

Loop through all connected devices:

```
for (i = 0; i < CY_BLE_CONN_COUNT; i++)
{
    if(Cy_BLE_GetConnectionState(appConnHandle[i]) == CY_BLE_CONN_STATE_CONNECTED)
    {
     /* Do some action */
    }
}
```

Use the Cy_BLE_GetDeviceRole(cy_stc_ble_conn_handle_t *connHandle) function to discover the role of the link-layer device connected to a peer device with the connection handle indicated by the connHandle parameter.

Write attributes (characteristic, descriptors) requests from the peer device(s) for adopted services (e.g. BAS, HIDS, HRS, etc) handled by the BLE Component.

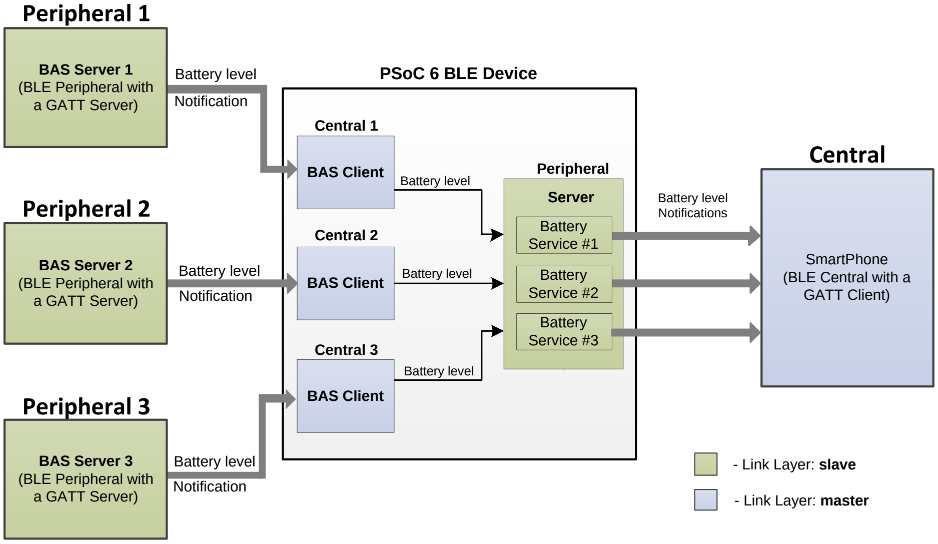
For the Custom service, write attributes requests handled by the application level in the AppCallback () callback event function.

The following code shows the handling of the Write operation by a peer device for the Custom service:

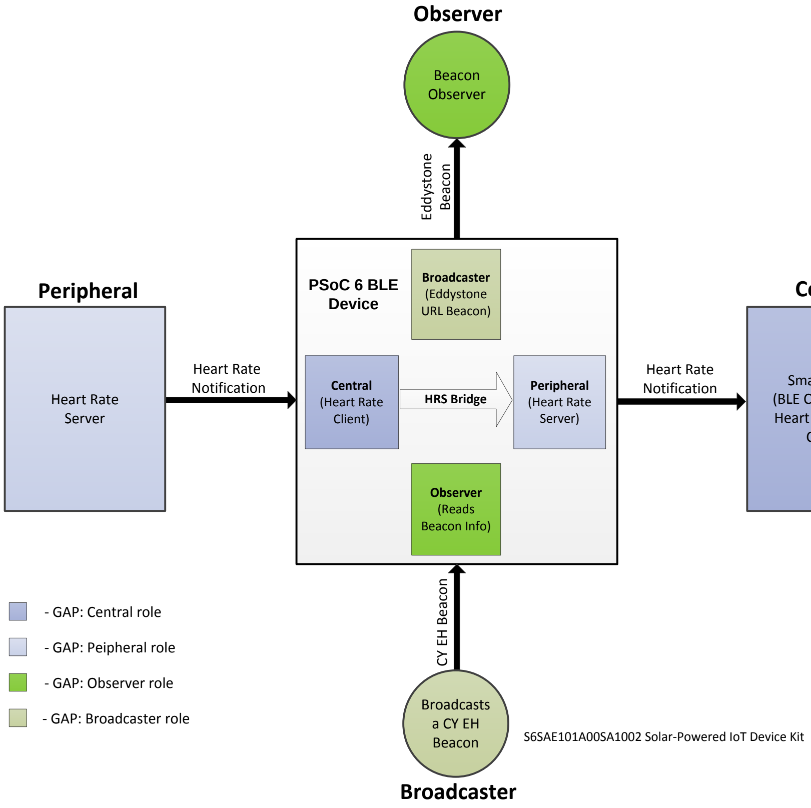
```
/* Call the back event function to handle various events from the BLE Stack */
void AppCallback (uint32 event, void* eventParam)
{
    switch (event)
    {
    ...
        case CY_BLE_EVT_GATTS_WRITE_REQ:
        {
            cy_en_ble_gatt_err_code_t gattErr;
            cy_stc_ble_gatt_write_param_t *writeParam =
                (cy_stc_ble_gatt_write_param_t *)eventParam;

            /* Store data in the database */
            gattErr = Cy_BLE_GATTS_WriteAttributeValuePeer(
                       &writeParam->connHandle, &writeParam->handleValPair);

            if(gattErr != CY_BLE_GATT_ERR_NONE)
            {
              /* Send an Error Response */
               cy_stc_ble_gatt_err_info_t errInfo =
                 {
                     .opCode     = CY_BLE_GATT_WRITE_REQ,
                     .attrHandle = writeParam->handleValPair.attrHandle,
                     .errorCode  = gattErr
                 };
                 Cy_BLE_GATTS_SendErrorRsp(&writeParam->connHandle, &errInfo);
           }
     ...
    }
}
```

The common MMMS usage are Multi-Master Single-Slave and Multi-Role when the BLE_PDL Component is configured in all the GAP roles (Central, Peripheral, Observer, and Broadcaster).

Multi-Master Single-Slave Usage Block Diagram



Multi-Role Usage Block Diagram



Refer to code examples CE215118_BLE_Multi_Master_Single_Slave and CE215555_BLE_Multi_Role for more details.

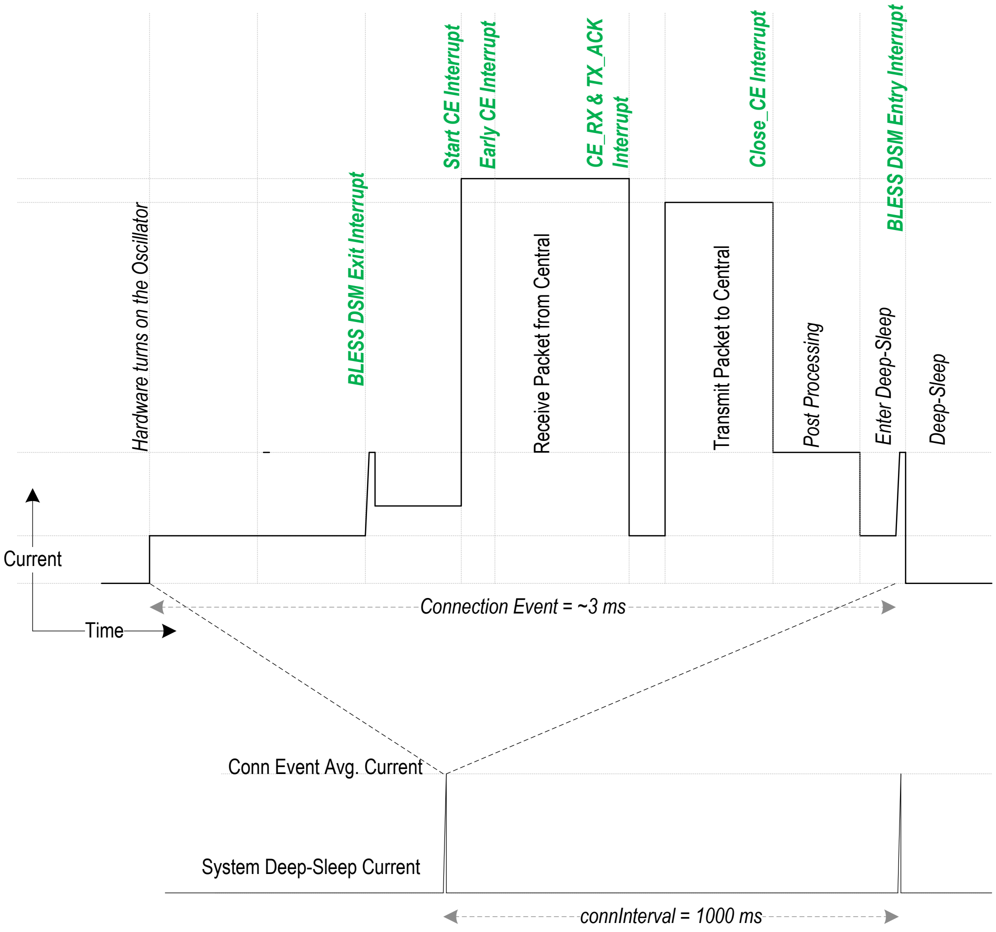
BLE Interrupt Notification Callback

The BLE Component exposes BLE interrupt notifications to the application which indicates a different link layer and radio state transitions to the user from the BLESS interrupt context.

The user registers for a particular type of a callback and the BLE component will call that registered callback basing on the registered mask (Refer to the Cy_BLE_RegisterInterruptCallback() and Cy_BLE_UnRegisterInterruptCallback() APIs). All interrupts masks are specified in the cy_en_ble_interrupt_callback_feature_t enumeration. The possible interrupts which can trigger a user callback:

1. **CY_BLE_INTR_CALLBACK_BLESS_STACK_ISR** – Executed on every trigger of the BLESS interrupt.
2. **CY_BLE_INTR_CALLBACK_BLESS_INTR_STAT_DSM_EXITED** – Executed when the BLESS exits Deep Sleep mode and enters Active mode. A BLESS Deep Sleep exit can be triggered automatically by link layer hardware or by different BLE_PDL data transfer APIs, which needs the BLESS to be active.
3. **CY_BLE_INTR_CALLBACK_BLELL_CONN_EXT_INTR_EARLY** – Executed when the BLESS connection engine in Slave mode detects a BLE packet that matches its access address.
4. **CY_BLE_INTR_CALLBACK_BLELL_CONN_INTR_CE_RX** – Executed when the BLESS connection engine receives a non-empty packet from the peer device
5. **CY_BLE_INTR_CALLBACK_BLELL_CONN_INTR_CE_TX_ACK** – Executed when the BLESS connection engine receives an ACK packet from the peer device for the previously transmitted packet.
6. **CY_BLE_INTR_CALLBACK_BLELL_CONN_INTR_CLOSE_CE** – Executed when the BLESS connection engine closes the connection event. This interrupt is executed on every connection interval for connection, irrespective of data tx/rx state.

7. **CY_BLE_INTR_CALLBACK_BLESS_INTR_STAT_DSM_ENTERED** – Executed when the BLESS enters Deep Sleep mode. A user call to the Cy_SysPm_DeepSleep() function will trigger the BLESS Deep Sleep entry sequence. A time instance when each of these interrupts (#1 to #7) are triggered in a connection event are shown below (green).



8. **CY_BLE_INTR_CALLBACK_BLELL_SCAN_INTR_ADV_RX** – Executed when the BLESS scan engine receives an advertisement packet from the peer device.
9. **CY_BLE_INTR_CALLBACK_BLELL_SCAN_INTR_SCAN_RSP_RX** – Executed when the BLESS scan engine receives a scan response packet from the peer device in response to a scan request from the scanner.
10. **CY_BLE_INTR_CALLBACK_BLELL_ADV_INTR_CONN_REQ_RX** – Executed when the BLESS advertisement engine receives a connection request from the peer central device.

An application can use these interrupt callbacks to know when the RF activity is about to begin/end or when the BLE device changes its state from advertisement to connected, or when the BLESS transitions between active and low-power modes, etc. These BLESS real-tie states can be used to synchronize an application with the BLESS or prevent radio interference with other peripherals, etc.

This feature is enabled via define CY_BLE_INTR_NOTIFY_FEATURE_ENABLE in *cy_ble_config.h*.

```
#define CY_BLE_INTR_NOTIFY_FEATURE_ENABLE (1u) /* 1u - Enable / 0u – Disable */
```

BLE Dual mode requires an additional define IPC channel and IPC Interrupt structure to send notifications from the controller core to the host core. Use the following defines:

```
#define CY_BLE_INTR_NOTIFY_IPC_CHAN       (15u) /* valid range:9..15, default:15 */
#define CY_BLE_INTR_NOTIFY_IPC_INTR       (15u) /* valid range:9..15, default:15 */
#define CY_BLE_INTR_NOTIFY_IPC_INTR_PRIOR (1u)  /* valid range:0..7,  default:1 */
```

Unsupported Features

The BLE_PDL Component stack does not support the following optional Bluetooth v5.0 protocol features, as listed in Vol 6, Part B, section 4.6 of the specification:

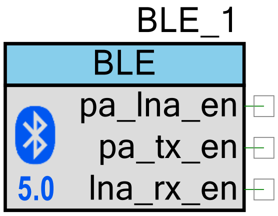
- Connection Parameters Request Procedure (Vol 6, Part B, section 4.6.2)
- Extended Reject Indication (Vol 6, Part B, section 4.6.3)
- Slave-initiated Features Exchange (Vol 6, Part B, section 4.6.4)
- Stable Modulation Index - Transmitter (Vol 6, Part B, section 4.6.10)
- Stable Modulation Index - Receiver (Vol 6, Part B, section 4.6.11)
- LE Extended Advertising (Vol 6, Part B, section 4.6.12)
- LE Periodic Advertising (Vol 6, Part B, section 4.6.13)
- Channel Selection Algorithm #2 (Vol 6, Part B, section 4.6.14)
- Minimum Number of Used Channels Procedure (Vol 6, Part B, section 4.6.15)

Low-Power Modes

The BLE middleware automatically registers Sleep and Deep Sleep callback functions. This functions requests the BLE Stack to put Bluetooth Low Energy Sub-System (BLESS) to low-power mode.

Input/Output Connections

This section describes the input and output connections for the BLE. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.



pa_lna_en – Output *

This signal is needed to put the front end to sleep or in standby whenever there is no radio activity. The signal is ON when either PA control or LNA control is ON.

The polarity of this signal is configurable and can be set in the EXT_PA_LNA_CTRL register by Cy_BLE_ConfigureExtPA() API.

This output is visible if the **Enable external Power Amplifier (PA)** or **Low Noise Amplifier (LNA) chip enable control** parameter is selected on the **Advanced** tab.

pa_tx_en – Output *

This signal is turned ON during transmission and turned OFF when not transmitting. This signal is active a little earlier than the actual start of transmission to allow for the time it takes for the Power amplifier to ramp up. This delay can be set in the EXT_PA_LNA_DLY_CNFG register.

The polarity of this signal is configurable and can be set in the EXT_PA_LNA_CTRL register by Cy_BLE_ConfigureExtPA() API.

This output is visible if the **Enable external PA Tx control output** parameter is selected on the **Advanced** tab.

lna_rx_en – Output *

This signal is needed to choose between the bypass path and the LNA path. This signal is ON during reception and OFF when the receiver is OFF.

The polarity of this signal is configurable and can be set in the EXT_PA_LNA_CTRL register by Cy_BLE_ConfigureExtPA() API.

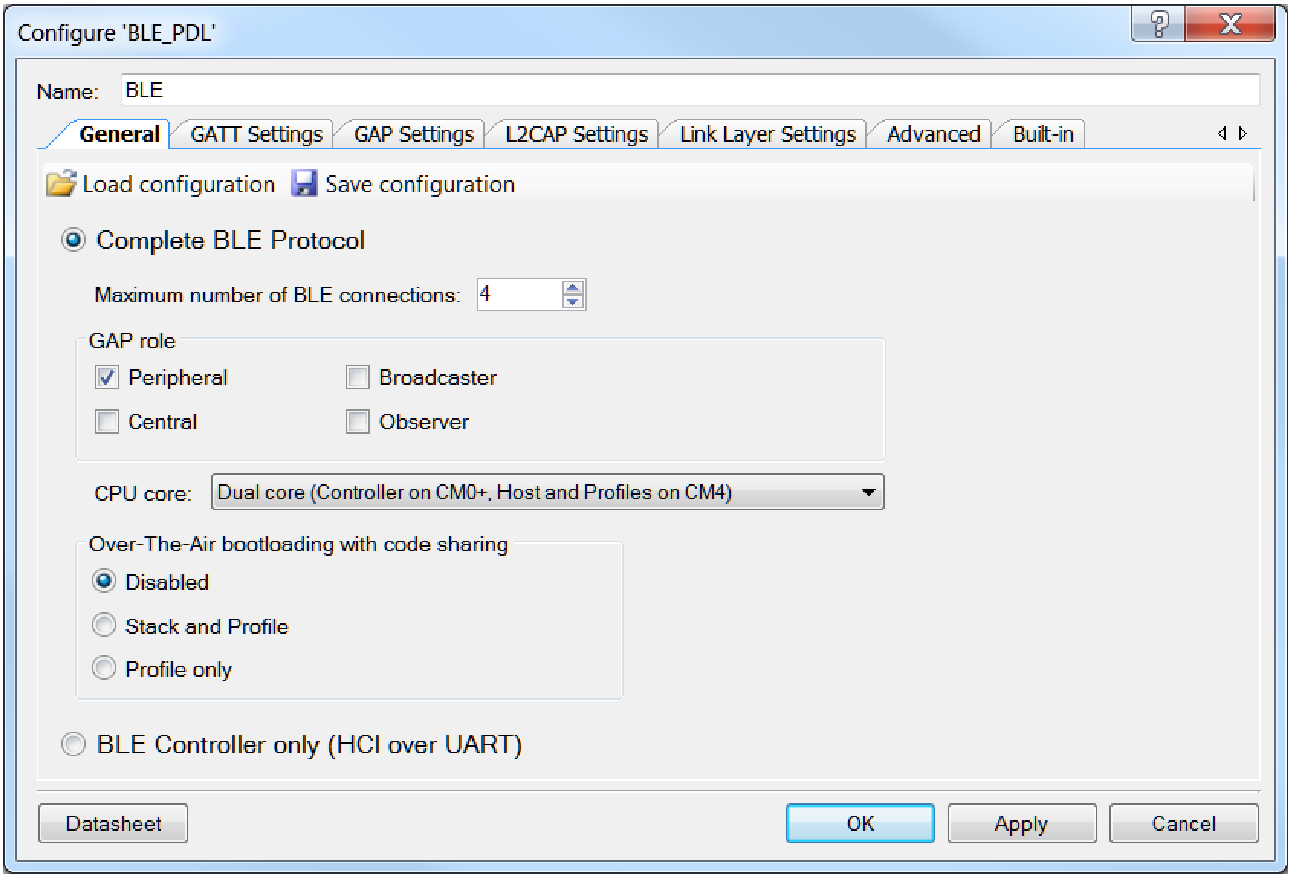
This output is visible if the **Enable external LNA Rx control output** parameter is selected on the **Advanced** tab.

Component Parameters

Drag a BLE_PDL Component onto your design and double-click it to open the Configure dialog. This dialog has the following tabs with different parameters.

General Tab

The **General** tab allows general configuration of the BLE_PDL Component. This tab contains tools to load and save configurations as also three main areas for the type of configuration.



Name: BLE

● Complete BLE Protocol

Maximum number of BLE connections: 4

GAP role

☑ Peripheral □ Broadcaster

□ Central □ Observer

CPU core: Dual core (Controller on CM0+, Host and Profiles on CM4)

Over-The-Air bootloading with code sharing

● Disabled

○ Stack and Profile

○ Profile only

○ BLE Controller only (HCI over UART)

Load Configuration/Save Configuration

Use the **Load Configuration** button to load the previously saved xml Component configuration; use the Save Configuration button to save the current configuration for use in other designs. It is possible to import and export the customizer configuration in xml format.

Note In order to load or save a Profile in the **Bluetooth Developer Studio** compliant format, use **Load BDS Profile** and **Save Profile in BDS format** toolbar commands on the **GATT Settings** tab.

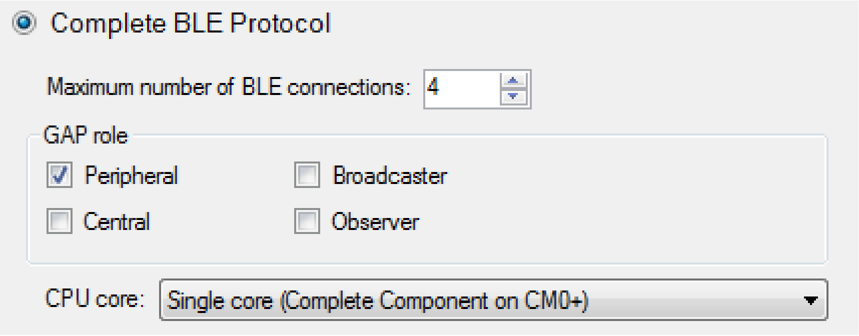
Mode Selection

On the main part of this tab, there are two options to select a mode:

- Complete BLE Protocol
- BLE Controller Only (HCI over UART)

General Tab – Complete BLE Protocol

The Complete BLE Protocol mode enables both BLE Host and Controller. All GAP roles are exposed for configuration.


● Complete BLE Protocol

Maximum number of BLE connections: 4

GAP role

☑ Peripheral ☐ Broadcaster

☐ Central ☐ Observer

CPU core: Single core (Complete Component on CM0+)

Maximum number of BLE connections

This parameter displays how many BLE connections (both Central and Peripheral) are allowed. Valid range is from 1 to 4. Refer to the Multi Connection Support section for more details.

Gap Role

The **GAP role** parameter can take the following values:

- **Peripheral** – Defines a device that advertises using connectable advertising packets and so becomes a slave once connected. Peripheral devices need a Central device, as the Central device initiates connections. Through the advertisement data, a Peripheral device can broadcast the general information about a device.
- **Central** – Defines a device that initiates connections to peripherals and will therefore become a master when connected. Peripheral devices need a Central device, as the Central device initiates connections.
- **Broadcaster** – Similar to the Peripheral role, the device sends advertising data. However Broadcaster does not support connections and can only send data but not receive them.
- **Observer** – When in this role, the device scans for Broadcasters and reports the received information to an application. The Observer role does not allow transmissions.

CPU Core

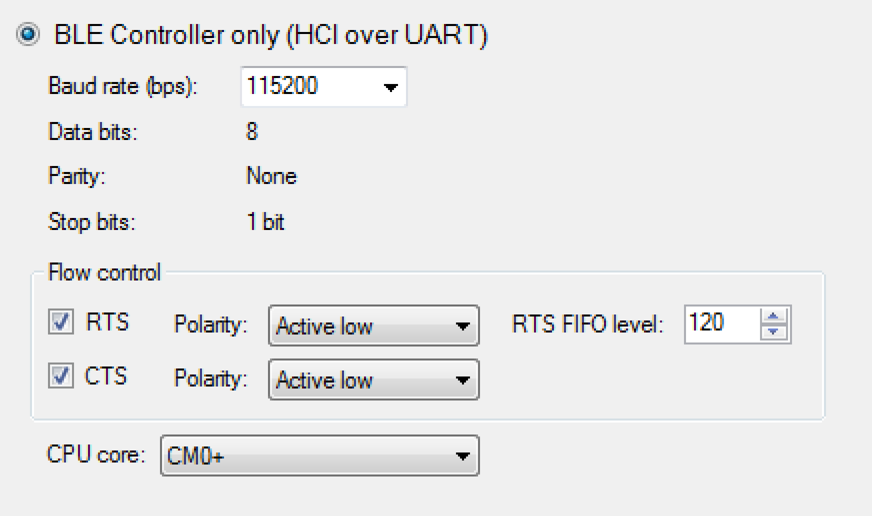
The **CPU Core** parameter defines the cores usage. It can take the following values:

- **Single core (Complete Component on CM0+)** – Only CM0+ core will be used.
- **Single core (Complete Component on CM4)** – Only CM4 core will be used.

  Note For single core MPNs, only **Single core (Complete Component on CM4)** option is allowed.
- **Dual core (Controller on CM0+, Host and Profiles on CM4)** – Both cores will be used: CM0+ for the Controller and CM4 for the Host and Profiles.

General Tab – BLE Controller only (HCI over UART)

Choosing this configuration places the Component in HCI mode, which enables use of the device as a BLE controller. It also allows communication with a host stack using a Component embedded UART. When choosing this mode, the **GATT Settings** tab, **GAP Settings** tab, and **L2CAP Settings** tab become unavailable.



● BLE Controller only (HCI over UART)

Baud rate (bps): 115200

Data bits: 8

Parity: None

Stop bits: 1 bit

Flow control

☑ RTS Polarity: Active low RTS FIFO level: 120

☑ CTS Polarity: Active low

CPU core: CM0+

The UART is a full-duplex 8 data bit, 1 stop bit, no parity with Flow control interface.

- **Baud rate (bps) –** Configures the UART baud rate.
- **RTS –** This parameter enables the Ready to Send (RTS) output signal. The RTS signal is the part of flow control functionality used by the receiver. As long as the receiver is ready to accept more data, it will keep the RTS signal active. The RTS FIFO level parameter determines if RTS remains active. Default value: true.

- **RTS Polarity –** This parameter defines active polarity of the RTS output signal as Active Low (default) or Active High.
- **RTS FIFO level –** This parameter determines whether the RTS signal remains active. While the RX FIFO has fewer entries than the RTS FIFO level, the RTS signal remains active. Otherwise, the RTS signal becomes inactive. The RTS remains inactive unit data from RX FIFO will be read to match RTS FIFO level. Default value: 120.
- **CTS –** This parameter enables the Clear to Send (CTS) input signal to be routed out to the pin. The CTS signal is the part of flow control functionality used by the transmitter. The transmitter checks whether the CTS signal is active before sending data from the TX FIFO. The transmission of data is suspended if the CTS signal is inactive, and transmission will be resumed when the CTS signal becomes active again. Default value: true.
- **CTS Polarity –** This parameter defines active polarity of the CTS input signal as Active Low (default) or Active High.
- **CPU Core –** This parameter defines the cores usage: CM0+ or CM4.

Over-The-Air bootloading with code sharing

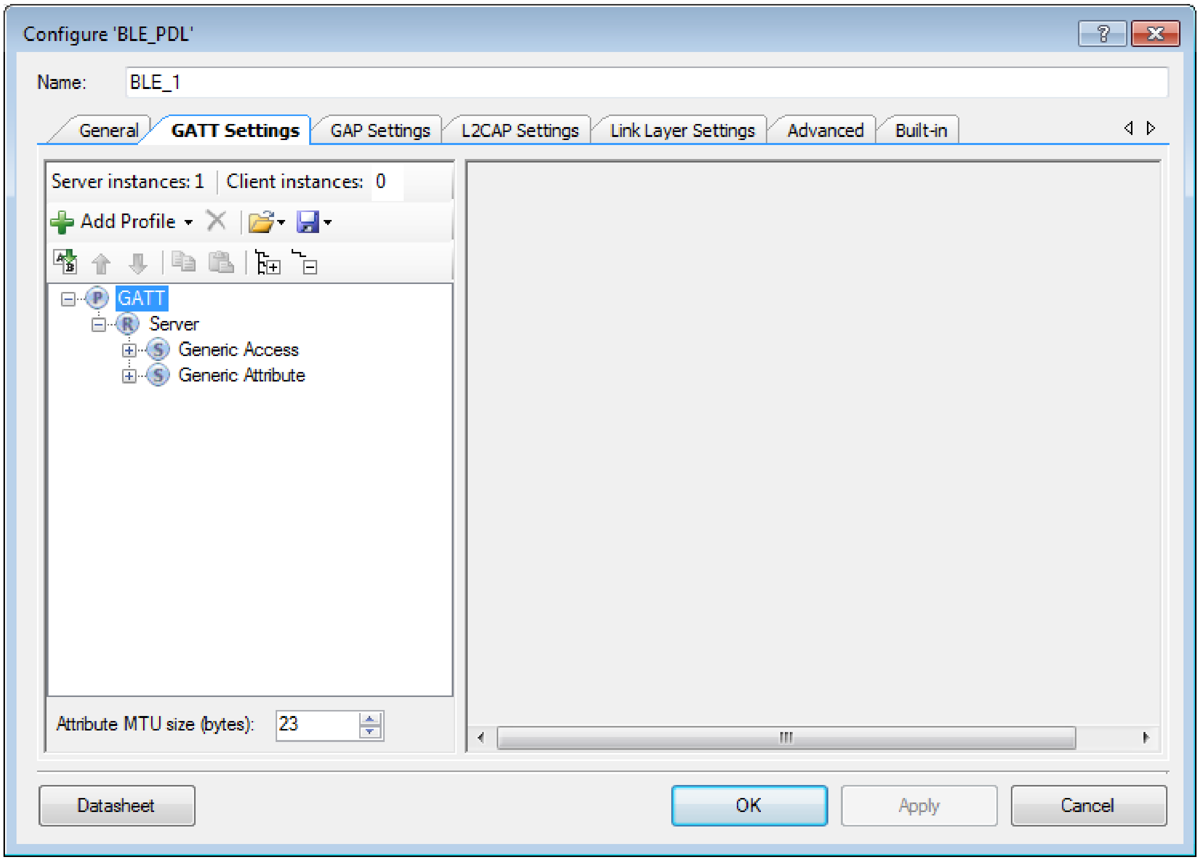
This option is used in the over-the-air (OTA) implementation. It allows you to share the BLE component code between two component instances: one instance with profile-specific code and one with the stack. This parameter allows you to choose between the following options:

- **Disabled** – This option disables the OTA feature.
- **Stack and Profiles** – When this option is selected, the component represents only the stack portion of BLE with mandatory bootloader Service and other user preferred services. This component option is used to isolate the stack and the application Profiles. When this option is selected, the Profiles tab is auto-populated with the BLE bootloader Service and users should be able to add other services on top of the bootloader Service.

  Note This mode requires approximately 3024 additional bytes of heap memory. If there is not enough heap memory, the BLE Component will not work. The Heap size can be modified by editing linker scripts.
- **Profile only** – This option makes the component only have the profile-specific code. Stack is excluded.

GATT Settings Tab

The **GATT Settings** tab is used to configure Profile-specific parameters. It is directly affected by the choice of **Profile** settings set in the **General** tab. The **GATT Settings** tab has 3 areas: toolbars, a Profiles tree, and a parameters configuration section.



Configure 'BLE_PDL'

Name: BLE_1

Server instances: 1 | Client instances: 0

- GATT
  - Server
    - Generic Access
    - Generic Attribute

Attribute MTU size (bytes): 23

Toolbars

The toolbars contain navigation options and a means to add or delete Services, Characteristics, and Descriptors.

- **Server instances –** The number of GATT Server instances. The BLE_PDL Component supports a single instance of a GATT Server (single GATT database). You can add additional Services or complete Profiles to the existing Server Profiles tree to build the GATT database. This single GATT database will be reused across all BLE connections.

  Note The CCCD values for each of the active connections will be unique.

- **Client instances –** The number of GATT Client instances. One GATT Client instance exists per connection. You can configure up to four GATT Client instances. All GATT Client instances have one common Client Profiles tree configuration.

- **Add Profile –** This option is available when the GATT node is highlighted in the Profile tree. It allows adding a whole Profile to the Profiles tree. This option doesn't remove existing Services from the tree. Several Profiles can exist in the tree simultaneously.

- **Add Service –** This option is available when the **Profile Role** is highlighted in the Profile tree. It allows loading of Services in the selected **Profile Role**. In GATT server configuration, this option adds the selected service data to the server GATT database and enables service specific APIs. In GATT client configuration, the data structures for auto discovery of this service is created by the Component. If services that are not populated in the GUI are discovered during auto discovery, the Component ignores those service and the application is responsible for discovering the details of such services. Refer to the Profile section for the available Services.

- **Add Characteristic –** This option is available when a Service is highlighted in the Profile tree. The Characteristic options are unique to each Service and are all loaded automatically when a Service is added to the design. The **Add Characteristic** button can be used to manually add new Characteristics to the Service. All Characteristics for the above mentioned Services plus Custom Characteristic are available for selection.

- **Add Descriptor –** This option is available when a Characteristic is highlighted in the Profile tree. Similar to the Characteristic options, Descriptor options are unique to a Characteristic and are all automatically loaded when a Characteristic is added to the design. For more information about BLE Characteristic Descriptors, refer to developer.bluetooth.org. (**Note** You should be a member of Bluetooth SIG to have full access to this site.)

- **Delete –** Deletes the selected Service, Characteristic, or Descriptor.

- **Load/Save –** Imports/Exports Profiles, Services, Characteristics, and Descriptors as shown in the tree. This functionality is independent of the **Load Configuration/Save Configuration** buttons on the **General** tab. That is, this allows you to customize this tree independent of the general settings. Each exported file type will have its own extension.

  The BLE_PDL Component supports import and export of profiles in the file format of **Bluetooth Developer Studio** tool. Use **Load BDS Profile** command to import the BDS profile and **Save Profile in BDS format** command to export the profile into the BDS file format.

- **Rename –** Renames the selected item in the Profiles tree.

- **Move Up/Down –** Moves the selected item up or down in the Profiles tree.

- **Copy/Paste –** Copies/pastes items in the Profiles tree.

- **Expand All –** Expands all items in the Profiles tree.

- **Collapse all Services –** Collapses all Services in the Profiles tree.

Profiles Tree

The Profiles tree is used to view GATT Services, Characteristics, and Descriptors of the GATT Server and Client roles. By navigating through the tree, you can quickly add, delete, or modify Services, Characteristics, and Descriptors using the toolbar buttons or the context menu. You can configure the parameters by clicking an item on the tree. These parameters will show in the Parameters Configuration section.

Parameters Configuration

The Parameters Configuration section allows you to configure a Profile, Service, or Characteristic by selecting the type of Service or Characteristic in the tree.

Attribute MTU Size

Maximum Transmission Unit size (bytes) of an attribute to be used in the design. Valid range is from 23 to 512 bytes. This value is used to respond to an Exchange MTU request from the GATT Client.

Profiles

You can add a whole Profile to the Profiles tree from a list of supported Profiles. The following Profiles are available for selection:

Alert Notification

This Profile enables a GATT Client device to receive different types of alerts and event information, as well as information on the count of new alerts and unread items, which exist in the GATT Server device.

- **Alert Notification Server** Profile role – Specified as a GATT Server. Requires the following Service: **Alert Notification Service**.
- **Alert Notification Client** Profile role – Specified as a GATT Client.

Refer to the Alert Notification Profile Specification for detailed information about the Alert Notification Profile.

Automation IO

This Profile enables a device to connect and interact with an Automation IO Module (IOM) in order to access digital and analog signals.

- **Automation IO Server** Profile role – Specified as a GATT Server. Requires the following Service: **Automation IO Service**.

- **Automation IO Client** Profile role – Specified as a GATT Client.

Refer to the Automation IO Profile Specification for detailed information about the Automation IO Profile.

Blood Pressure

This Profile enables a device to connect and interact with a Blood Pressure Sensor device for use in consumer and professional health care applications.

- **Blood Pressure Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Blood Pressure Service**, **Device Information Service**.
- **Blood Pressure Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Blood Pressure Service**. Support of **Device Information Service** is optional.

Refer to Blood Pressure Profile Specification for detailed information about the Blood Pressure Profile.

Continuous Glucose Monitoring

This Profile enables a device to connect and interact with a Continuous Glucose Monitoring Sensor device for use in consumer healthcare applications.

- **Continuous Glucose Monitoring Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Continuous Glucose Monitoring Service**, **Device Information Service**. Optionally may include **Bond Management Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Continuous Glucose Monitoring Service**. Support of **Bond Management Service** and **Device Information Service** is optional.

Refer to Continuous Glucose Monitoring Profile Specification for detailed information about the Continuous Glucose Monitoring Profile.

Cycling Power

This Profile enables a Collector device to connect and interact with a Cycling Power Sensor for use in sports and fitness applications.

- **Cycling Power Sensor** Profile role – Specified as a GATT Server. Requires the following Service: **Cycling Power Service**. Optionally may include **Device Information Service** and **Battery Service**.
- **Cycling Power Sensor and Broadcaster** Profile role. Requires the following Service: **Cycling Power Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Cycling Power Service**. Support of **Device Information Service** and **Battery Service** is optional.
- **Cycling Power Observer** Profile role. Can only talk to a device with the **Cycling Power Broadcaster** role.

Refer to Cycling Power Profile Specification for detailed information about the Cycling Power Profile.

Cycling Speed and Cadence

This Profile enables a Collector device to connect and interact with a Cycling Speed and Cadence Sensor for use in sports and fitness applications.

- **Cycling Speed and Cadence Sensor** Profile role – Specified as a GATT Server. Requires the following Service: **Cycling Speed and Cadence Service**. Optionally may include **Device Information Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Cycling Speed and Cadence Service**. Support of **Device Information Service** is optional.

Refer to Cycling Speed and Cadence Profile Specification for detailed information about the Cycling Speed and Cadence Profile.

Environmental Sensing Profile

This Profile enables a Collector device to connect and interact with an Environmental Sensor for use in outdoor activity applications.

- **Environmental Sensor** Profile role – Specified as a GATT Server. Requires the following Service: **Environmental Sensing Service**. Optionally may include **Device Information Service** and **Battery Service**.

- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Environmental Sensing Service**. Support of **Device Information Service** and **Battery Service** is optional.

Refer to Environmental Sensing Profile Specification for detailed information about the Environmental Sensing Profile.

Find Me

The Find Me Profile defines the behavior when a button is pressed on one device to cause an alerting signal on a peer device.

- **Find Me Target** Profile role – Specified as a GATT Server. Requires the following Service: **Immediate Alert Service**.
- **Find Me Locator** Profile role – Specified as a GATT Client. Requires support of the following Service: **Immediate Alert Service**.

Refer to Find Me Profile Specification for detailed information about the Find Me Profile.

Glucose

This Profile enables a device to connect and interact with a Glucose Sensor for use in consumer healthcare applications.

- **Glucose Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Glucose Service**, **Device Information Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Glucose Service**. Support of **Device Information Service** is optional.

Refer to Glucose Profile Specification for detailed information about the Glucose Profile.

Health Thermometer

This Profile enables a Collector device to connect and interact with a Thermometer sensor for use in healthcare applications.

- **Thermometer** Profile role – Specified as a GATT Server. Requires the following Services: **Health Thermometer Service**, **Device Information Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Health Thermometer Service**. Support of **Device Information Service** is optional.

Refer to Health Thermometer Profile Specification for detailed information about the Health Thermometer Profile.

HTTP Proxy

This Service allows a Client device, typically a sensor, to communicate with a Web Server through a gateway device. HTTP Proxy Service is not available in the **Add Profile** drop-down list. It can be added as a separate Service.

Refer to HTTP Proxy Service Specification for detailed information about the HTTP Proxy Service.

Heart Rate

This Profile enables a Collector device to connect and interact with a Heart Rate Sensor for use in fitness applications.

- **Heart Rate Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Heart Rate Service**, **Device Information Service**.
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Heart Rate Service**. Support of **Device Information Service** is optional.

Refer to Heart Rate Profile Specification for detailed information about the Heart Rate Profile.

HID over GATT

This Profile defines how a device with BLE wireless communications can support HID Services over the BLE protocol stack using the Generic Attribute Profile.

- **HID Device** Profile role – Specified as a GATT Server. Requires the following Services: **HID Service**, **Battery Service**, and **Device Information Service**. Optionally may include **Scan Parameters Service** as part of the **Scan Server** role of the **Scan Parameters** Profile. **HID Device** supports multiple instances of **HID Service** and **Battery Service** and may include any other optional Services.
- **Boot Host** Profile role – Specified as a GATT Client. Requires support of the following Service: **HID Service**. Support of **Battery Service** and **Device Information Service** is optional.
- **Report Host** Profile role – Specified as a GATT Client. Requires support of the following Services: **HID Service**, **Battery Service**, **Device Information Service**. Support of **Scan Client** role of the **Scan Parameters** is optional.

- **Report and Boot Host** Profile role – Specified as a GATT Client. Requires support of the following Services: **HID Service**, **Battery Service**, **Device Information Service**. Support of **Scan Client** role of the **Scan Parameters** is optional.

Refer to HID over GATT Profile Specification for detailed information about the HID over GATT Profile.

Indoor Positioning

The Indoor Positioning Service exposes location information to support mobile devices to position themselves in an environment where GNSS signals are not available. For example, in indoor premises. The location information is mainly exposed via advertising and the GATT-based service is primarily intended for configuration.

The Indoor Positioning Service is not available in the Profile drop-down list. It can be added as a separate Service.

Refer to Indoor Positioning Service Specification for detailed information about the Indoor Positioning Service.

Internet Protocol Support

This Profile provides the support of exchanging IPv6 packets between devices over the Bluetooth Low Energy transport. The IPSP defines two roles – Node role and Router role. A device may support both Node role and Router role. A device supporting the Node role is likely to be a sensor or actuator. A device supporting the Router role is likely to be an Access Point (such as home router, mobile phone, or similar).

- **Node** Profile role – Specified as a GATT Server. Requires the following Service: **Internet Protocol Support Service**.

- **Router** Profile role – Specified as a GATT Client. Requires support of the following Services: **Internet Protocol Support Service**.

Refer to Internet Protocol Support Profile Specification for detailed information about IPSP.

Location and Navigation

This Profile enables devices to communicate with a Location and Navigation Sensor for use in outdoor activity applications.

- **Location and Navigation Sensor** Profile role – Specified as a GATT Server. Requires the following Service: **Location and Navigation Service**. Optionally may include **Device Information Service** and **Battery Service**.

- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Location and Navigation Service**. Support of **Device Information Service** and **Battery Service** is optional.

Refer to Location and Navigation Profile Specification for detailed information about the Location and Navigation Profile.

Phone Alert Status

This Profile enables a device to alert its user about the alert status of a phone connected to the device.

- **Phone Alert Server** Profile role – Specified as a GATT Server. Requires the following Services: **Phone Alert Status Service**.
- **Phone Alert Client** Profile role – Specified as a GATT Client. Requires support of the following Service: **Phone Alert Service**.

Refer to Phone Alert Status Profile Specification for detailed information about the Phone Alert Status Profile.

Proximity

The Proximity Profile enables proximity monitoring between two devices.

- **Proximity Reporter** Profile role – Specified as a GATT Server. Requires the following Service: **Link Loss Service**. Optionally may include **Immediate Alert Service** and **Tx Power Service** if both are used. Using only one of the optional Services is not allowed.
- **Proximity Monitor** Profile role – Specified as a GATT Client. Requires support of the following Services: **Link Loss Service**. Support of **Immediate Alert Service** and **Tx Power Service** is optional. Same restrictions apply as to **Proximity Reporter**.

Refer to Proximity Profile Specification for detailed information about the Proximity Profile.

Pulse Oximeter

This Profile enables a device to connect and interact with a Pulse Oximeter device for use in consumer and professional health care applications.

- **Pulse Oximeter Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Pulse Oximeter Service**, **Device Information Service**. Optionally may include Bond Management Service, Current Time Service and Battery Service.

- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Pulse Oximeter** and **Device Information Service.** Support of Bond Management Service, Current Time Service and Battery Service are optional.

Refer to Pulse Oximeter Profile Specification for detailed information about the Pulse Oximeter Profile.

Running Speed and Cadence

This Profile enables a Collector device to connect and interact with a Running Speed and Cadence Sensor for use in sports and fitness applications.

- **Running Speed and Cadence Sensor** Profile role – Specified as a GATT Server. Requires the following Service: **Running Speed and Cadence Service**. Optionally may include **Device Information Service**.

- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Running Speed and Cadence Service**. Support of **Device Information Service** is optional.

Refer to Running Speed and Cadence Profile Specification for detailed information about the Running Speed and Cadence Profile.

Scan Parameters

This Profile defines how a Scan Client device with BLE wireless communications can write its scanning behavior to a Scan Server, and how a Scan Server can request updates of the Scan Client scanning behavior.

- **Scan Server** Profile role – Specified as a GATT Server. Requires the following Service: **Scan Parameters Service**.

- **Scan Client** Profile role – Specified as a GATT Client. Required support of the following Service: **Scan Parameters Service**.

Refer to Scan Parameters Profile Specification for detailed information about the Scan Parameters Profile.

Time

The Time Profile enables the device to get the date, time, time zone, and DST information and control the functions related to time.

- **Time Server** Profile role – Specified as a GATT Server. Requires the following Service: **Current Time Service**. Optionally may include **Next DST Change Service** and **Reference Time Update Service**.

- **Time Client** Profile role – Specified as a GATT Client. Requires support of the following Service: **Current Time Service**. Support of **Next DST Change Service** and **Reference Time Update Service** is optional.

Refer to Time Profile Specification for detailed information about the Time Profile.

Weight Scale

The Weight Scale Profile is used to enable a data collection device to obtain data from a Weight Scale that exposes the Weight Scale Service.

- **Weight Scale** Profile role – Specified as a GATT Server, and may be also a GATT Client.

  Requires the following Services: **Weight Scale Service** and **Device Information Service**.

  Optionally may include: **User Data Service**, **Body Composition Service**, **Battery Service** and **Current Time Service**.

- **Collector** Profile role – Specified as a GATT Client, and may be also a GATT Service.

  Required support of the following Service: **Weight Scale Service** and **Device Information Service**.

  Support of **User Data Service**, **Body Composition Service**, **Battery Service** and **Current Time Service** is optional.

Refer to Weight Scale Profile Specification for detailed information about the Weight Scale Profile.

Wireless Power Transfer

The Wireless Power Transfer Profile (A4WP) enables communication between Power Receiver Unit and Power Transmitter Unit in the Wireless Power Transfer systems.

- **Power Receiver Unit** Profile role – Specified as a GATT Server. Requires the following Service: **Wireless Power Transfer**.

- **Power Transmitter Unit** Profile role – Specified as a GATT Client. Requires support of the following Service: **Wireless Power Transfer**.

Wireless Power Transfer Profile is a custom service defined by the Alliance for Wireless Power (A4WP). Refer to the AirFuel Alliance web site for detailed information about the Wireless Power Transfer Profile.

Bootloader Profile

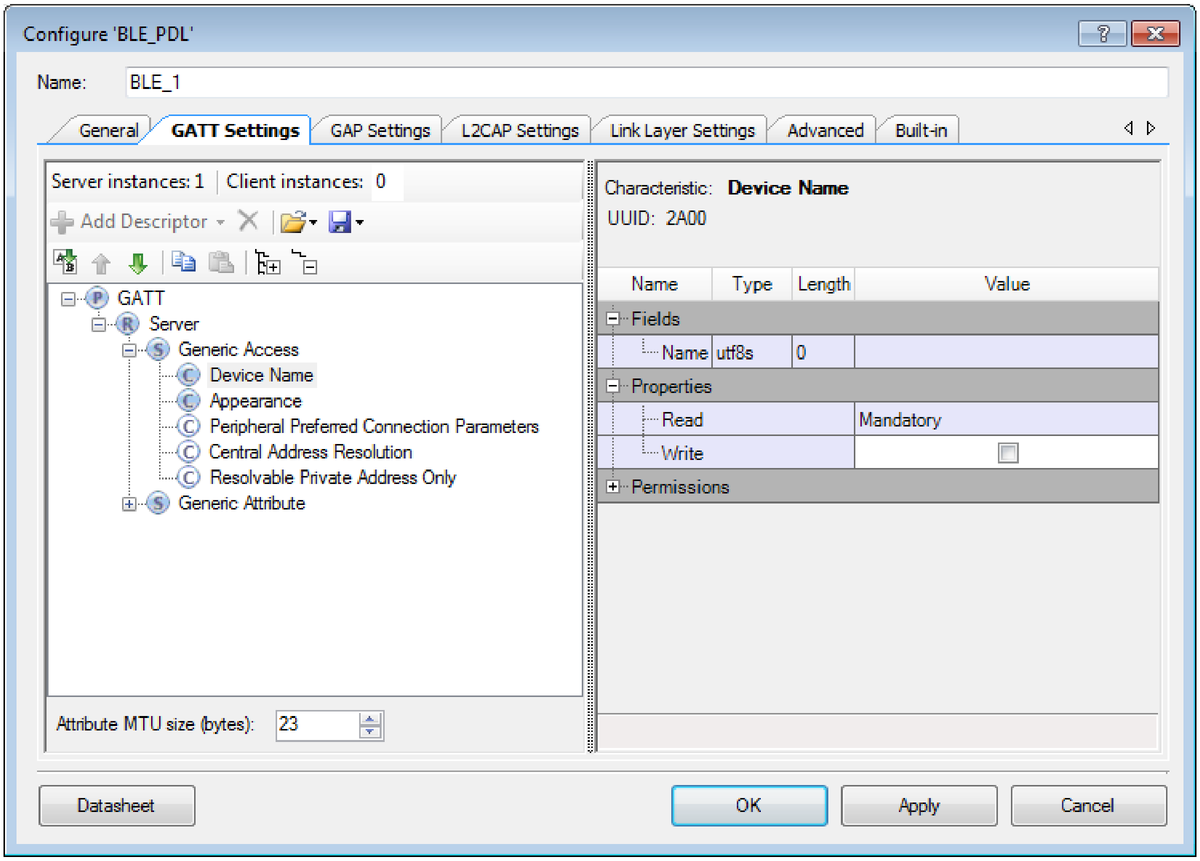
The Component supports the Bootloader Profile and Bootloader Service, which allow a Bootloader Component to update the existing firmware on the Cypress BLE device. The Bootloader Service uses the Bluetooth Low Energy interface as a communication interface. It can be added to any of the profiles if the design requires updating the firmware Over-the-Air (OTA).

Refer to Bootloader Service Configuration section for detailed information about the Bootloader Service.

Notes

- All Profiles must have a **Generic Access Service** and a **Generic Attribute Service**.
- The Service Characteristics are configurable only if they belong to a GATT Server node.
- The security settings located in the **GAP Settings** tab are applied globally. In addition to this, you may manually configure the security of each Characteristic/Descriptor.
- Tree node icons may have two colors: blue and white. Blue color indicates that a node is mandatory and cannot be deleted. White color indicates that a node is optional.

Generic Access Service



Name: BLE_1

Server instances: 1 | Client instances: 0

Characteristic: **Device Name**

UUID: 2A00

Name	Type	Length	Value
Fields			
Name	utf8s	0	
Properties			
Read			Mandatory
Write			☐
Permissions			

Attribute MTU size (bytes): 23

This Service is used to define the basic Bluetooth connection and discovery parameters. Click on the Characteristic under the **Generic Access Service** to view that particular Characteristic settings. You perform the actual Characteristics configuration in the **General** options located in the **GAP Settings** tab.

- **Device Name**: This is the name of your device. It has a read (without authentication/authorization) property associated with it by default. This parameter can be up to 248 bytes. The value comes from the **Device Name** field on the GAP Settings tab, under General.
- **Appearance**: The device's logo or appearance, which is a SIG defined 2-byte value. It has a read (without authentication/authorization) property associated with it by default. The value comes from the **Appearance** field on the GAP Settings tab, under General.

- **Peripheral Preferred Connection**: A device in the peripheral role can convey its preferred connection parameter to the peer device. This parameter is 8 bytes in total and is composed of the following sub-parameters.

  Note This parameter will only be available when the device supports a Peripheral role. Refer to the GAP Settings Tab Peripheral preferred connection parameters section for more information.

  - □ **Minimum Connection Interval:** This is a 2-byte parameter that denotes the minimum permissible connection time.
  - □ **Maximum Connection Interval**: This is a 2-byte parameter that denotes the maximum permissible connection time.
  - □ **Slave Latency**: This is a 2-byte value and defines the latency between consecutive connection events.
  - □ **Connection Supervision Timeout Multiplier**: This is a 2-byte value that denotes the LE link supervision timeout interval. It defines the timeout duration for which an LE link needs to be sustained in case of no response from the peer device over the LE link.

    Note For proper operation, the Connection Supervision Timeout must be larger than **(1 + Slave latency) * Connection Interval * 2** (ms). Refer to Bluetooth Core Specification Volume 6, Part B, Chapter 4.5.2 for more information on Connection Supervision Timeout.

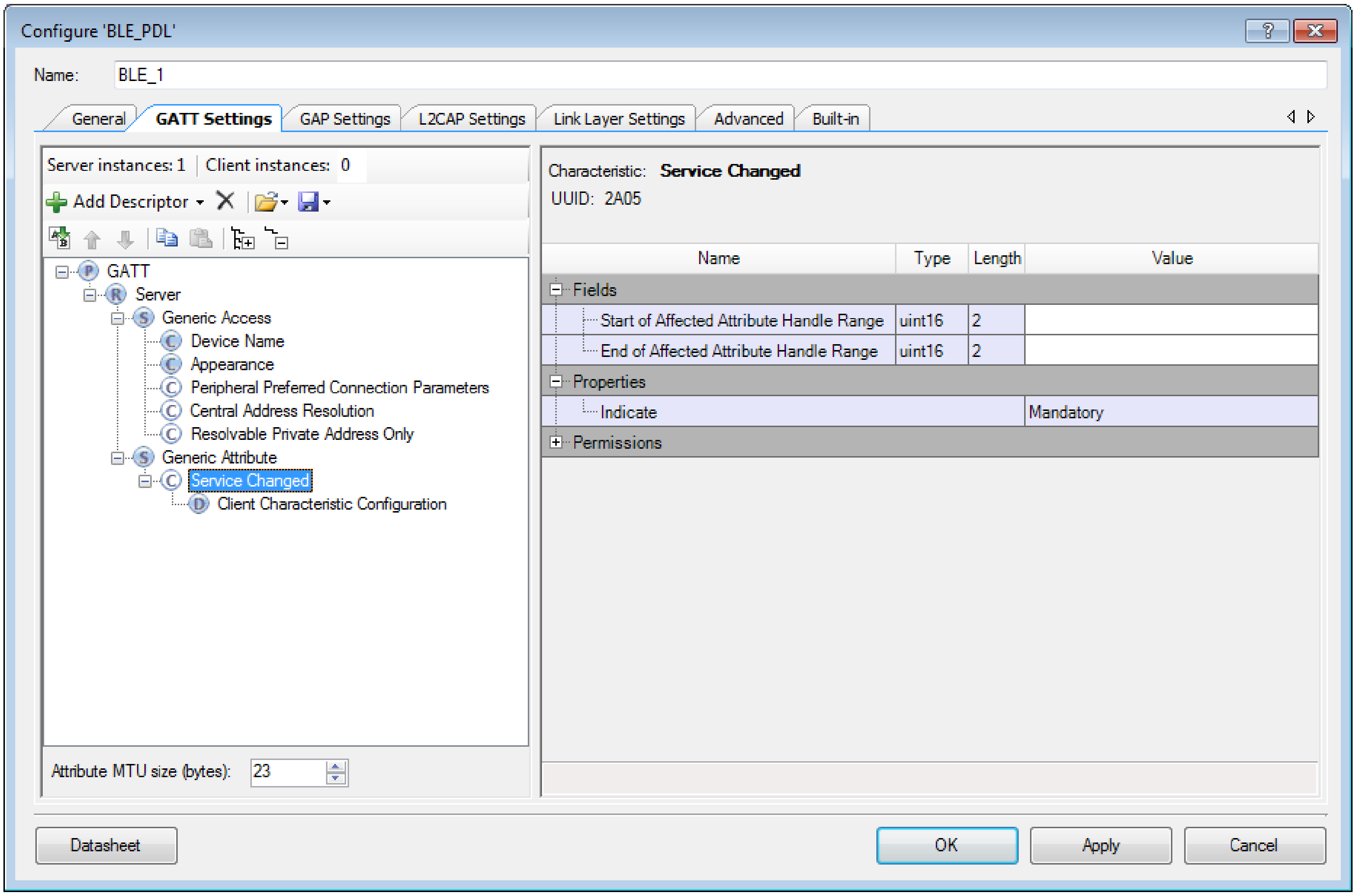
  Note The above parameters are used for connection parameters update procedure over L2CAP if a GAP central device does not use the peripheral preferred connection parameters. For example, iOS7 ignores peripheral preferred connection parameter Characteristics and establishes a connection with a default 30 ms connection interval. The peripheral device should request a connection parameter update by sending an L2CAP connection parameter update request at an appropriate time.

  A typical peripheral implementation should initiate L2CAP connection parameter update procedure once any Characteristic is configured for periodic notification or indication.

- **Central address resolution**: A device in the central role can convey whether it supports privacy with address resolution. The Peripheral shall check if the peer device supports address resolution by reading the Central Address Resolution characteristic before using directed advertisement where the initiator address is set to a Resolvable Private Address (RPA).

- **Resolvable Private Address Only**: Defines whether the device will only use Resolvable Private Addresses (RPAs) as local addresses.

Generic Attribute Service



Characteristic: **Service Changed**
UUID: 2A05

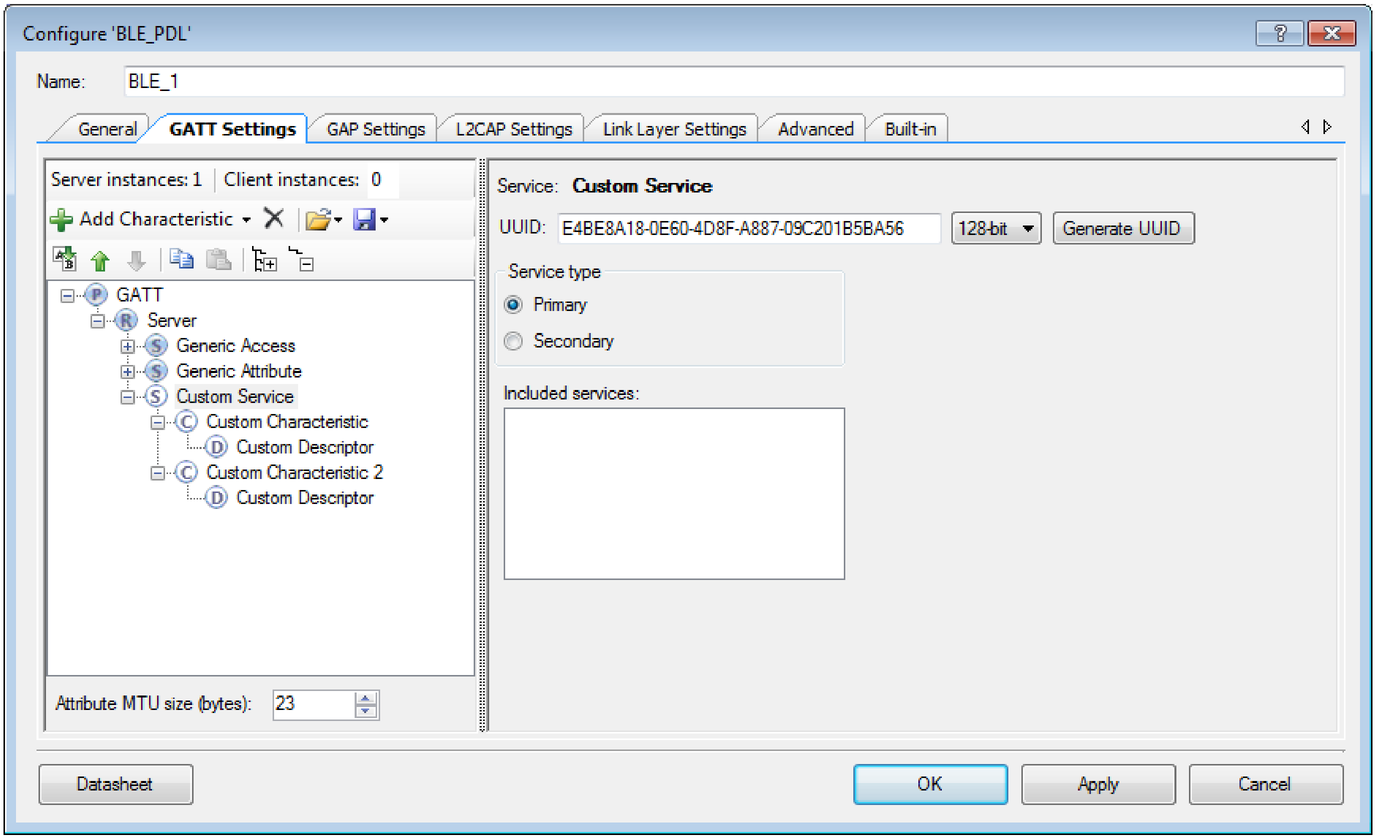
Name	Type	Length	Value
Fields			
Start of Affected Attribute Handle Range	uint16	2	
End of Affected Attribute Handle Range	uint16	2	
Properties			
Indicate			Mandatory
Permissions			

Attribute MTU size (bytes): 23

Click on the Characteristic under the Generic Attribute Service to configure that particular Characteristic.

- **Service Changed** - This Characteristic is used to indicate to the connected devices that a Service has changed (i.e., added, removed, or modified). It is used to indicate to GATT Clients that have a trusted relationship (i.e., bond) with the GATT Server when GATT based Services have changed when they re-connect to the GATT Server. It is mandatory for the device in the GATT Client role. For the device in the GATT Server role, the Characteristic is mandatory if the GATT Server changes the supported Services in the device.

Custom Service Configuration



Name: BLE_1

Server instances: 1 | Client instances: 0

```
GATT
  Server
    Generic Access
    Generic Attribute
    Custom Service
      Custom Characteristic
        Custom Descriptor
      Custom Characteristic 2
        Custom Descriptor
```

Attribute MTU size (bytes): 23

Service: **Custom Service**

UUID: E4BE8A18-0E60-4D8F-A887-09C201B5BA56 128-bit

Service type

● Primary

○ Secondary

Included services:

UUID

A universally unique identifier of the service. This field is editable for Custom Services. Use the **Generate** button to generate a random 128-bit UUID.

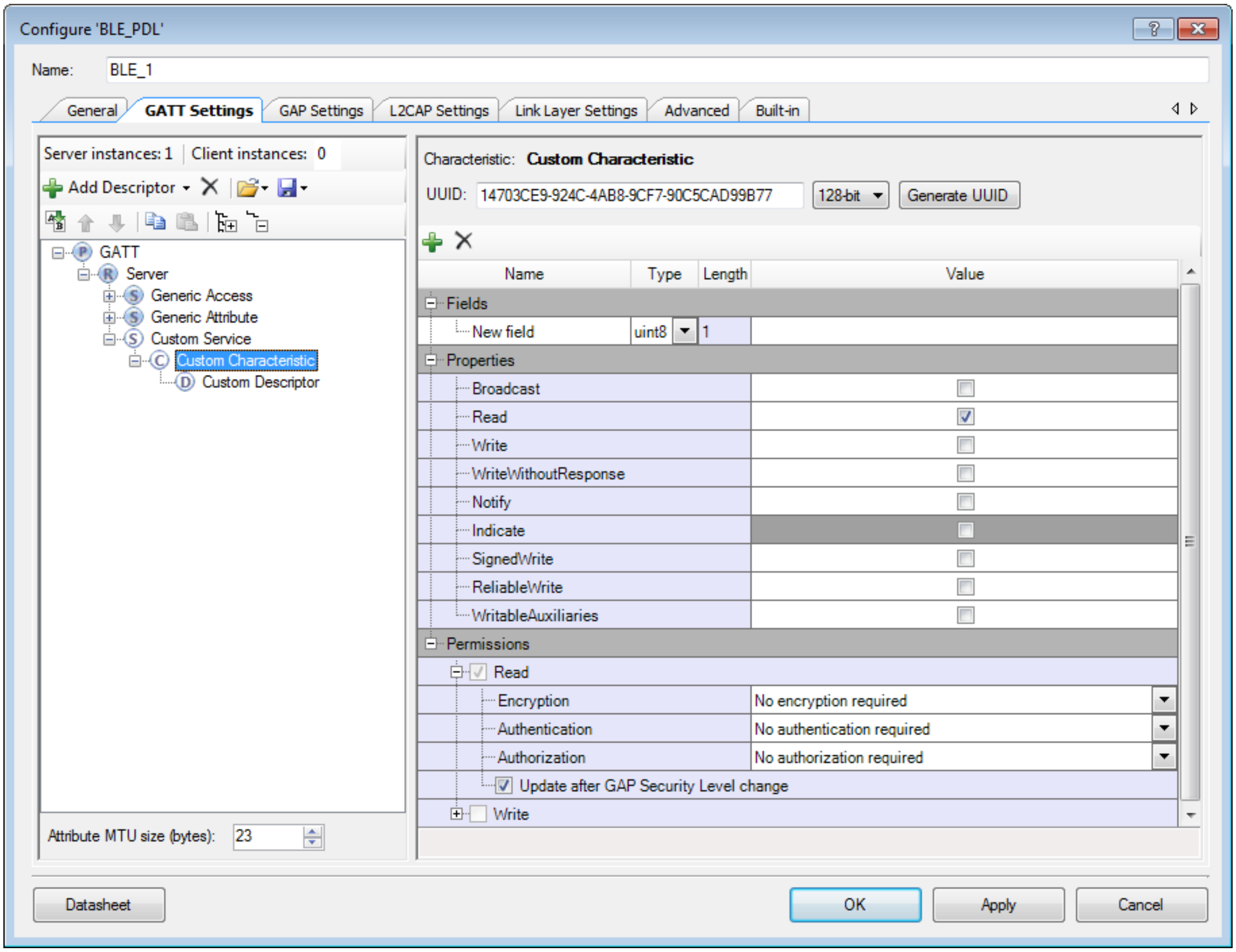
Service type

- **Primary** – Represents the primary functionality of the device.
- **Secondary** – Represents an additional functionality of the device. The secondary service must be included in another service.

Included services

- The list of the Services that can be included in the selected Service. Each Service may have one or more included Services. The included Services provide the additional functionality for the Service.

Custom Characteristic Configuration


Configure 'BLE_PDL'

Name: BLE_1

Server instances: 1 | Client instances: 0

Characteristic: **Custom Characteristic**

UUID: 14703CE9-924C-4AB8-9CF7-90C5CAD99B77 | 128-bit | Generate UUID

Name	Type	Length	Value
Fields			
New field	uint8	1	
Properties			
Broadcast			□
Read			☑
Write			□
WriteWithoutResponse			□
Notify			□
Indicate			□
SignedWrite			□
ReliableWrite			□
WritableAuxiliaries			□
Permissions			
☑ Read			
Encryption			No encryption required
Authentication			No authentication required
Authorization			No authorization required
☑ Update after GAP Security Level change			
□ Write			

Attribute MTU size (bytes): 23

UUID

A universally unique identifier of the Characteristic. This field is editable for Custom Characteristics. Use the **Generate** button to generate a random 128-bit UUID.

Fields

Fields represent a Characteristic value. The default value for each field can be set in the **Value** column. In case of the Custom Characteristic, the fields are customizable.

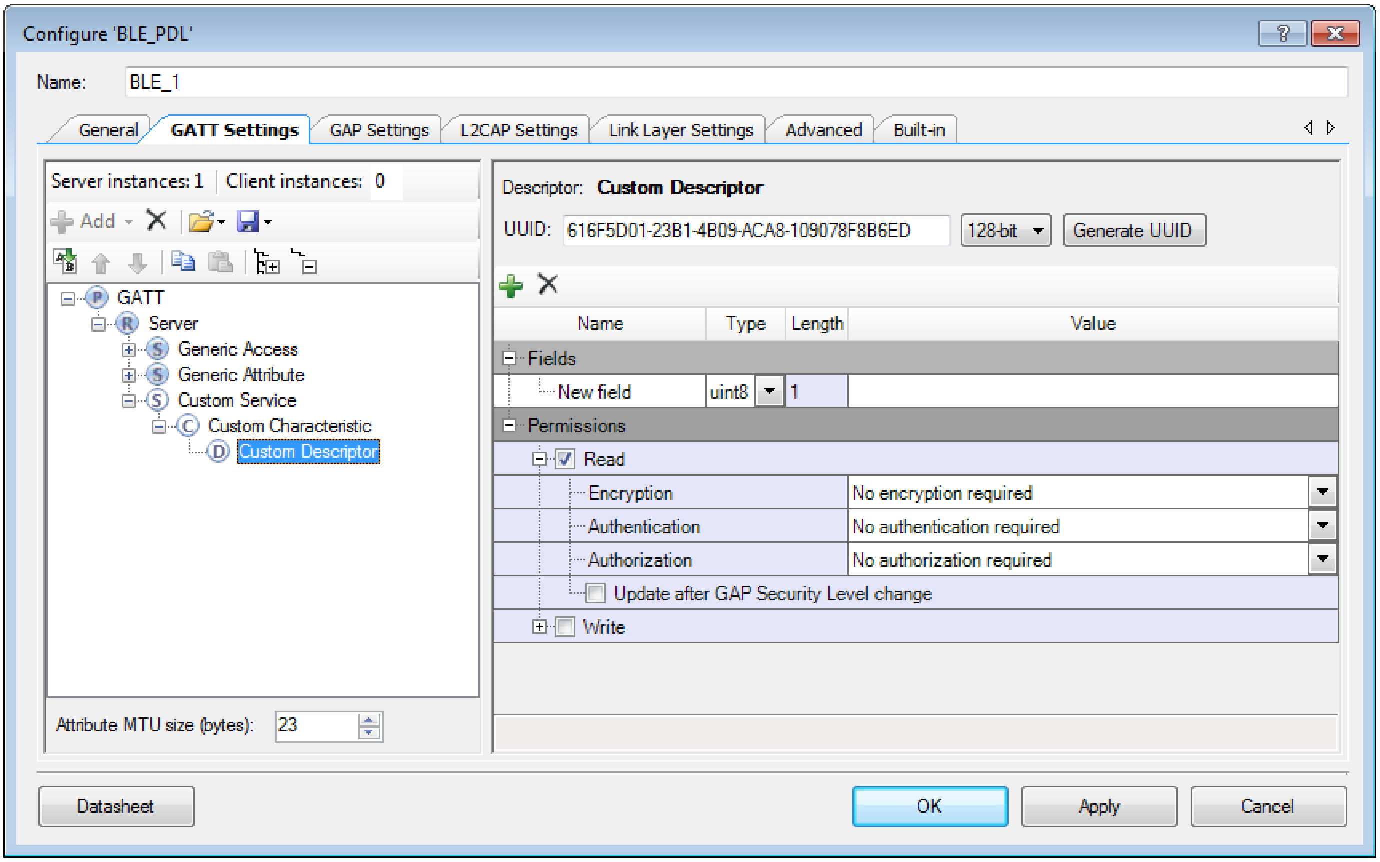
Properties

The Characteristic properties define how the Characteristic value can be used. Some properties (Broadcast, Notify, Indicate, Reliable Write, Writable Auxiliaries) require the presence of a corresponding Characteristic Descriptor. For details, please see Bluetooth Core Specification Vol.3, part G (GATT), section 3.3.1.1 “Characteristic Properties”.

Permissions

Characteristic permissions define how the Characteristic Value attribute can be accessed and the security level required for this access. Access permissions are set based on the Characteristic properties. The **Update after GAP Security Level change** check box determines if the Security permissions are automatically updated when the **Security Mode** or **Security Level** parameters are changed in the Security Configuration 0 on the **GAP Settings** tab. Additional Security configurations don’t affect attribute permissions.

Custom Descriptor Configuration



Name: BLE_1

Server instances: 1 | Client instances: 0

Descriptor: **Custom Descriptor**

UUID: 616F5D01-23B1-4B09-ACA8-109078F8B6ED 128-bit

Name	Type	Length	Value
Fields			
New field	uint8	1	
Permissions			
☑ Read			
Encryption			No encryption required
Authentication			No authentication required
Authorization			No authorization required
□ Update after GAP Security Level change			
□ Write			

Attribute MTU size (bytes): 23

UUID

A universally unique identifier of the Descriptor. This field is editable for Custom Descriptors. Use the **Generate** button to generate a random 128-bit UUID.

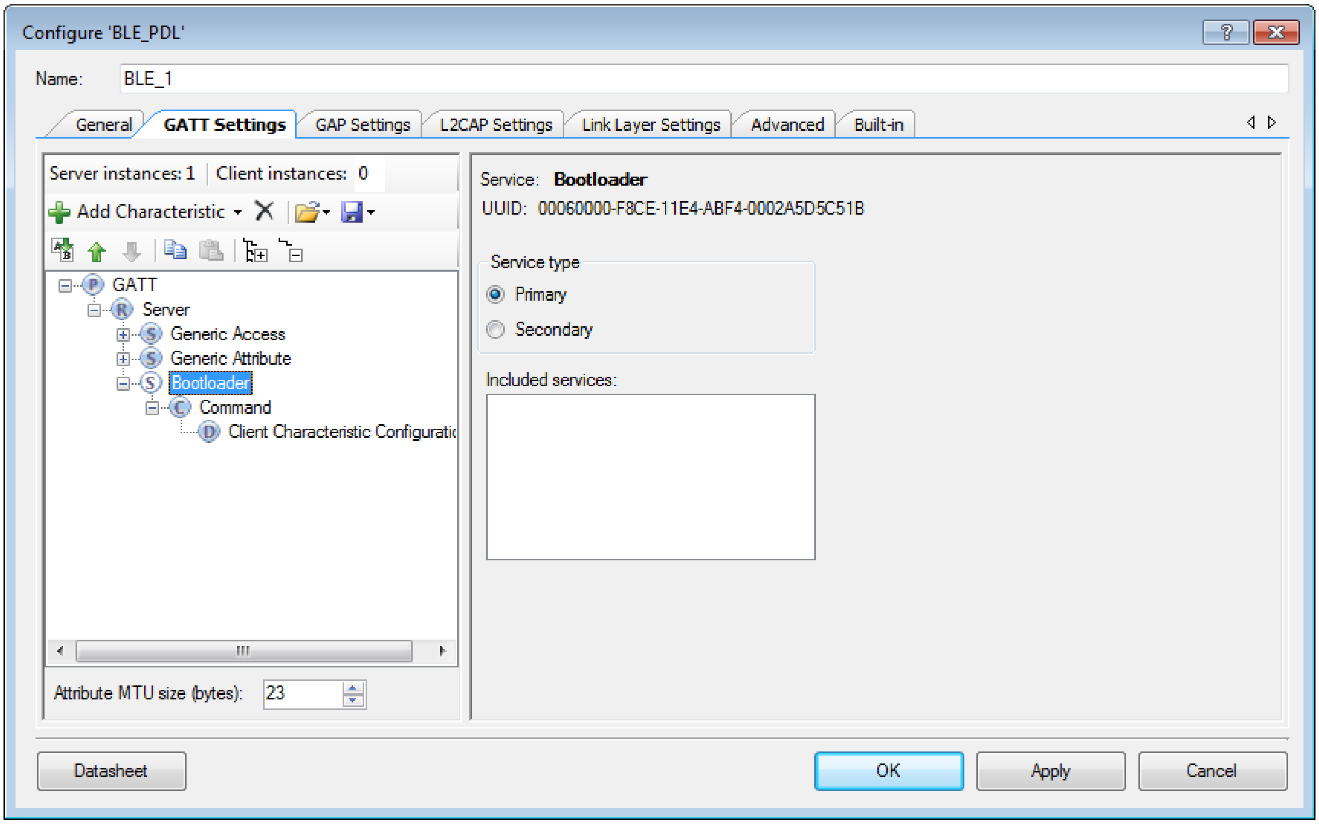
Fields

Fields represent a Descriptor value. The default value for each field can be set in the **Value** column. In case of the Custom Descriptor, the fields are customizable.

Permissions

Descriptor permissions define how the Descriptor attribute can be accessed and the security level required for this access.

Bootloader Service Configuration



Configure 'BLE_PDL'

Name: BLE_1

Server instances: 1 | Client instances: 0

- GATT
  - Server
    - Generic Access
    - Generic Attribute
    - Bootloader
      - Command
        - Client Characteristic Configuratic

Service: **Bootloader**

UUID: 00060000-F8CE-11E4-ABF4-0002A5D5C51B

Service type

● Primary

○ Secondary

Included services:

Attribute MTU size (bytes): 23

UUID

A universally unique identifier of the service. The UUID is set to 00060000-F8CE-11E4-ABF4-0002A5D5C51B.

Service type

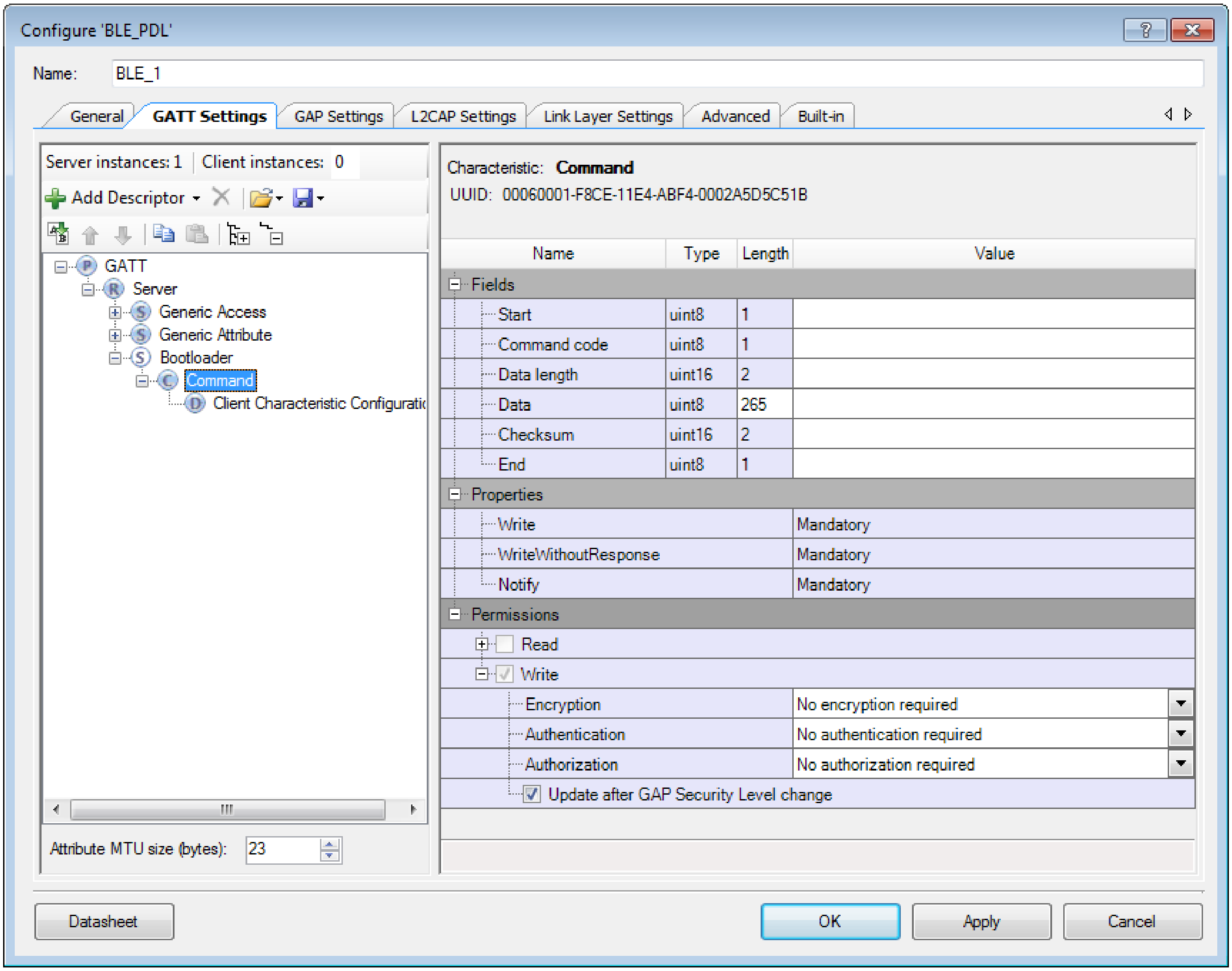
- **Primary** – Represents the primary functionality of the device.

- **Secondary** – Represents additional functionality of the device. The secondary service must be included in another service.

Included services

- The list of the Services that can be included in the selected Service. Each Service may have one or more included Services. The included Services provide the additional functionality for the Service.

Command Characteristic Configuration



Name: BLE_1

Server instances: 1 Client instances: 0

Characteristic: **Command**

UUID: 00060001-F8CE-11E4-ABF4-0002A5D5C51B

Name	Type	Length	Value
⊟ Fields			
Start	uint8	1	
Command code	uint8	1	
Data length	uint16	2	
Data	uint8	265	
Checksum	uint16	2	
End	uint8	1	
⊟ Properties			
Write			Mandatory
WriteWithoutResponse			Mandatory
Notify			Mandatory
⊟ Permissions			
⊞ □ Read			
⊟ ☑ Write			
Encryption			No encryption required
Authentication			No authentication required
Authorization			No authorization required
☑ Update after GAP Security Level change			

Attribute MTU size (bytes): 23

UUID

A universally unique identifier of the Characteristic. The UUID is set to 00060001-F8CE-11E4-ABF4-0002A5D5C51B.

Fields

Fields represent Command Characteristic values, such as the following.

- Start of packet – This constant defines the start of the bootloader packet.
- Command – This field defines the bootloader command. Since the bootloader commands are dependent on the revision of the Cypress Bootloader/Bootloadable Component, refer to the Bootloader/ Bootloadable Component datasheet for the list and description of bootloader commands.
- Status Code – This field defines the status code of the command.
- Data Length – This field defines the length of the bootloader command/response and should be set to the maximum command data length that can be used in the design. The maximum command data length should be obtained from the Bootloader Component datasheet.

  Per the specifics of the BLE protocol, if the command requires a response larger than 20 bytes, the attribute MTU size should be increased. To support the responses with data length set to 56 (response for **Get Metadata** command), the attribute MTU size should be set to 66. This can be seen from the following equation:

  MTU size = Data Length + Bootloader command overhead + notification parameters overhead

  Where:

  - *Data Length* = the response data length
  - *Bootloader command overhead* = 7
  - *Notification parameters overhead* = 3

  Not following this will result in the BLE_PDL Component failing to send a response to the requested command.
- Data – This field defines the bootloader command data. The length of this field is specified by the Data Length field.
- Checksum – This field defines the checksum that is computed for the entire packet with the exception of the Checksum and End of Packet fields.
- End of Packet – This constant defines the end of the bootloader packet.

Properties

The Command Characteristic can be Written or Notified.

Permissions

Characteristic permissions define how the Characteristic Value attribute can be accessed, as well as the security level required for this access. Access permissions are set based on the Characteristic properties. The **Update after GAP Security Level change** check box determines if the Security permissions are automatically updated when the **Security Mode** or **Security Level** parameters are changed on the GAP Settings.

GAP Settings Tab

The GAP parameters define the general connection settings required when connecting Bluetooth devices. It contains various sections of parameters based on the item you select in the tree.

The **GAP Settings** tab displays the settings possible based on the GAP role selected in the **General** tab. This tab allows the default settings of the active tree item to be restored by using the **Restore Defaults** button.

The following sections show the different categories of parameters based on what item you select in the tree.

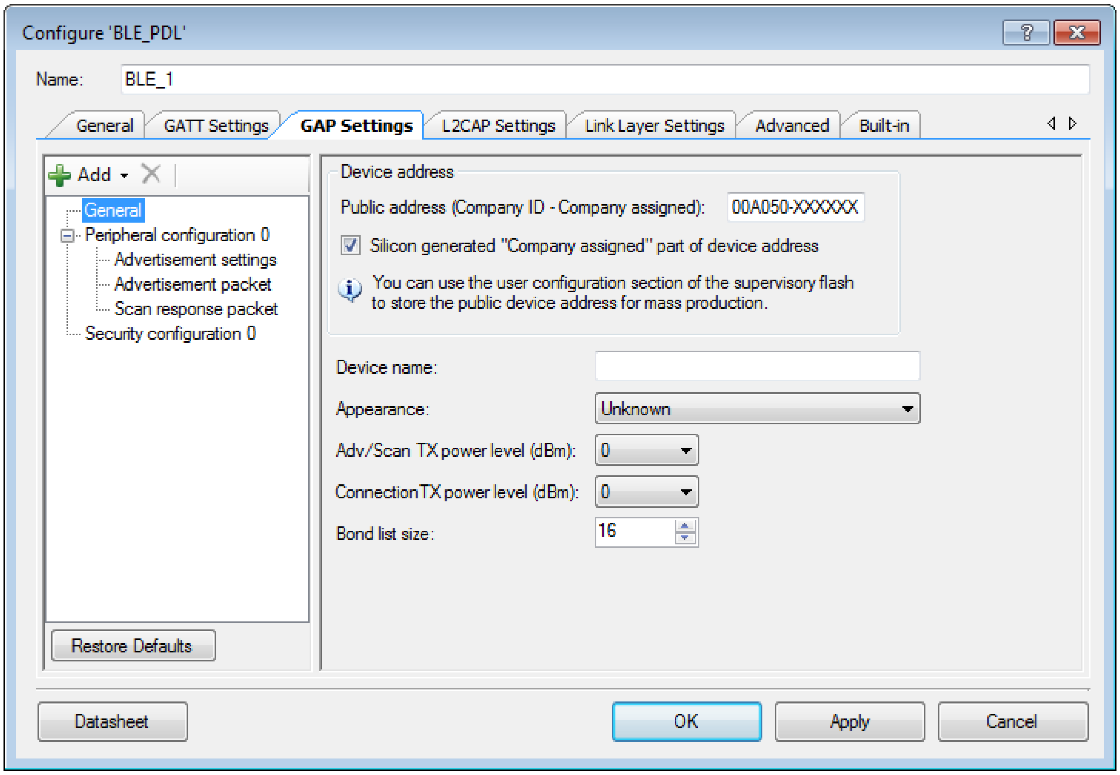
Toolbar

The toolbar contains a means to add or delete GAP role configurations and Security configurations.

- **Add –** allows adding Peripheral, Central, Broadcaster, Observer, or Security configurations. Available options depend on the GAP role selected in the **General** tab. You can add several configurations for one GAP role and switch between them using firmware.
- **Delete –** Deletes the selected Configuration.

GAP Settings Tab – General

This section contains general GAP parameters:



Name: BLE_1

Device address

Public address (Company ID - Company assigned): 00A050-XXXXXX

☑ Silicon generated "Company assigned" part of device address

You can use the user configuration section of the supervisory flash to store the public device address for mass production.

Device name:

Appearance: Unknown

Adv/Scan TX power level (dBm): 0

Connection TX power level (dBm): 0

Bond list size: 16

Public device address (Company ID – Company assigned)

This is a unique 48-bit Bluetooth public address that is used to identify the device. It is divided into the following two parts:

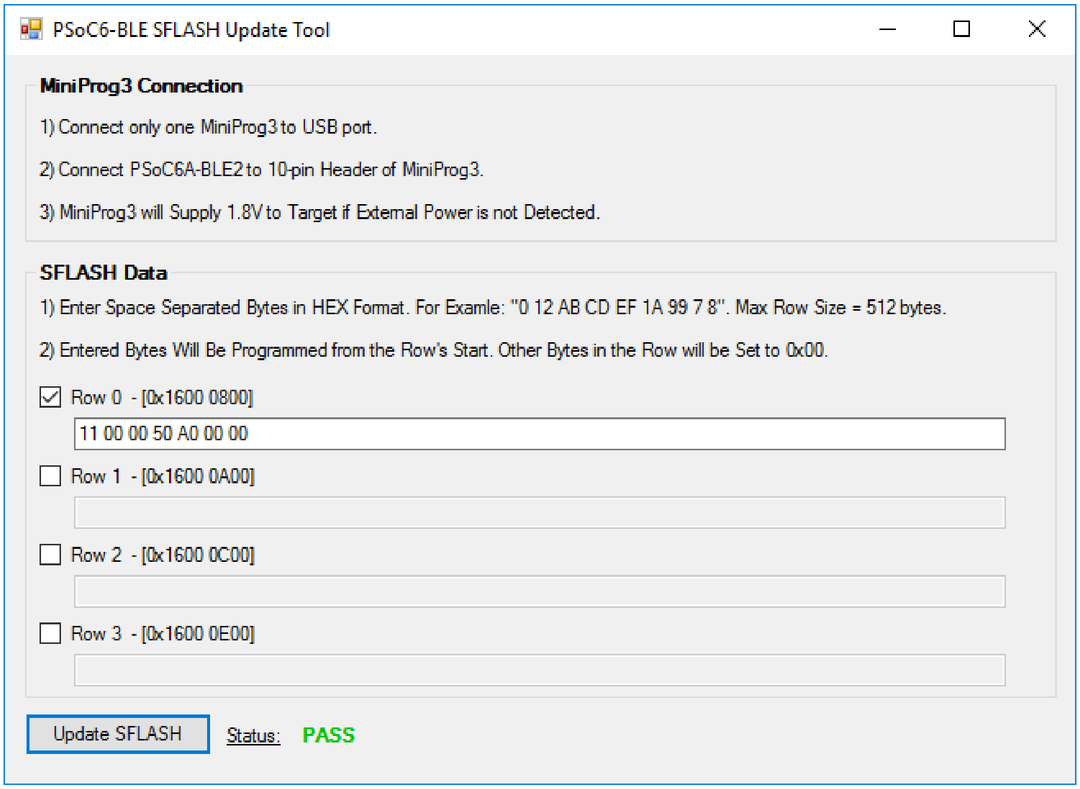
- **“Company ID”** part is contained in the 24 most significant bits. It is a 24-bit Organization Unique Identifier (OUI) address assigned by IEEE.
- **“Company assigned”** part is contained in the 24 least significant bits.

The address configured here is static and is designed to be used for development purposes only. During production, the device address should be programmed into the user’s SFLASH location for the device address (row 0 of user SFLASH) via the SWD interface. Normally this address must be programmed only once during mass production, and then never changed in-field. However, user flash can be reprogrammed in-field many times.

During prototyping (FW design), the device address can be programmed into the user’s SFLASH location using the MiniProg3 and the sample application installed in the following PSoC Programmer folder:

C:\Program Files (x86)\Cypress\Programmer\Examples\Misc\PSoC6-BLE2-SFLASH-Update\Executable

Enter the device address structure of type cy_stc_ble_gap_bd_addr_t in the Row 0 line to store it in the SFLASH.



PSoC6-BLE SFLASH Update Tool

MiniProg3 Connection

1) Connect only one MiniProg3 to USB port.

2) Connect PSoC6A-BLE2 to 10-pin Header of MiniProg3.

3) MiniProg3 will Supply 1.8V to Target if External Power is not Detected.

SFLASH Data

1) Enter Space Separated Bytes in HEX Format. For Examle: "0 12 AB CD EF 1A 99 7 8". Max Row Size = 512 bytes.

2) Entered Bytes Will Be Programmed from the Row's Start. Other Bytes in the Row will be Set to 0x00.

☑ Row 0 - [0x1600 0800]

11 00 00 50 A0 00 00

□ Row 1 - [0x1600 0A00]

□ Row 2 - [0x1600 0C00]

□ Row 3 - [0x1600 0E00]

Status: PASS

Row 1, Row 2 and Row 3 are not used by the Component and available for user information storage.

This application is provided in source code, and can be used as a reference example for implementation in production programmers.

Silicon generated “Company assigned” part of device address

When checked, the “Company assigned” part of the device address is generated using the factory programmed die X/Y location, wafer ID and lot ID of the silicon.

Device Name

The device name to be displayed on the peer side. It has a read (without authentication/authorization) property associated with it by default. This parameter can be up to 248 bytes.

Note This parameter configures the **GAP Service Device Name** Characteristic located in the **Profile Tree**. It is available for modification only when the device is a GATT Server.

Appearance

The device's logo or appearance, which is a SIG defined 2-byte value. It has a read (without authentication/authorization) property associated with it by default.

Note This parameter configures the **GAP Service Appearance** Characteristic located in the **Profile Tree**, It is available for modification only when the device is a GATT Server.

Adv/Scan TX power level

The initial transmitter power level (dBm) of the advertisement or scan channels upon startup. Default: 0 dBm. Possible values: -20 dBm, -16 dBm, -12 dBm, -6 dBm, 0 dBm, 4 dBm.

Connection TX power level

The initial transmitter power level (dBm) of the connection channels upon startup. Default: 0 dBm. Possible values: -20 dBm, -16 dBm, -12 dBm, -6 dBm, 0 dBm, 4 dBm.

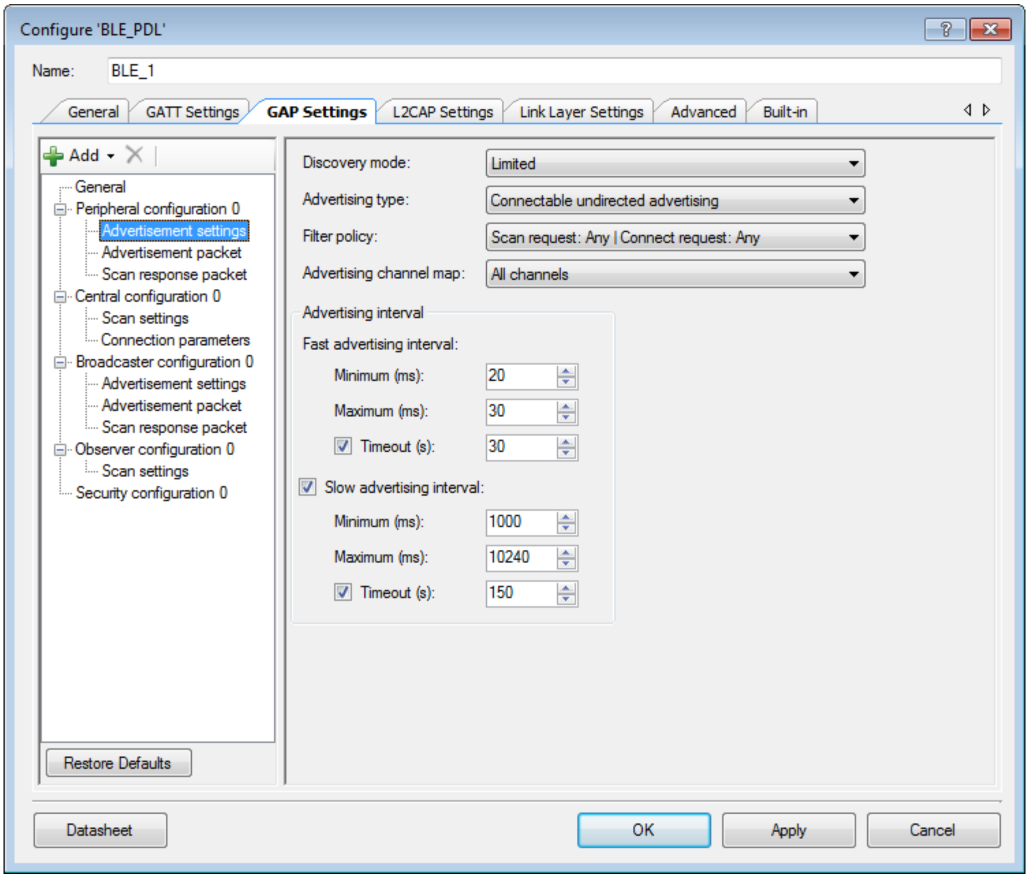
Bond list size

The maximum number of bonded devices supported by this device. Valid range is from 1 to 128. Default: 16.

Note. The maximum number of bonded devices is also limited by available Flash (emulated EEPROM area) size that will be consumed to store data. Consumed Flash size is calculated as multiple of number of supported services and multiple of number of supported bonded devices.

GAP Settings Tab – Advertisement Settings

These parameters are available when the device is configured to contain a Peripheral or Broadcaster **GAP role**.



Name: BLE_1

Discovery mode: Limited

Advertising type: Connectable undirected advertising

Filter policy: Scan request: Any | Connect request: Any

Advertising channel map: All channels

Advertising interval

Fast advertising interval:

Minimum (ms): 20

Maximum (ms): 30

☑ Timeout (s): 30

☑ Slow advertising interval:

Minimum (ms): 1000

Maximum (ms): 10240

☑ Timeout (s): 150

Discovery mode

- **Non-discoverable** – In this mode, the device can't be discovered by a Central device.
- **Limited Discoverable** – This mode is used by devices that need to be discoverable only for a limited period of time, during temporary conditions, or for a specific event. The device which is advertising in Limited Discoverable mode are available for a connection to Central device which performs Limited Discovery procedure. The timeout duration is defined by the applicable advertising timeout parameter.

- **General Discoverable** – In this mode, the device should be used by devices that need to be discoverable continuously or for no specific condition. The device which is advertising in General Discoverable mode are available for a connection to Central device which performs General Discovery procedure.

Advertising type

This parameter defines the advertising type to be used by the LL for an appropriate **Discovery mode**.

- **Connectable undirected advertising** – This option is used for general advertising of the advertising and scan response data. It allows any other device to connect to this device.
- **Scannable undirected advertising** – This option is used to broadcast advertising data and scan response data to active scanners.
- **Non-connectable undirected advertising** – This option is used to just broadcast advertising data.

Filter policy

This parameter defines how the scan and connection requests are filtered.

- **Scan request: Any | Connect request: Any** – Process scan and connect requests from all devices.
- **Scan request: White List | Connect request: Any** – Process scan requests only from devices in the White List and connect requests from all devices.
- **Scan request: Any | Connect request: White List** – Process scan requests from all devices and connect requests only from devices in the White List.
- **Scan request: White List | Connect request: White List** – Process scan and connect requests only from devices in the White List.

Advertising channel map

This parameter is used to enable a specific advertisement channel.

- **Channel 37** – enables advertisement channel #37
- **Channel 38** – enables advertisement channel #38
- **Channel 39** – enables advertisement channel #39
- **Channels 37 and 38** – enables advertisement channels #37 and #38
- **Channel 37 and 39** – enables advertisement channels #37 and #39

- **Channels 38 and 39** – enables advertisement channels #38 and #39
- **All channels** – enables all three advertisement channels

Advertising Interval

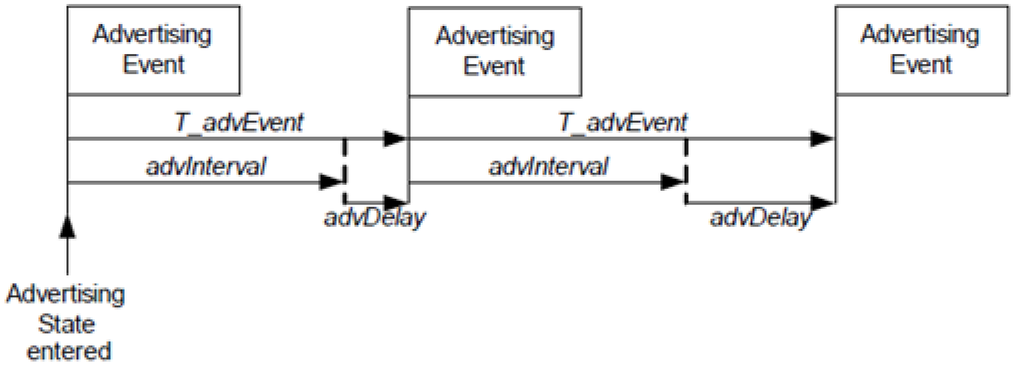
This parameter defines the interval between two advertising events. Set the permissible minimum and maximum values of two Advertisement interval types: **Fast advertising interval** and **Slow advertising interval**. Typically after the device initialization, a peripheral device uses the Fast advertising interval. After the **Fast advertising interval timeout** value expires, and if a connection with a Central device is not established, then the Profile switches to Slow advertising interval to save the battery life. After the **Slow advertising interval timeout** value expires, 'CY_BLE_EVT_GAPP_ADVERTISEMENT_START_STOP' event is generated.

Note The Advertising interval needs to be aligned with the selected Profile specification.

Note In **General Discovery mode**, timeouts are not supported.

- **Fast advertising interval** – This advertisement interval results in faster LE Connection. The BLE_PDL Component uses this interval value when the connection time is between the specified minimum and maximum values of the interval.
    - Minimum: The minimum interval for advertising the data and establishing the LE Connection. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 20 ms to 10240 ms.
    - Maximum: The maximum interval for advertising the data and establishing the LE Connection. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 20 ms to 10240 ms.
    - Timeout: The timeout value of advertising with fast advertising interval parameters. When unchecked, the device is advertising continuously and slow advertising settings become unavailable. The timeout cannot occur before the advertising interval is expired, that is why if a timeout value is less than fast advertising interval minimum value, a warning is displayed. This parameter is not applicable in **General discovery mode**.
- **Slow advertising interval** – Defines the advertising interval for slow advertising. This is an optional parameter which, if enabled, allows to implement advertising with a lower duty cycle to save battery life. The Slow advertising interval parameters are applied to the device after the internal fast advertising interval timeout occurs. The minimum and maximum values defined using this parameter allow the BLE Stack to expect the advertising to happen within these intervals. This parameter is not applicable in **General discovery mode**.
    - Minimum: The minimum interval for advertising the data and establishing the LE Connection. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 1000 ms to 10240 ms.

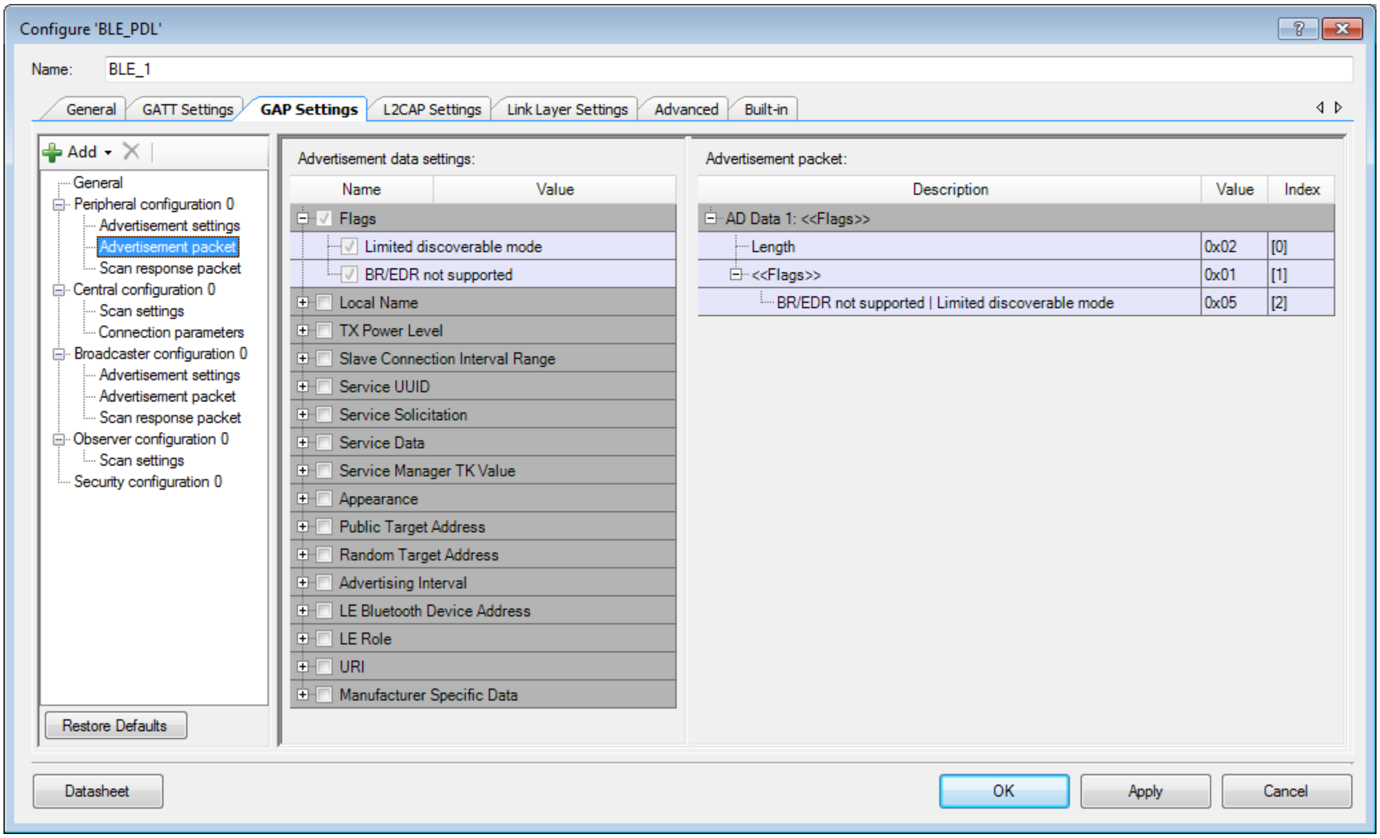
- □ Maximum: The maximum interval for advertising the data and establishing the LE Connection. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 1000 ms to 10240 ms.
- □ Timeout: The timeout value of advertising with slow advertising interval parameters. When unchecked, the device is advertising continuously. The timeout cannot occur before the advertising interval is expired, that is why if a timeout value is less than slow advertising interval minimum value, a warning is displayed.



  - – AdvDelay is a pseudo random delay 0-10 ms.
  - – The complete advertising Event consists of one advertising PDU sent on each of used advertising channels.

GAP Settings Tab – Advertisement packet

This section displays when the device is configured to contain a Peripheral or Broadcaster **GAP role**. It is used to configure the **Advertisement data** to be used in device advertisements.



Name: BLE_1

- General
- Peripheral configuration 0
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
- Central configuration 0
  - Scan settings
  - Connection parameters
- Broadcaster configuration 0
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
- Observer configuration 0
  - Scan settings
- Security configuration 0

Advertisement data settings:

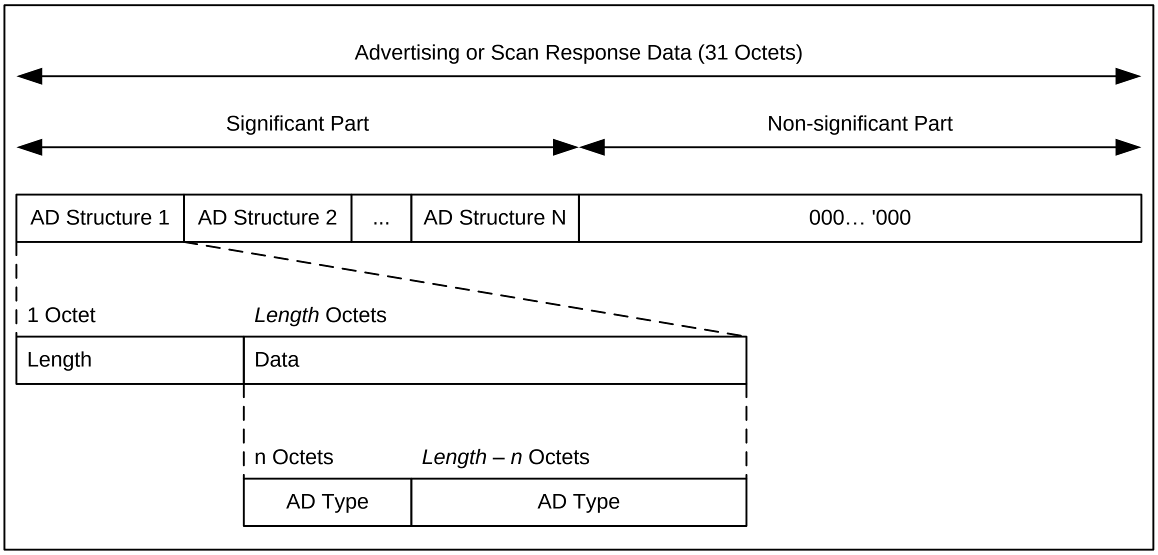
Name	Value
☑ Flags	
☑ Limited discoverable mode	
☑ BR/EDR not supported	
□ Local Name	
□ TX Power Level	
□ Slave Connection Interval Range	
□ Service UUID	
□ Service Solicitation	
□ Service Data	
□ Service Manager TK Value	
□ Appearance	
□ Public Target Address	
□ Random Target Address	
□ Advertising Interval	
□ LE Bluetooth Device Address	
□ LE Role	
□ URI	
□ Manufacturer Specific Data	

Advertisement packet:

Description	Value	Index
AD Data 1: <<Flags>>		
Length	0x02	[0]
<<Flags>>	0x01	[1]
BR/EDR not supported | Limited discoverable mode	0x05	[2]

Advertisement / Scan response data settings

Advertisement (AD) or **Scan response data** packet is a 31 byte payload used to declare the device's BLE capability and its connection parameters. The structure of this data is shown below as specified in the Bluetooth specification.



The data packet can contain a number of AD structures. Each of these structures is composed of the following parameters.

- **AD Length**: Size of the **AD Type** and **AD Data** in bytes.
- **AD Type**: The type of advertisement within the AD structure.
- **AD Data**: Data associated with the **AD Type**.

The total length of a complete Advertising packet cannot exceed 31 bytes.

An example structure for **Advertisement data** or **Scan response data** is as follows.

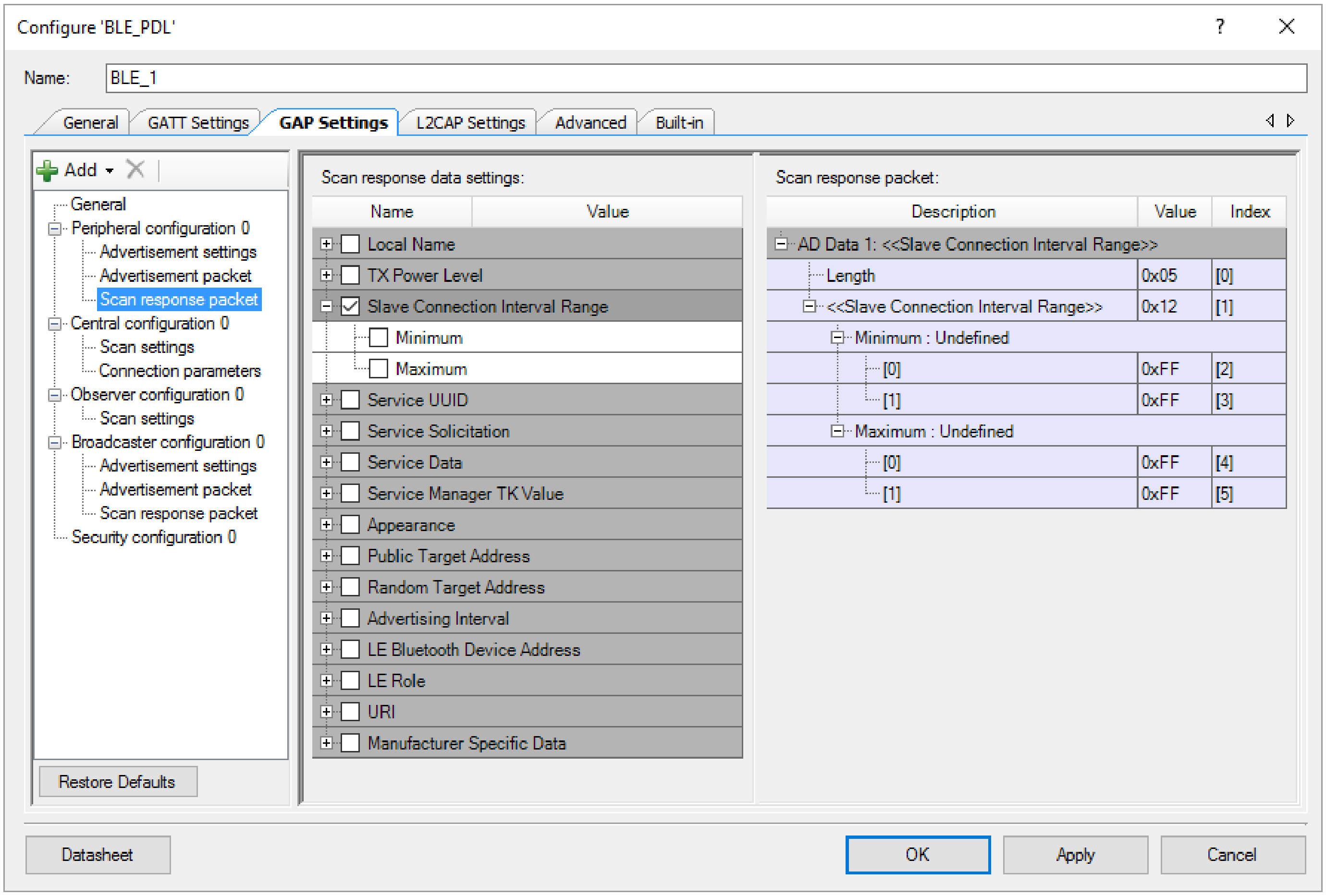
- AD Structure Element Definition:
  - □ **AD Length**: Size of **AD Type** and associated **AD Data** = 5 bytes
  - □ **AD Type** (1 byte): 0x03 (Service UUID)
  - □ **AD Data** (4 bytes): 0x180D, 0x180A (Heart Rate Service, Device Information Service)

The following table shows the **AD Types**.

AD Type	Description
Flags	Flags to broadcast underlying BLE transport capability such as Discoverable mode, LE only, etc.
Local Name	Device Name (complete of shortened). The device name value comes from the **Device name** field on the **GAP Settings** tab, under **General**.
Tx Power Level	Transmit Power Level. Taken from the **Adv/Scan TX power level** field on the **GAP Settings** tab, under **General.**
Slave Connection Interval Range	Preferred connection interval range for the device. Not available in **Broadcaster** GAP role.
Service UUID	List of Service UUIDs to be broadcasted that the device has implemented. There are different AD Type values to advertise 16-bit, 32-bit and 128-bit Service UUIDs. 16-bit and 32-bit Service UUIDs are used if they are assigned by the Bluetooth SIG.
Service Solicitation	List of Service UUIDs from the central device that the peripheral device would like to use. There are different AD Type values to advertise 16-bit, 32-bit and 128-bit Service UUIDs.
Service Data	2/4/16-byte Service UUID, followed by additional Service data.
Security Manager TK value	Temporal key to be used at the time of pairing. Not available in **Broadcaster** GAP role.
Appearance	The external appearance of the device. The value comes from the **Appearance** field on the **GAP Settings** tab, under **General**.
Public Target Address	The public device address of intended recipients.
Random Target Address	The random device address of intended recipients.
Advertising Interval	The Advertising interval value that is calculated as an average of Fast advertising interval minimum and maximum values configured on the **GAP Settings** tab, under **Advertisement Settings**.
LE Bluetooth Device Address	The device address of the local device. The value comes from the **Public device address** field on the **GAP Settings** tab, under **General**.
LE Role	Supported LE roles. Not available in **Broadcaster** GAP role.
URI	URI, as defined in the IETF STD 66.
Manufacturer Specific Data	2 bytes company identifier followed by manufacturer specific data.
Indoor Positioning	The data specified in the Indoor Positioning Service Specification. This is available when the Indoor Positioning Service is present in the Profile.

GAP Settings Tab – Scan response packet

This section displays when the device is configured to contain a Peripheral or Broadcaster **GAP role**. It is used to configure the Scan response data packet to be used in response to device scanning performed by a GATT Client device.



Configure 'BLE_PDL'

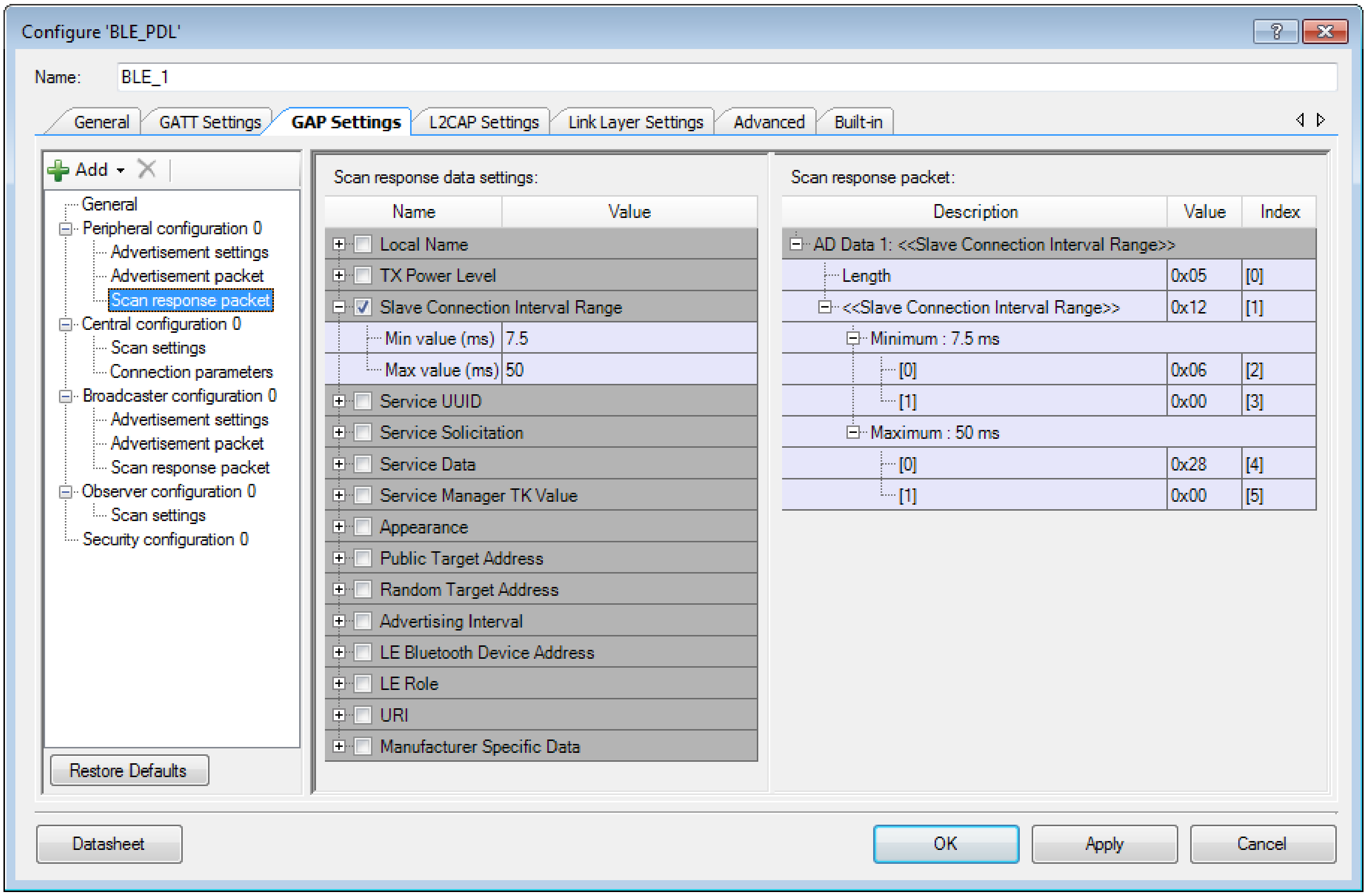
Name: BLE_1

Scan response data settings:

Name	Value
□ Local Name	
□ TX Power Level	
☑ Slave Connection Interval Range	
□ Minimum	
□ Maximum	
□ Service UUID	
□ Service Solicitation	
□ Service Data	
□ Service Manager TK Value	
□ Appearance	
□ Public Target Address	
□ Random Target Address	
□ Advertising Interval	
□ LE Bluetooth Device Address	
□ LE Role	
□ URI	
□ Manufacturer Specific Data	

Scan response packet:

Description	Value	Index
AD Data 1: <<Slave Connection Interval Range>>		
Length	0x05	[0]
<<Slave Connection Interval Range>>	0x12	[1]
Minimum : Undefined		
[0]	0xFF	[2]
[1]	0xFF	[3]
Maximum : Undefined		
[0]	0xFF	[4]
[1]	0xFF	[5]



Name: BLE_1

Scan response data settings:

Name	Value
□ Local Name	
□ TX Power Level	
☑ Slave Connection Interval Range	
Min value (ms)	7.5
Max value (ms)	50
□ Service UUID	
□ Service Solicitation	
□ Service Data	
□ Service Manager TK Value	
□ Appearance	
□ Public Target Address	
□ Random Target Address	
□ Advertising Interval	
□ LE Bluetooth Device Address	
□ LE Role	
□ URI	
□ Manufacturer Specific Data	

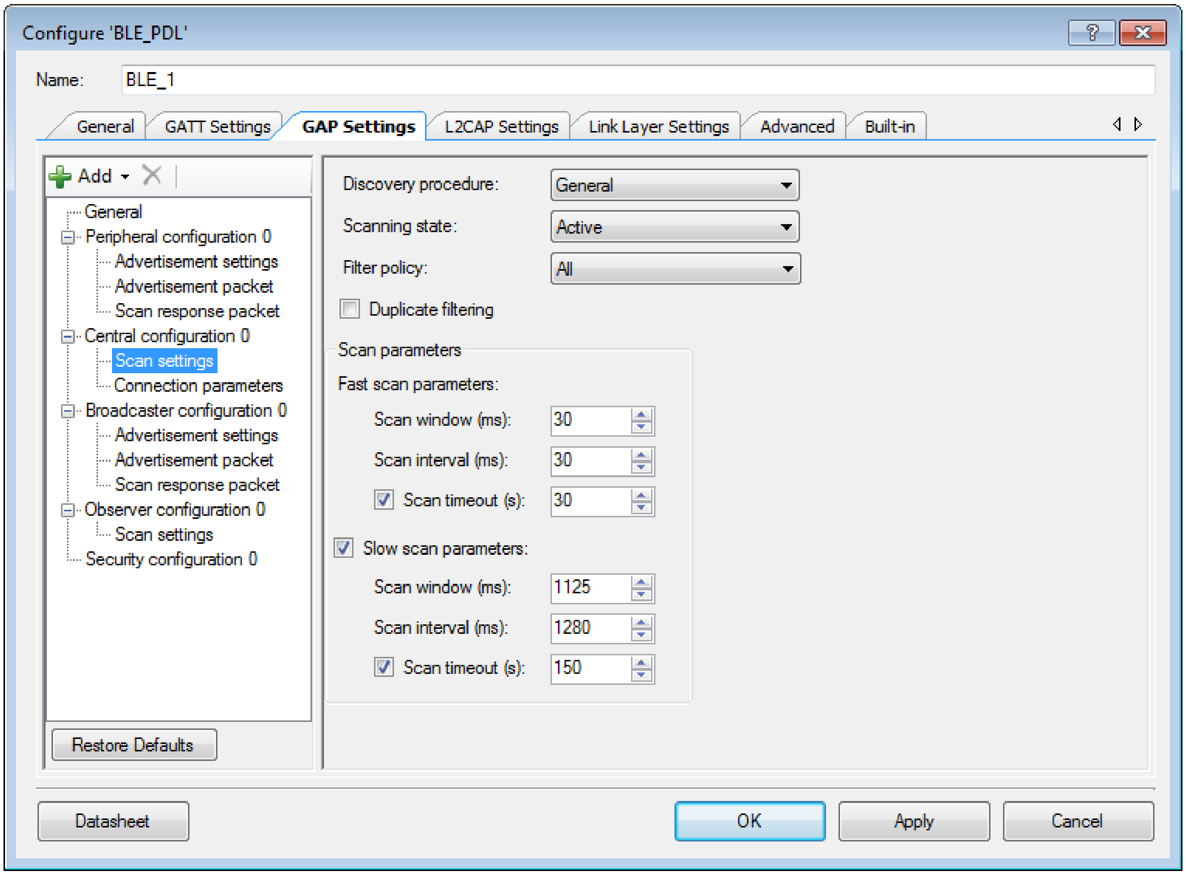
Scan response packet:

Description	Value	Index
AD Data 1: <<Slave Connection Interval Range>>		
Length	0x05	[0]
<<Slave Connection Interval Range>>	0x12	[1]
Minimum : 7.5 ms		
[0]	0x06	[2]
[1]	0x00	[3]
Maximum : 50 ms		
[0]	0x28	[4]
[1]	0x00	[5]

The packet structure of a Scan response packet is the same as an Advertisement packet. See Advertisement / Scan response data settings for information on configuring the Scan response packet.

GAP Settings Tab – Scan settings

These parameters are available when the device is configured to contain Central or Observer **GAP role**. Typically during a device discovery, the GATT Client device initiates the scan procedure. It uses **Fast scan parameters** for a period of time, approximately 30 to 60 seconds, and then it reduces the scan frequency using the **Slow scan parameters**.



Name: BLE_1

Discovery procedure: General

Scanning state: Active

Filter policy: All

□ Duplicate filtering

Scan parameters

Fast scan parameters:

Scan window (ms): 30

Scan interval (ms): 30

☑ Scan timeout (s): 30

☑ Slow scan parameters:

Scan window (ms): 1125

Scan interval (ms): 1280

☑ Scan timeout (s): 150

Note The scan interval needs to be aligned with the user-selected Profile specification.

Discovery procedure

- **Limited –** A device performing this procedure shall discover the device doing limited discovery mode advertising only.
- **General –** A device performing this procedure shall discover the devices doing general and limited discovery advertising.

Scanning state

- **Passive –** In this state a device can only listen to advertisement packets.

- **Active –** In this state a device may ask an advertiser for additional information.

Filter policy

This parameter defines how the advertisement packets are filtered.

- **All** – Process all advertisement packets.
- **White List Only** – Process advertisement packets only from devices in the White List.

Duplicate filtering

When enabled, this activates filtering of duplicated advertisement data. If disabled, the BLE stack will not perform filtering of advertisement data.

Note The controller firmware has 8 address locations reserved to cache the previously seen advertiser devices and filter duplicate packets from them. If there are more than 8 advertising devices in the proximity of a scanner during the scan period, then the address storing buffer is exhausted. The firmware algorithm for overwriting the address cache buffer is implemented in FIFO fashion. When the scanner sees more than 8 advertisers, then 9th advertiser replaces the 1st one, 10th advertiser replaces the 2nd one, and so on in the address cache. After flushing the 1st advertiser from the address cache, if the scanner sees the first advertiser's ADV packet again, then it thinks that it is a new device (as 1st advertiser is no longer in the address cache) resulting in sending the ADV packet to the host.

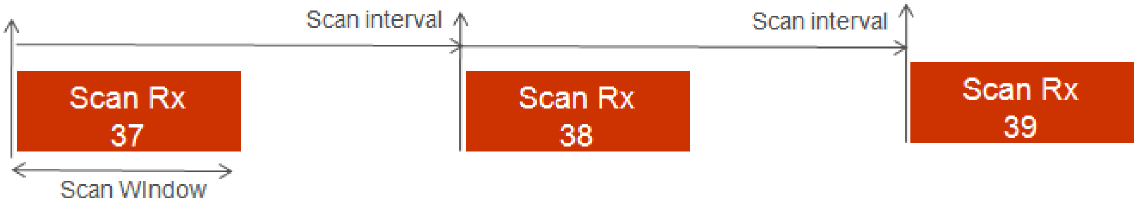
Scan parameters

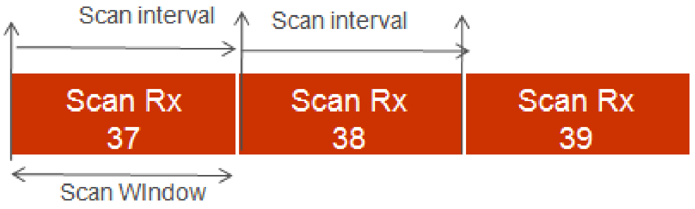
These parameters define the scanning time and interval between scanning events. Two different sets of Scan parameters are used: **Fast scan parameters** and **Slow scan parameters**. Typically, after the device initialization, a central device uses the Fast scan parameters. After the **Fast scan timeout** value expires, and if a connection with a Peripheral device is not established, then the Profile switches to Slow scan parameters to save the battery life. After the **Slow scan timeout** value expires, 'CY_BLE_EVT_GAPC_SCAN_START_STOP ' event is generated. See API documentation.

- **Fast scan parameters** – This connection type results in a faster connection between the GATT Client and Server devices than it is possible using a normal connection.
  - □ **Scan Window**: This parameter defines the scan window when operating in **Fast connection**. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 2.5 ms to 10240 ms. **Scan Window** must be less than the **Scan Interval**. Default: 30 ms.
  - □ **Scan Interval**: This parameter defines the scan interval when operating in **Fast connection**. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 2.5 ms to 10240 ms. Default: 30 ms.

  - □ **Scan Timeout**: The timeout value of scanning with fast scan parameters. Default: 30 s. When unchecked, the device is scanning continuously. The timeout cannot occur before the scanning interval is expired, that is why if a timeout value is less than slow scanning interval minimum value, a warning is displayed.

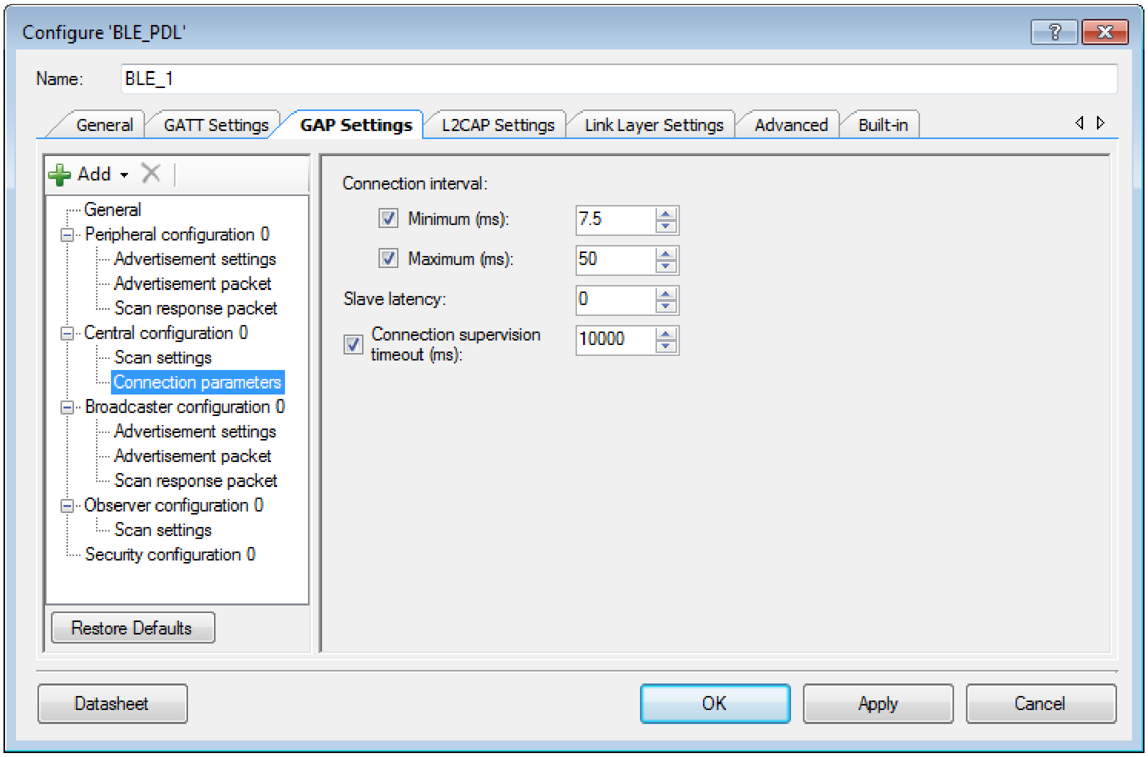
- **Slow scan parameters** – This connection results in a slower connection between the GATT Client and GATT Server devices than is possible using a normal connection. However this method consumes less power.
  - □ **Scan Window**: This parameter defines the scan window when operating in **Slow Connection**. The parameter is configured to increment in multiples of 0.625ms. Valid range is from 2.5 ms to 10240 ms. **Scan Window** must be less than the **Scan Interval**. Default: 1125 ms.
  - □ **Scan Interval**: This parameter defines the scan interval when operating in **Slow Connection**. The parameter is configured to increment in multiples of 0.625 ms. Valid range is from 2.5 ms to 10240 ms. Default: 1280 ms.
  - □ **Scan Timeout**: The timeout value of scanning with slow scan parameters. Default: 150 s. When unchecked, the device is scanning continuously. The timeout cannot occur before the scanning interval is expired, that is why if a timeout value is less than slow scanning interval minimum value, a warning is displayed.





GAP Settings Tab – Connection parameters

These parameters define the preferred BLE interface connection settings of the Central.



Name: BLE_1

Connection interval:

☑ Minimum (ms): 7.5

☑ Maximum (ms): 50

Slave latency: 0

☑ Connection supervision timeout (ms): 10000

Note The scaled values of these parameters used internally by the BLE stack. These are the actual values sent over the air.

- **Connection interval –** The Central device connecting to a Peripheral device needs to define the time interval for a connection to happen.
  - Minimum (ms): This parameter is the minimum permissible connection time value to be used during a connection event. It is configured in steps of 1.25 ms. The range is from 7.5 ms to 4000 ms. Unchecked means no specific minimum.
  - Maximum (ms): This parameter is the maximum permissible connection time value to be used during a connection event. It is configured in steps of 1.25 ms. The range is from 7.5 ms to 4000 ms. Unchecked means no specific maximum.

  Note In multi connection use case, the recommended minimum connection interval per connection should be greater than N * Max Time taken by individual connections to complete a BLE Connection Event (CE).

  Min_CI = N * Average Time Per CE

Average time for each CE is the amount of time taken to complete one BLE Tx and Rx transaction. This time varies depending on the Link Layer Data Length Extension (DLE) and BLE data rate (1 Mbps or 2 Mbps) configuration. The application can use the following timing lookup table for the CE value:

1. If DLE is enabled and data rate is 1Mbps, Average time = 6ms
2. If DLE is enabled and data rate is 2Mbps, Average time = 3.5ms
3. If DLE is disabled and data rate is 1Mbps, Average time = 2ms
4. If DLE is disabled and data rate is 2Mbps, Average time = 1.6ms

For example, if application supports 4 BLE connections with DLE and 1Mbps data rate, then the recommended minimum connection interval for each of the connections is:

4 * 6 = 24ms.

Note Connection intervals less than this value will still work, but under certain circumstances, the real time control procedures (connection update, channel map update etc.) with shorter update instance might result in link disconnection.

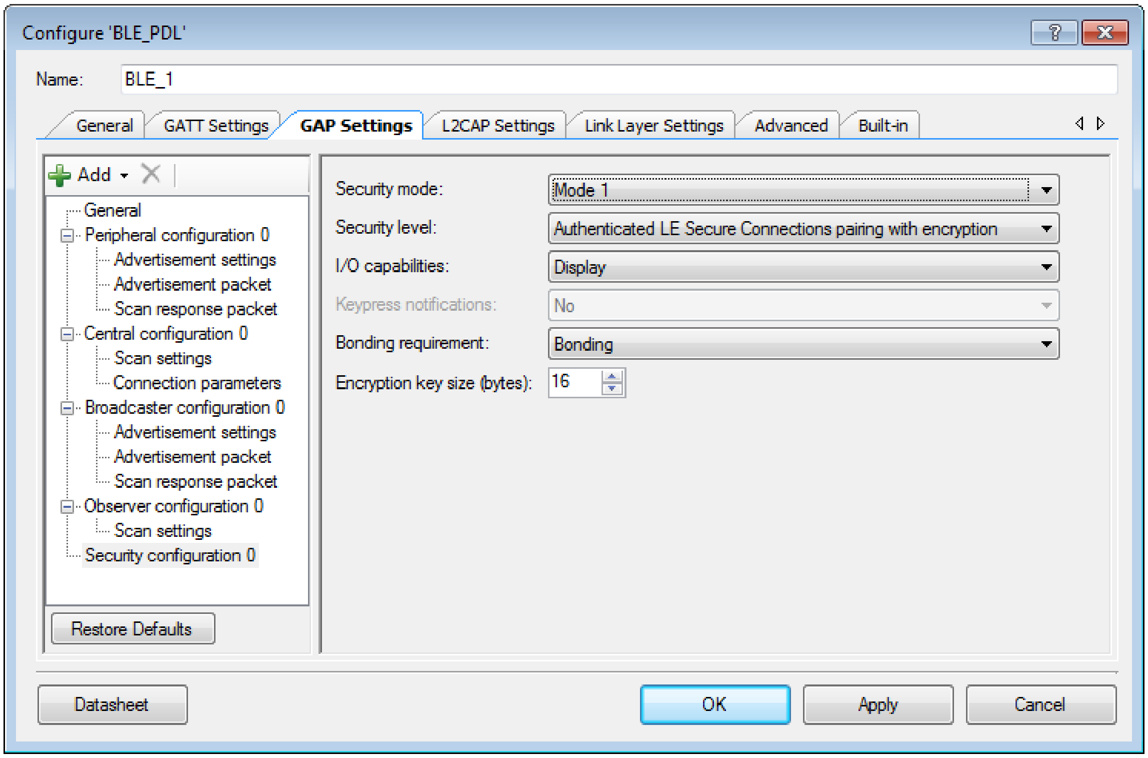
- **Slave Latency –** Defines the latency of the slave in responding to a connection event in consecutive connection events. This is expressed in terms of multiples of connection intervals, where only one connection event is allowed per interval. The range is from 0 to 499 events.

- **Connection Supervision Timeout –** This parameter defines the LE link supervision timeout interval. It defines the timeout duration for which an LE link needs to be sustained in case of no response from peer device over the LE link. The time interval is configured in multiples of 10 ms. Unchecked means no specific value. The range is from 100 ms to 32000 ms.

  Note For proper operation, the Connection Supervision Timeout must be larger than **(1 + Slave latency) * Connection Interval * 2** (ms). Refer to Bluetooth Core Specification Volume 6, Part B, Chapter 4.5.2 for more information on Connection Supervision Timeout.

GAP Settings Tab – Security

This section contains several parameters to configure the global security options for the Component. These parameters are configurable only if a connectable GAP role, Peripheral or Central, is selected. You can optionally set each Characteristic using its own unique security setting in the **Profile Tree**.



Security mode: Mode 1
Security level: Authenticated LE Secure Connections pairing with encryption
I/O capabilities: Display
Keypress notifications: No
Bonding requirement: Bonding
Encryption key size (bytes): 16

Security mode

Defines GAP security modes for the Component. Both available modes may support authentication.

- Mode 1 – Used in designs where data encryption is required.
- Mode 2 – Used in designs where data signing is required.

Security level

Enables different levels of security depending on the selected **Security mode**:

- If Mode1 is selected, then the following security levels are available.
  - No Security – With this level of security, the device will not use encryption or authentication.
  - Unauthenticated pairing with encryption – With this level of security, the device will send encrypted data after establishing a connection with the remote device.

  - Authenticated pairing with encryption – With this level of security, the device will send encrypted data after establishing a connection with the remote device. To establish a connection, devices should perform the authenticated paring procedure.
  - Authenticated LE Secure Connections pairing with encryption – With this level of security, the device uses an algorithm called Elliptic curve Diffie–Hellman (ECDH) for key generation, and a new pairing procedure for the key exchange. It also provides a new protection method from Man-In-The-Middle (MITM) attacks - Numeric Comparison.

- If Mode 2 is selected, then the following security levels are available.
  - Unauthenticated pairing with data signing – With this level of security, the device will perform data signing prior to sending it to the remote device after they establish a connection.
  - Authenticated pairing with data signing – With this level of security, the device will perform data signing prior to sending it to the remote device after they establish a connection. To establish a connection, the devices should perform the authenticated paring procedure.

Keypress notifications

Provides an option for a keyboard device during the LE secure pairing process to send key press notifications when the user enters or deletes a key. This option is available when the **Security level** is set to Authenticated LE Secure Connections pairing with encryption and **I/O capabilities** option is set to either Keyboard or Keyboard and Display.

I/O capabilities

This parameter refers to the device's input and output capability that can enable or restrict a particular pairing method or security level.

- Display – Used in devices with display capability and may display authentication data. GAP authentication is required.
- Display Yes/No – Used in devices with display and at least two input keys for Yes/No action. GAP authentication is required.
- Keyboard – Used in devices with numeric keypad. GAP authentication is required.
- No Input No Output – Used in devices that don't have any capability to enter or display the authentication key data to the user. Used in mouse-like devices. No GAP authentication is required.
- Keyboard and Display – Used in devices like PCs and tablets. GAP authentication is required.

Bonding Requirement

This parameter is used to configure the bonding requirements. The purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices, to be used for future authentication. The maximum number of remote devices that can be bonded is 128.

- **Bonding:** The device will store the link key of a connection after paring with the remote device in the flash memory and if a connection will be lost and re-established, the devices will use the previously stored key for the connection.

  Note Bonding information is stored in RAM and should be written to Flash if it needs to be retained during shutdown. Refer to the Functional_Description section for details on bonding and Flash write usage.

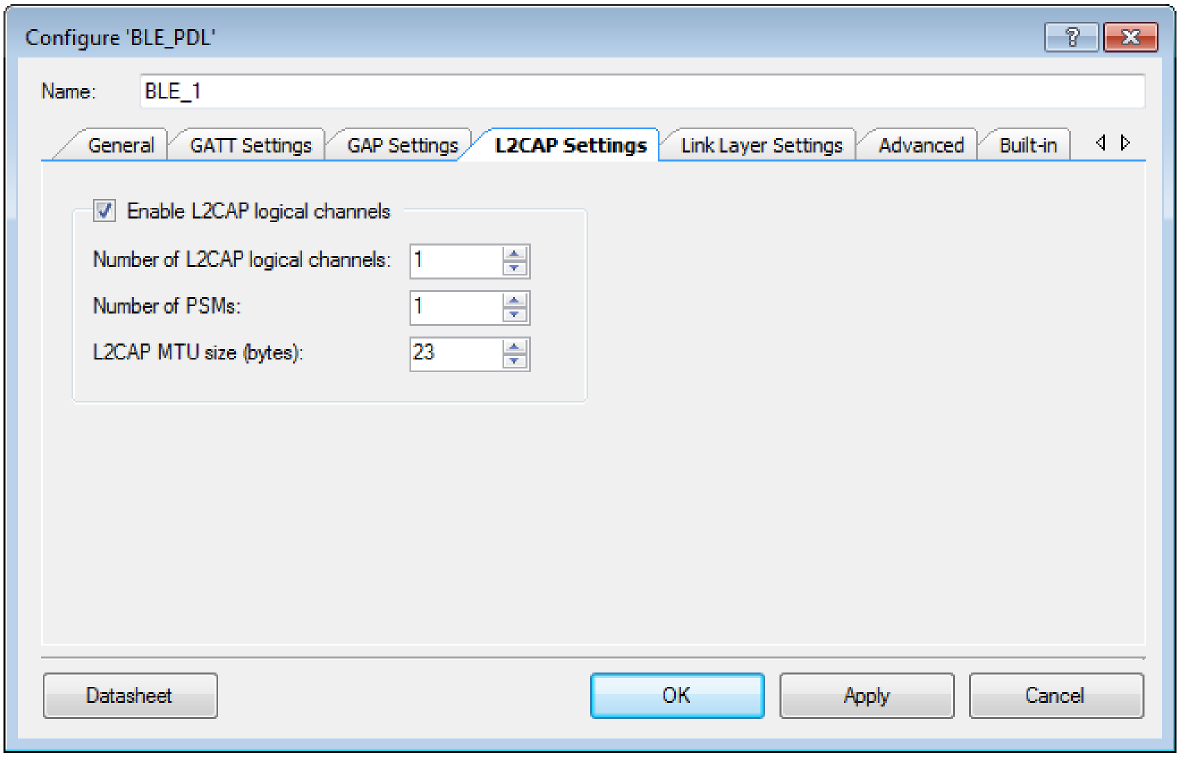
- **No Bonding:** The pairing process will be performed on each connection establishment.

Encryption Key Size

This parameter defines the encryption key size based on the Profile requirement. The valid values of encryption key size are 7 to 16 bytes.

L2CAP Settings Tab

The L2CAP settings define parameters for L2CAP connection oriented channel configuration.



Configure 'BLE_PDL'

Name: BLE_1

☑ Enable L2CAP logical channels

Number of L2CAP logical channels: 1

Number of PSMs: 1

L2CAP MTU size (bytes): 23

Enable L2CAP logical channels

This parameter enables configuration of the L2CAP logical channels. Default: true.

Number of L2CAP logical channels

This parameter defines the number of LE L2CAP connection oriented logical channels required by the application. Valid range is from 1 to 255. Default: 1.

Number of PSMs

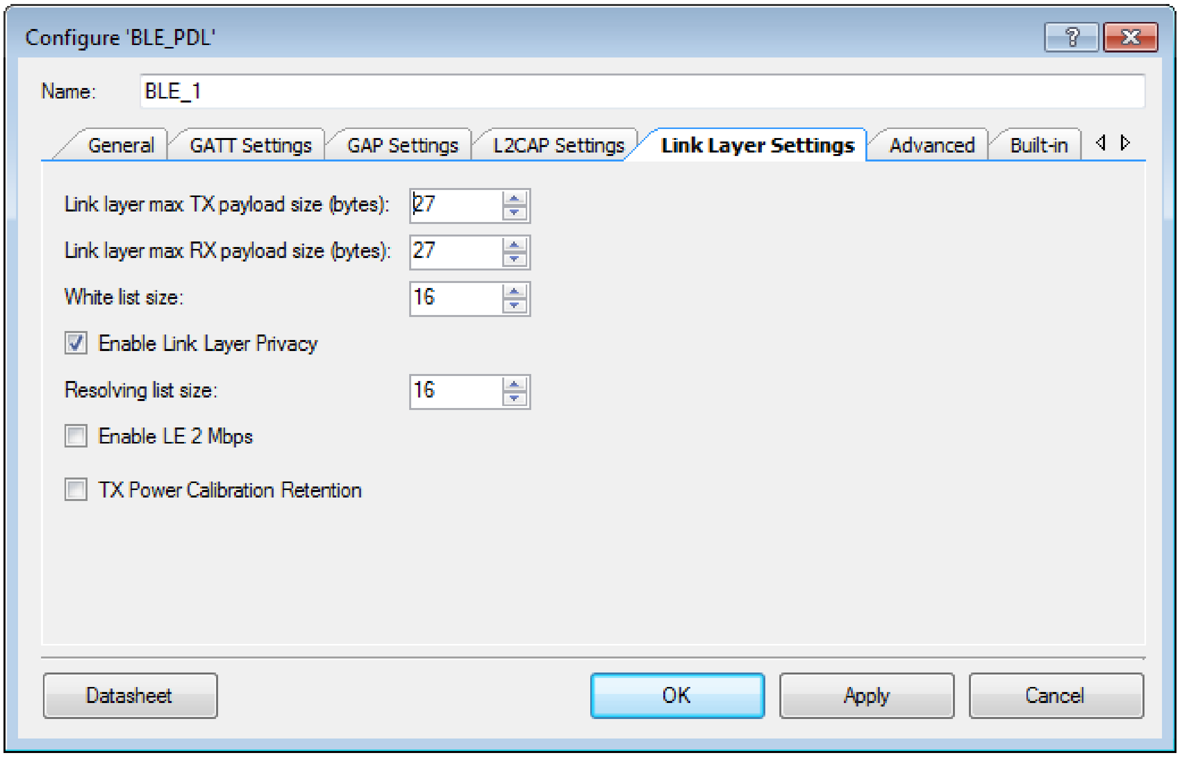
This parameter defines the number of PSMs required by the application. Valid range is from 1 to 255. Default: 1.

L2CAP MTU size

This parameter defines the maximum SDU size of an L2CAP packet. Valid range is from 23 to 65488 bytes. Default: 1280 bytes when **Internet Protocol Support Service** is supported and 23 bytes otherwise.

Link Layer Settings

The Link Layer settings parameters for Link Layer.



Name: BLE_1

Link layer max TX payload size (bytes): 27

Link layer max RX payload size (bytes): 27

White list size: 16

☑ Enable Link Layer Privacy

Resolving list size: 16

☐ Enable LE 2 Mbps

☐ TX Power Calibration Retention

Link Layer Max Tx Payload Size

The maximum link layer transmits payload size to be used in the design. The actual size of the link layer transmit packet is decided based on the peer device's link layer receive packet size during Data Length Update Procedure and will be informed through 'CY_BLE_EVT_GAP_DATA_LENGTH_CHANGE' event. Valid range is from 27 to 251 bytes.

Link Layer Max Rx Payload Size

The maximum link layer receives payload size to be used in the design. The actual size of the link layer receive packet is decided based on the peer device's link layer transmit packet size during Data Length Update Procedure and will be informed through 'CY_BLE_EVT_GAP_DATA_LENGTH_CHANGE' event. Valid range is from 27 to 251 bytes.

Setting the Link Layer Max Tx Payload Size or Link Layer Max Rx Payload Size to the value greater than 27 enables the LE Data Length Extension feature.

White list size

The maximum number of devices that can be added to the white list. Valid range is from 1 to 16. Default: 16.

Enable Link Layer Privacy

Enables LL Privacy 1.2 feature of Bluetooth 4.2 and enables generation of CY_BLE_EVT_GAP_ENHANCE_CONN_COMPLETE and CY_BLE_EVT_GAPC_DIRECT_ADV_REPORT events.

Note that CY_BLE_EVT_GAP_DEVICE_CONNECTED event is not generated when this feature is enabled.

Resolving list size

The maximum number of peer devices whose addresses should be resolved by this device. This parameter is applicable when the **Enable Link Layer Privacy** feature is enabled. Valid range is from 1 to 16. Default: 16.

Enable LE 2 Mbps

Enables LE 2 Mbps feature of Bluetooth 5.0.

The 2 Mbps feature enables new Physical (PHY) modulation scheme allowing to increase data throughput between two devices which support this feature. Refer to BluetoothCore Specification v5.0 for more details about this feature.

Use Cy_BLE_SetDefaultPhy() API after CY_BLE_EVT_STACK_ON event to set preferred default PHY for all connections, or Cy_BLE_SetPhy() API to set PHY for the current connection.

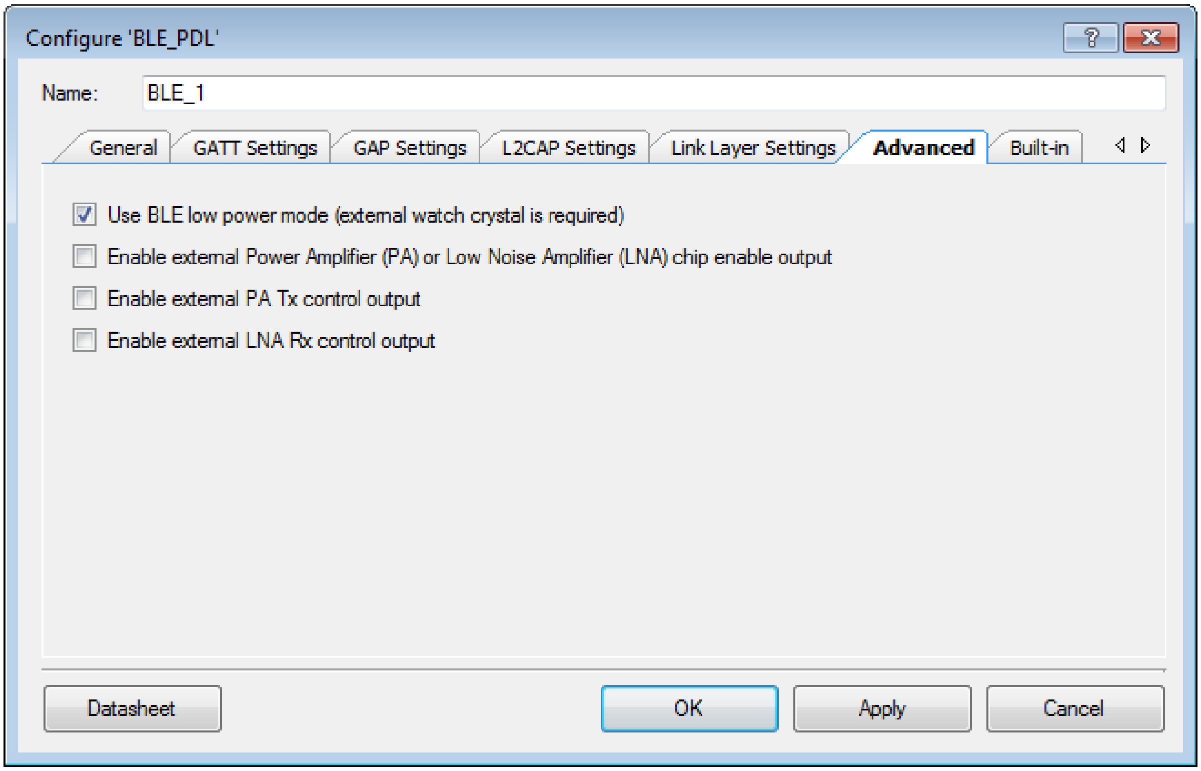
CY_BLE_EVT_PHY_UPDATE_COMPLETE event will indicate when Controller has changed the transmitter PHY or receiver PHY in use.

Enable Tx Power Calibration Retention

When enabled, BLE radio Tx power calibration is performed only once after programming and the calibration values are retained in SFLASH location. This retained value is reloaded to radio power calibration registers during consecutive device reboots. This reduces the BLE stack boot up time significantly

Advanced Tab

The Advanced parameters define parameters for low-power mode and external power amplification.



Name: BLE_1

☑ Use BLE low power mode (external watch crystal is required)

□ Enable external Power Amplifier (PA) or Low Noise Amplifier (LNA) chip enable output

□ Enable external PA Tx control output

□ Enable external LNA Rx control output

Use BLE low power mode

This parameter identifies if the low-power mode support is required for the BLE_PDL Component. Default: true.

When this parameter is set, WCO or PILO must be selected as the LFCLK source in the Design-Wide Resources Clock Editor. This configuration is a requirement if you intend to use the Component in the low-power mode.

Enable external Power Amplifier (PA) or Low Noise Amplifier (LNA) chip enable output

This parameter enables external PA and LNA chip enable control pins. Default: false.

Enable external PA Tx control output

When selected, ext_pa_tx_ctl_out signal from the BLE radio is routed on a GPIO. Default: false.

Enable external LNA Rx control output

When selected, ext_lna_rx_ctl_out signal from the BLE radio is routed on a GPIO. Default: false.

By default, all PA outputs operate in High enable, Low disable mode. Use Cy_BLE_ConfigureExtPA() API after CY_BLE_EVT_STACK_ON event to change default outputs polarity.

Application Programming Interface

Application Programming Interface (API) is provided by the BLE middleware library from the PDL. The middleware is copied into the “pdl\middleware\ble\” directory of the application project after a successful build.

The Component generates the configuration structures described in the Global variables and Preprocessor Macros sections. Pass the generated data structures to the associated BLE middleware function in the application initialization code to configure the peripheral. Once the peripheral is initialized, the application code can perform run-time changes by the middleware API functions.

The BLE middleware contains a comprehensive API list to allow you to configure the BLE stack, the underlying chip hardware and the BLE service specific configuration using software. You may access the GAP, GATT and L2CAP layers of the stack using these.

Refer to the BLE Middleware Library documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open PDL Documentation…**” option in the drop-down menu.

In addition to the PDL API, the BLE_PDL Component provides an instance-based Component API with additional functionality available through PSoC Creator.

Note All BLE_PDL Component API names begin with Cy_BLE_. This is a unique feature of the BLE_PDL Component, and allows only one instance of the Component to be placed in your design.

Global Variables

The BLE_PDL Component populates the following peripheral initialization data structure(s).

uint8_t cy_ble_initVar = 0u

Indicates whether the BLE has been initialized. The variable is initialized to 0 and set to 1 the first time Cy_BLE_Start() is called. This allows the Component to restart without reinitialization after the first call to the Cy_BLE_Start() routine. If reinitialization of the Component is required, the variable should be set to 0 before the Cy_BLE_Start() routine is called. Alternatively, the BLE can be reinitialized by calling the Cy_BLE_Init() function.

cy_stc_ble_config_t cy_ble_config

The configuration structure for BLE.

cy_stc_ble_aios_config_t cy_ble_aiosConfig

The configuration structure for the Automation Input Output Service.

cy_stc_ble_ancs_config_t cy_ble_ancsConfig

The configuration structure for the Apple Notification Center Service.

cy_stc_ble_ans_config_t cy_ble_ansConfig

The configuration structure for the Alert Notification Service.

cy_stc_ble_bas_config_t cy_ble_basConfig

The configuration structure for the Battery Service.

cy_stc_ble_bcs_config_t cy_ble_bcsConfig

The configuration structure for the Body Composition Service.

cy_stc_ble_bls_config_t cy_ble_blsConfig

The configuration structure for the Blood Pressure Service.

cy_stc_ble_bms_config_t cy_ble_bmsConfig

The configuration structure for the Bond Management Service.

cy_stc_ble_cgms_config_t cy_ble_cgmsConfig

The configuration structure for the Continuous Glucose Monitoring Service.

cy_stc_ble_cps_config_t cy_ble_cpsConfig

The configuration structure for the Cycling Power Service.

cy_stc_ble_cscs_config_t cy_ble_cscsConfig

The configuration structure for the Cycling Speed and Cadence Service.

cy_stc_ble_cts_config_t cy_ble_ctsConfig

The configuration structure for the Current Time Service.

cy_stc_ble_custom_config_t cy_ble_customConfig

The configuration structure for the Custom Services.

cy_stc_ble_dis_config_t cy_ble_disConfig

The configuration structure for the Device Information Service.

cy_stc_ble_ess_config_t cy_ble_essConfig

The configuration structure for the Environmental Sensing Service.

cy_stc_ble_gls_config_t cy_ble_glsConfig

The configuration structure for the Glucose Service.

cy_stc_ble_hids_config_t cy_ble_hidsConfig

The configuration structure for the HID service.

cy_stc_ble_hps_config_t cy_ble_hpsConfig

The configuration structure for the HTTP Proxy Service.

cy_stc_ble_hrs_config_t cy_ble_hrsConfig

The configuration structure for the Heart Rate Service.

cy_stc_ble_hts_config_t cy_ble_htsConfig

The configuration structure for the Health Thermometer Service.

cy_stc_ble_ias_config_t cy_ble_iasConfig

The configuration structure for the Immediate Alert Service.

cy_stc_ble_ips_config_t cy_ble_ipsConfig

The configuration structure for the Indoor Positioning Service.

cy_stc_ble_lls_config_t cy_ble_llsConfig

The configuration structure for the Link Loss Service.

cy_stc_ble_lns_config_t cy_ble_lnsConfig

The configuration structure for the Location and Navigation Service.

cy_stc_ble_ndcs_config_t cy_ble_ndcsConfig

The configuration structure for the Next DST Change Service.

cy_stc_ble_pass_config_t cy_ble_passConfig

The configuration structure for the Phone Alert Status Service.

cy_stc_ble_plxs_config_t cy_ble_plxsConfig

The configuration structure for the Pulse Oximeter Service.

cy_stc_ble_rscs_config_t cy_ble_rscsConfig

The configuration structure for the Running Speed and Cadence Service.

cy_stc_ble_rtus_config_t cy_ble_rtusConfig

The configuration structure for the Reference Time Update Service.

cy_stc_ble_scps_config_t cy_ble_scpsConfig

The configuration structure for the Scan Parameter Service.

cy_stc_ble_tps_config_t cy_ble_tpsConfig

The configuration structure for the Tx Power Service.

cy_stc_ble_uds_config_t cy_ble_udsConfig

The configuration structure for the User Data Service.

cy_stc_ble_wpts_config_t cy_ble_wptsConfig

The configuration structure for the Wireless Power Transfer Service.

cy_stc_ble_wss_config_t cy_ble_wssConfig

The configuration structure for the Weight Scale Service.

Preprocessor Macros

The BLE_PDL Component generates the following preprocessor macro(s). Note that actual macro values vary depending on the Component settings in the Configure dialog.

#define CY_BLE_CONFIG_MODE

The BLE operation mode.

#define CY_BLE_CONFIG_HOST_CORE

The core for the Host in Profile mode.

#define CY_BLE_CONFIG_HCI_CONTR_CORE

The core for the Controller in HCI mode.

#define CY_BLE_CONFIG_STACK_MODE

The BLE Stack core mode:

- CY_BLE_STACK_MODE_SINGLE_SOC - The Host and Controller with a software interface.
- CY_BLE_STACK_MODE_DUAL_IPC - The Host and Controller with an IPC interface.

#define CY_BLE_CONFIG_CONN_COUNT

The maximum number of BLE connections. The valid range is from 1 to 4.

#define CY_BLE_CONFIG_GATTC_COUNT

The number of BLE connections (Client).

#define CY_BLE_CONFIG_GAP_PERIPHERAL_COUNT

The number of GAP Peripheral configurations structures.

#define CY_BLE_CONFIG_GAP_BROADCASTER_COUNT

The number of GAP Broadcaster configurations structures.

#define CY_BLE_CONFIG_GAP_CENTRAL_COUNT

The number of GAP Central configurations structures.

#define CY_BLE_CONFIG_GAP_OBSERVER_COUNT

The number of GAP Observer configurations structures.

#define CY_BLE_CONFIG_AUTH_INFO_COUNT

The number of Security configurations structures.

#define CY_BLE_CONFIG_AUTO_POPULATE_WHITELIST

Provides an option to link the whitelist to the bonded device list.

#define CY_BLE_CONFIG_MAX_RESOLVABLE_DEVICES

The maximum number of peer devices whose addresses should be resolved by this device.

#define CY_BLE_CONFIG_ENABLE_LL_PRIVACY

LL Privacy 1.2 feature.

#define CY_BLE_CONFIG_ENABLE_PHY_UPDATE

LE 2 Mbps feature.

#define CY_BLE_CONFIG_GAP_SECURITY_LEVEL

The GAP security level.

#define CY_BLE_CONFIG_GAP_ROLE

The GAP Role.

#define CY_BLE_CONFIG_BONDING_REQUIREMENT

The Bonding requirement.

#define CY_BLE_CONFIG_GATT_MTU

The GATT MTU Size.

#define CY_BLE_CONFIG_GATT_DB_MAX_VALUE_LEN

The GATT maximum attribute length.

#define CY_BLE_CONFIG_GATT_RELIABLE_CHAR_COUNT

The number of characteristics supporting a Reliable Write property.

#define CY_BLE_CONFIG_GATT_RELIABLE_CHAR_LENGTH

The total length of characteristics with Reliable Write property.

#define CY_BLE_CONFIG_GATT_ENABLE_EXTERNAL_PREP_WRITE_BUFF

The parameter to enable an application to provide a dynamically allocated buffer for preparing a Write request.

#define CY_BLE_CONFIG_L2CAP_ENABLE

The parameter to enable configuration of the L2CAP logical channels.

#define CY_BLE_CONFIG_L2CAP_MTU

The L2CAP MTU size.

#define CY_BLE_CONFIG_L2CAP_MPS

The L2CAP MPS size.

#define CY_BLE_CONFIG_L2CAP_LOGICAL_CHANNEL_COUNT

The number of L2CAP logical channels.

#define CY_BLE_CONFIG_L2CAP_PSM_COUNT

The number of L2CAP PSMs.

#define CY_BLE_CONFIG_LL_MAX_TX_PAYLOAD_SIZE

The maximum Tx payload size.

#define CY_BLE_CONFIG_LL_MAX_RX_PAYLOAD_SIZE

The maximum Rx payload size.

#define CY_BLE_CONFIG_GATT_ROLE

The GATT role.

#define CY_BLE_CONFIG_HIDSS_SERVICE_COUNT

The maximum supported count of HID services for the GATT Server role.

#define CY_BLE_CONFIG_HIDSS_REPORT_COUNT

The maximum supported count of HID reports for the GATT Server role.

#define CY_BLE_CONFIG_HIDSC_SERVICE_COUNT

The maximum supported count of HID services for the GATT Client role.

#define CY_BLE_CONFIG_HIDSC_REPORT_COUNT

The maximum supported count of HID reports for the GATT Client role.

#define CY_BLE_CONFIG_BASS_SERVICE_COUNT

The maximum supported count of BAS services for the GATT Server role.

#define CY_BLE_CONFIG_BASC_SERVICE_COUNT

The maximum supported count of BAS reports for the GATT Client role.

#define CY_BLE_CONFIG_ES_TOTAL_CHAR_COUNT

The maximum supported count of ESS characteristics for the GATT Client role.

#define CY_BLE_CONFIG_AIO_TOTAL_CHAR_COUNT

The maximum supported count of AIOS characteristics for the GATT Client role.

#define CY_BLE_CONFIG_CUSTOMS_SERVICE_COUNT

The maximum supported count of Custom services for the GATT Server role.

#define CY_BLE_CONFIG_CUSTOMC_SERVICE_COUNT

The maximum supported count of Custom services for the GATT Client role.

#define CY_BLE_CONFIG_CUSTOM_SERVICE_CHAR_COUNT

The maximum supported count of the Custom Service characteristics.

#define CY_BLE_CONFIG_CUSTOM_SERVICE_CHAR_DESCRIPTORS_COUNT

The maximum supported count of the Custom Service descriptors in one characteristic.

Component Functions

This Component also includes a set of Component-specific functions that provide simplified access to the basic BLE middleware operation.

cy_en_ble_api_result_t Cy_BLE_Start (cy_ble_callback_t *callbackFunc*)

This function initializes the BLE Stack which consists of the BLE Stack Manager, BLE Controller, and BLE Host modules. It takes care of initializing the Profile layer, schedulers, Timer and other platform-related resources required for the BLE_PDL Component. It also registers the callback function for BLE events that will be registered in the BLE stack.

Note that this function does not reset the BLE Stack.

For HCI mode of operation, this function will not initialize the BLE Host module.

Calling this function results in generation of a CY_BLE_EVT_STACK_ON event on successful initialization of the BLE Stack.

In Dual Core mode, this function should be called on both cores in the following sequence:

- call this function on CM0+ core to initialize the Controller.
- start CM4 core by calling Cy_SysEnableCM4() function.
- call this function on CM4 core to initialize the Host and Profiles.

Parameters:

callbackFunc	Event callback function to receive events from BLE stack. cy_ble_callback_t is a function pointer type.

Returns:

cy_en_ble_api_result_t : Return value indicates if the function succeeded or failed. The following are possible error codes.

Error codes	Description
CY_BLE_SUCCESS	On successful operation.
CY_BLE_ERROR_INVALID_PARAMETER	On passing a NULL pointer to the function when the BLE stack is not built in the HCI mode. CY_BLE_ERROR_INVALID_PARAMETER is never returned in HCI mode.
CY_BLE_ERROR_REPEATED_ATTEMPTS	On invoking this function more than once without calling Cy_BLE_StackShutdown() function between calls to this function.
CY_BLE_ERROR_MEMORY_ALLOCATION_FAILED	There is insufficient memory available.

Global Variables

The cy_ble_initVar variable is used to indicate the initial configuration of this Component. The variable is initialized to a zero (0u) and set to one (1u) the first time Cy_BLE_Start() is called. This allows for Component initialization without re-initialization in all subsequent calls to the Cy_BLE_Start() routine.

cy_en_ble_api_result_t Cy_BLE_Stop (void)

This function stops any ongoing operation in the BLE Stack and forces the BLE Stack to shut down. The only function that can be called after calling this function is Cy_BLE_Start() when Cy_BLE_GetState() API returns CY_BLE_STATE_STOPPED state.

Calling this function results in generation of a CY_BLE_EVT_STACK_SHUTDOWN_COMPLETE event on a successful stack shutdown.

Returns:

cy_en_ble_api_result_t: Return value indicates if the function succeeded or failed. Following are the possible error codes.

Errors codes	Description
CY_BLE_SUCCESS	On successful operation.
CY_BLE_ERROR_INVALID_OPERATION	On calling Cy_BLE_Stop before calling Cy_BLE_Start.

Callback Functions

The BLE_PDL Component requires that you define a callback function for handling BLE stack events. This is passed as a parameter to the Cy_BLE_Start() API. The callback function is of type cy_ble_callback_t, as defined by:

```
void (* cy_ble_callback_t)(uint32_t eventCode, void *eventParam);
```

- eventCode: The stack event code
- eventParam : Stack event parameters

The callback function should then evaluate the eventCode (and eventParam for certain events) and provide stack event-specific actions. Hence the events are used to build your application specific state machine for general events such as advertisement, scan, connection and timeout.

The BLE stack triggers the application event handler callback for any pending events generated by the link layer, after calling the Cy_BLE_ProcessEvents() API method. However, other BLE_PDL Component API methods requesting host-generated actions can also trigger the application event handler callback, causing events to be processed before these API methods return.

Similarly, you will need to provide a callback function for each Service that you wish to use. This function is also of type cy_ble_callback_t and is passed as a parameter to the Service-specific callback registration function. The callback function is used to evaluate the Service-specific events and to take appropriate action as defined by your application. Then a Service specific state machine can be built using these events.

Code snippets

- For an application callback: void Cy_BLE_AppCallback(uint32_t eventCode, void *eventParam){<all general events>}
- For each Cy_BLE_<service>RegisterAttrCallback API function: Cy_BLE_<service>RegisterAttrCallback(Cy_BLE_<service>CallBack);
- For each service callback: void Cy_BLE_<service>CallBack(uint32_t eventCode, void *eventParam) {<all service-specific events>}

Code Examples and Application Notes

Code Examples

PSoC Creator provides numerous codes examples that include schematics and example code in the Find Code Example dialog. For Component-specific examples, open the dialog from the Component Catalog or an instance of the Component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Code Example" topic in the PSoC Creator Help for more information.

Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes. Application Notes that use this Component include:

- AN210781 – Getting started with PSoC 6 BLE
- AN95089 – PSoC/PRoC BLE Crystal Oscillator Selection and Tuning Techniques
- AN91445 – Antenna Design and RF Layout Guidelines

Industry Standards

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are three types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- Component-specific deviations – deviations that are applicable only for the common part of this Component
- Profile specific deviations – deviations that are applicable only for a specific Profile of the Component. Refer to the BLE Middleware Library documentation for a detailed description of the Profile specific deviations.

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) along with information on the MISRA compliance verification environment.

The BLE_PDL Component has the following specific deviations.

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
1.1	R	This rule states that code shall conform to C ISO/IEC 9899:1990 standard.	This Component supports ISO:C99 standard.
1.2	R	No reliance shall be placed on undefined or unspecified behavior.	This specific behavior is explicitly covered in rule 5.1.
3.1	A	A cast should not be performed between a pointer to volatile object and an integral type.	The cast from unsigned int to pointer does not have any unintended effect, as it is a consequence of the definition of a structure based on hardware registers.
5.1	R	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.	This rule applies to ISO:C90 standard. Component-specific code conforms to ISO:C99 that does not require this limitation.
10.1	R	The value of an expression of integer type shall not be implicitly converted to a different underlying type under some circumstances.	An operand of essentially enum type is being converted to unsigned type as a result of an arithmetic or conditional operation. The conversion does not have any unintended effect.
11.4	A	A cast should not be performed between a pointer to object type and a different pointer to object type.	A cast involving pointers is conducted with caution that the pointers are correctly aligned for the type of object being pointed to.

This Component has the following embedded Components: cy_isr, SCB. Refer to the corresponding Component datasheets for information on their MISRA compliance and specific deviations.

Bluetooth Qualification

BLE solutions provided by Cypress are listed on the Bluetooth SIG website as certified solutions. The qualification is modular, allowing greater flexibility to customers. The following is the list of Qualified Design IDs (QD ID) and Declaration IDs.

QD ID(s)	Declaration ID#	Description
99158	D037716	PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity delivers ultra-low-power, best-in-class flexible and built-in security for the Internet of Things (IoT). It's built on an ultra-low-power 40-nm process technology and uses a dual-core ARM® Cortex®-M architecture, which includes an Cortex-M4F and Cortex-M0+. The BLE radio is built using an ultra-low-power 55-nm process. PSoC 6 MCU with BLE Connectivity combines a BLE subsystem with software-defined analog and digital peripherals, CapSense®, programmable interconnect, a high-performance dual-core architecture, and critical security features in a single chip.

Resources

The BLE_PDL Component uses one BLESS block, two external crystals, interrupt(s), and an optional SCB Block:

Configuration	Resource Type				
	BLESS[1]	SCB [2]	Interrupt	ECO	WCO [3]
Profile Mode	1	-	1	1	1
HCI Mode	1	1	2	1	1

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

Note Final characterization data for PSoC 6 devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
BLE Sub-system Specifications						
RF Receiver Specification (1 Mbps)						
RXS, IDLE	RX Sensitivity with Ideal Transmitter	–	–95	–	dBm	Across RF Operating Frequency Range
RXS, IDLE	RX Sensitivity with Ideal Transmitter	–	–93	–	dBm	255-byte Packet Length, Across Frequency Range
RXS, DIRTY	RX Sensitivity with Dirty Transmitter	–	–92	–	dBm	RF-PHY Specification (RCV-LE/CA/01/C)
PRXMAX	Maximum received signal strength at < 0.1% PER	–	0	–	dBm	RF-PHY Specification (RCV-LE/CA/06/C)
CI1	Co-channel interference, Wanted Signal at –67 dBm and Interferer at FRX	–	9	21	dB	RF-PHY Specification (RCV-LE/CA/03/C)

[1] The BLESS Component instantiates an SCB Component when configured in HCI Mode. Refer to the SCB Component datasheet for its resource usage.

[2] The BLE Component instantiates an SCB Component when configured in HCI Mode. Refer to the SCB Component datasheet for its resource usage.

[3] WCO is optional. It is used if Component Deep Sleep is required. If WCO is not used, then ILO is used as the LFCLK source.

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
CI2	Adjacent channel interference. Wanted Signal at –67 dBm and Interferer at FRX ± 1 MHz	–	3	15	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI3	Adjacent channel interference. Wanted Signal at –67 dBm and Interferer at FRX ± 2 MHz	–	–29	–17	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI4	Adjacent channel interference. Wanted Signal at –67 dBm and Interferer at FRX ± 3 MHz	–	–39	–27	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI5	Adjacent channel interference. Wanted Signal at –67 dBm and Interferer at Image frequency (FIMAGE)	–	–20	–9	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI6	Adjacent channel interference. Wanted Signal at –67 dBm and Interferer at Image frequency (FIMAGE ± 1 MHz)	–	–30	–15	dB	RF-PHY Specification (RCV-LE/CA/03/C)
RF Receiver Specification (2 Mbps)						
RXS,IDLE	RX Sensitivity with Ideal Transmitter		–92		dBm	Across RF Operating Frequency Range
RXS,IDLE	RX Sensitivity with Ideal Transmitter		–90		dBm	255-byte packet length, across Frequency Range
RXS,DIRTY	RX Sensitivity with Dirty Transmitter		–89		dBm	RF-PHY Specification (RCV-LE/CA/01/C)
PRXMAX	Maximum received signal strength at <0.1% PER		0		dBm	RF-PHY Specification (RCV-LE/CA/06/C)
CI1	Co-channel interference, Wanted Signal at –67 dBm and Interferer at FRX		9	21	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI2	Adjacent channel interference Wanted Signal at –67 dBm and Interferer at FRX ± 2 MHz		3	15	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI3	Adjacent channel interference Wanted Signal at –67 dBm and Interferer at FRX ± 4 MHz		–29	–17	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI4	Adjacent channel interference Wanted Signal at –67 dBm and Interferer at ± 6 MHz		–39	–27	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI5	Adjacent channel interference Wanted Signal at –67 dBm and Interferer at Image frequency (FIMAGE)		–20	–9	dB	RF-PHY Specification (RCV-LE/CA/03/C)
CI6	Adjacent channel interference Wanted Signal at –67 dBm and Interferer at (FIMAGE ± 2 MHz)		–30	–15	dB	RF-PHY Specification (RCV-LE/CA/03/C)
RF Receiver Specification (1 and 2 Mbps)						

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
OBB1	Out of Band Blocking Wanted Signal at –67 dBm and Interferer at F = 30-2000 MHz	–30	–27	-	dBm	RF-PHY Specification (RCV-LE/CA/04/C)
OBB2	Out of Band Blocking Wanted Signal at –67 dBm and Interferer at F = 2003-2399 MHz	–35	–27		dBm	RF-PHY Specification (RCV-LE/CA/04/C)
OBB3	Out of Band Blocking Wanted Signal at –67 dBm and Interferer at F= 2484-2997 MHz	–35	–27		dBm	RF-PHY Specification (RCV-LE/CA/04/C)
OBB4	Out of Band Blocking Wanted Signal at –67 dBm and Interferer at F=3000-12750 MHz	–30	–27		dBm	RF-PHY Specification (RCV-LE/CA/04/C)
IMD	Intermodulation Performance Wanted Signal at –64 dBm and 1 Mbps BLE, 3rd, 4th and 5th offset channel	–50			dBm	RF-PHY Specification (RCV-LE/CA/05/C)
RXSE1	Receiver spurious emission 30 MHz to 1.0 GHz			–57	dBm	100 kHz measurement bandwidth ETSI EN300 328 V1.8.1
RXSE2	Receiver Spurious emission 1.0 GHz to 12.75 GHz			–54	dBm	1 MHz measurement bandwidth ETSI EN300 328 V1.8.1
RF Transmitter Specifications						
TXP,ACC	RF Power Accuracy	–1	–	1	dB	–
TXP,RANGE	Frequency Accuracy		24		dB	–20 dBm to +4 dBm
TXP,0dBm	Output Power, 0 dB Gain setting		0		dBm	–
TXP,MAX	Output Power, Maximum Power Setting		4		dBm	–
TXP,MIN	Output Power, Minimum Power Setting		–20		dBm	–
F2AVG	Average Frequency deviation for 10101010 pattern	185			kHz	RF-PHY Specification (TRM-LE/CA/05/C)
F2AVG_2M	Average Frequency deviation for 10101010 pattern for 2 Mbps	370			kHz	RF-PHY Specification (TRM-LE/CA/05/C)
F1AVG	Average Frequency deviation for 11110000 pattern	225	250	275	kHz	RF-PHY Specification(TRM-LE/CA/05/C)
F1AVG_2M	Average Frequency deviation for 11110000 pattern for 2 Mbps	450	500	550	kHz	RF-PHY Specification (TRM-LE/CA/05/C)
EO	Eye opening = ¨F2AVG/¨F1AVG	0.8				RF-PHY Specification (TRM-LE/CA/05/C)
FTX, ACC	Frequency Accuracy	–150		150	kHz	RF-PHY Specification (TRM-LE/CA/06/C)
FTX, MAXDR	Maximum Frequency Drift	–50		50	kHz	RF-PHY Specification (TRM-LE/CA/06/C)

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
FTX, INITDR	Initial Frequency drift	–20		20	kHz	RF-PHY Specification (TRM-LE/CA/06/C)
FTX, DR	Maximum Drift Rate	–20		20	kHz/ 50 µs	RF-PHY Specification (TRM-LE/CA/06/C)
IBSE1	In Band Spurious Emission at 2 MHz offset (1 Mbps) In Band Spurious Emission at 4 MHz offset (2 Mbps)			–20	dBm	RF-PHY Specification (TRM-LE/CA/03/C)
IBSE2	In Band Spurious Emission at ≥ 3 MHz offset (1Mbps) In Band Spurious Emission at ≥ 6 MHz offset (2 Mbps)			–30	dBm	RF-PHY Specification (TRM-LE/CA/03/C)
TXSE1	Transmitter Spurious Emissions (Averaging), <1.0 GHz			–55.5	dBm	FCC-15.247
TXSE2	Transmitter Spurious Emissions (Averaging), >1.0 GHz			–41.5	dBm	FCC-15.247
RF Current Specifications						
IRX1_wb	Receive Current (1 Mbps)		4.4		mA	AVIN/PVIN and VIO current with buck
ITX1_wb_0dBm	TX Current at 0 dBm setting (1 Mbps)		4.2		mA	AVIN/PVIN and VIO current with buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
IRX1_nb	Receive Current (1 Mbps)		9.5		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX1_nb_0dBm	TX Current at 0 dBm setting (1 Mbps)		9		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX1_nb_4dBm	TX Current at 4 dBm setting (1 Mbps)		13		mA	AVIN/PVIN and VIO currentwithout buck(AVIN/PVIN = 1.8 V,VIO = 1.8 V)
ITX1_wb_4dBm	TX Current at 4 dBm setting (1 Mbps)		6.5		mA	AVIN/PVIN and VIO current with buck (AVIN/PVIN=1.8 V, VIO=1.8 V)
ITX1_nb_20dBm	TX Current at –20 dBm setting (1 Mbps)		7		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
IRX2_wb	Receive Current (2 Mbps)		4.4		mA	AVIN/PVIN and VIO current with buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX2_wb_0dBm	TX Current at 0 dBm setting (2 Mbps)		4.2		mA	AVIN/PVIN and VIO current with buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
IRX2_nb	Receive Current (2 Mbps)		9.5		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX2_nb_0dBm	TX Current at 0 dBm setting (2 Mbps)		9		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX2_nb_4dBm	TX Current at 4 dBm setting (2 Mbps)		13		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX2_wb_4dBm	TX Current at 4 dBm setting (2 Mbps)		6.5		mA	AVIN/PVIN and VIO current with buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
ITX2_nb_20dBm	TX Current at –20 dBm setting (2 Mbps)		7		mA	AVIN/PVIN and VIO current without buck (AVIN/PVIN = 1.8 V, VIO = 1.8 V)
General RF Specifications						
FREQ	RF operating frequency	2400		2482	MHz	–
CHBW	Channel spacing		2		MHz	–
DR1	On-air Data Rate (1 Mbps)		1000		Kbps	–
DR2	On-air Data Rate (2 Mbps)		2000		Kbps	–
TXSUP	Transmitter Startup time		80	82	s	–
RXSUP	Receiver Startup time		80	82	s	–
RSSI Specifications						
RSSI, ACC	RSSI Accuracy	–4	–	4	dB	–
RSSI, RES	RSSI Resolution		1		dB	–
RSSI, PER	RSSI Sample Period		6		s	–
System Level BLE Specifications						
Adv_Pwr	1.28s, 32 bytes, 0 dBm		21		µW	3.3 V, Buck, w/o Deep Sleep current
Conn_Pwr_300	300 ms, 0 byte, 0 dBm		33		µW	3.3 V, Buck, w/o Deep Sleep current
Conn_Pwr_1S	1000 ms, 0 byte, 0 dBm		10		µW	3.3 V, Buck, w/o Deep Sleep current
Conn_Pwr_4S	4000 ms, 0 byte, 0 dBm		3		µW	3.3 V, Buck, w/o Deep Sleep current

The following table summarizes the different measurements of the time taken by the BLE firmware stack to perform / initiate different BLE operations. The measurements have been performed with IMO with FLL set to 24 MHz, VDD set to 1.1V, connection interval set to 7.5 ms, and Encryption is enabled.

Operation	Duration (µs)
BLE Stack Start up Time (without DLE, Privacy, 2Mbps and Secure Connection)	21032
BLE Stack Start up Time (with DLE, Privacy, 2Mbps and Secure Connection)	21137
BLE Stack shutdown time	1170
CLOSE_CE ISR execution time – SSSS configuration	310
CLOSE_CE ISR execution time – MMMM configuration	254
EARLY_INTR ISR execution time – SSSS configuration	138
EARLY_INTR ISR execution time – MMMM configuration	84
Start Scan execution time	680
ADV Packet processing Time (ADV data length – 31 bytes)	212
SCAN_RESPONSE Packet Processing time (Scan response data length – 31 bytes)	218
Connection time on GAP Central	355824
Connection time on GAP Peripheral	780
Start advertisement execution time (Worst Case)	1568
Advertisement Data Update Time	368
Notification processing time on GATT Server 20 bytes without DLE (1-fragment)	635
Notification processing time on GATT Server 74 bytes without DLE (3-fragment)	1683
Notification processing time on GATT Server 244 bytes with DLE (1-fragment)	885
Notification processing time on GATT Server 495 bytes with DLE (2-fragment)	2340
Notification processing time on GATT Client 20 bytes without DLE (1-fragment)	480
Notification processing time on GATT Client 244 bytes with DLE (1-fragment)	806

BLE Stack Changes

This section lists changes made to the BLE Stack.

Version	Description of Changes	Reason for Changes / Impact
5.0.0.892	Updated to rerun the scheduler on close CE, if the event about to be scheduled is in past.	Blocking flash write operation blocks the CPU from executing any interrupts (including the BLESS isr) for > 17ms. Due to this, if the CI for connection is smaller than 17ms, the BLE link will disconnect.
	Modified the calibration target value for the Max output power level under the user-defined condition.	This is an enhancement to support the 5dBm Tx power in the BLE stack.
5.0.0.855	Changed the input parameter for the Cy_BLE_SetCustomEventMask() API from (uint8* mask) to (uint32 mask).	Consistent with the bit mask scheme.
	1. The timer context was never reset, it is made to reset during the Stack shutdown. 2. The timer context was not freed even when the timer start failed. It is made free when the timer start fails due to any reason.	The Cy_BLE_GAPC_StopScan() function returns CY_BLE_ERROR_MAX when the Cy_BLE_GAPC_StopScan() function is called to stop a previously-initiated scanning procedure. This issue occurs only when the authentication procedure is occurring and the application tries to stop the scan.
	1. The DLE control procedure timer is started only if the queuing Data Length Request in the Hardware FIFO is successful. 2. Added checks to handle the race condition between the ACK processing and Handling data PDU's from the Host.	Unexpected disconnection due to an LMP Response timeout in the embedded stress application. This issue is due to: 1. The DLE control procedure timer was started even if the sending Data Length Request PDU failed when queuing it in the Hardware FIFO. 2. Due to the race condition between the ACK processing and Handling data PDU's from the Host.
5.0.0.785	Initial BLE Stack version.	

Component Changes

This section lists the major changes in the Component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
2.0.c	Added the BLE user configuration file *cy_ble_config.h*.	To make BLE configuration more flexible. Allows redefining the configuration #define(s) generated by the BLE customizer and default BLE clock defines (from *cy_ble_clk.h*).
	Updated the BLE Interrupt Notification Feature.	Added support for BLE Dual mode.
	Updated the following functions: Cy_BLE_GATTS_ReadAttributeValueLocal() Cy_BLE_GATTS_ReadAttributeValuePeer()	These functions had the wrong Locally/Peer initiate operation flag.
	Deprecated the following defined values for the CY_BLE_CONFIG_STACK_MODE configuration parameter: CY_BLE_CONFIG_STACK_HOST_ONLY CY_BLE_CONFIG_STACK_DEBUG_UART CY_BLE_CONFIG_STACK_HOST_IPC.	Deprecated not supported (debug) modes.
	Renamed the following defined values: CY_BLE_CONFIG_STACK_DEBUG to CY_BLE_STACK_MODE_SINGLE_SOC CY_BLE_CONFIG_STACK_RELEASE to CY_BLE_STACK_MODE_DUAL_IPC.	The defines CY_BLE_CONFIG_STACK_DEBUG and CY_BLE_CONFIG_STACK_RELEASE were renamed to have more meaningful names.
	Updated the datasheet.	Updated the BLE Interrupt Notification Callback section. Updated the Quick Start section. Added a note about using the BLE in pair with Em_EEPROM. Removed the Component Errata section.
	Updated the BLE Stack to version 5.0.0.892.	See BLE Stack Changes.
2.0.b	Updated the datasheet.	Added the Multi-Connection Support section. Added the BLE Interrupt Notification Callback section. Updated the DC and AC Electrical Characteristics section. Added description for Over-The-Air bootloading with code sharing.

Version	Description of Changes	Reason for Changes / Impact
	Added new API: • Cy_BLE_IsDevicePaired(), • Cy_BLE_GetDeviceRole(), • Cy_BLE_IsPeerConnected(), • Cy_BLE_GATTS_WriteAttributeValuePeer(), • Cy_BLE_GATTS_WriteAttributeValueLocal(), • Cy_BLE_GATTS_ReadAttributeValuePeer(), • Cy_BLE_GATTS_ReadAttributeValueLocal(), • Cy_BLE_GATTS_IsIndicationEnabled(), • Cy_BLE_GATTS_SendIndication(), • Cy_BLE_GATTS_IsNotificationEnabled(). • Cy_BLE_GATTS_SendNotification(), • Cy_BLE_GATTS_SendErrorRsp(), • Cy_BLE_GATTC_SendConfirmation()	Improve usability
	Updated the BLE Stack to version 5.0.0.855.	See BLE Stack Changes.
2.0.a	Updated the datasheet.	Added description for new profiles: Automation IO and Pulse Oximeter. Update Quick Start section. Update MISRA Compliance section.
	Added new options for the CPU Core parameter.	Support of single core devices.
	Link Layer parameters moved to the separate tab in the customizer.	Update screens; Added description for Link Layer Settings
	Made the maximum number of connections configurable.	SRAM consumption reducing when extra connections are not used.
	Renamed API: • Cy_BLE_GAP_UpdateAdvScanData() to Cy_BLE_GAPP_UpdateAdvScanData(); • Cy_BLE_SetConnectionPriority() to Cy_BLE_GAP_SetConnectionPriority().	Consistent API naming scheme.
	Updated the BLE Stack to version 5.0.0.785.	See BLE Stack Changes.
2.0	Updated underlying PDL driver version. Updated the datasheet.	Update screens; Added description for Link Layer Settings
1.0.a	Updated datasheet.	Clarified information about IAR compiler. Added note for General discovery mode. Added Quick Start section.
1.0	Initial document for PDL version of the BLE Component.	
	Initial BLE Stack version 5.0.0.	

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.