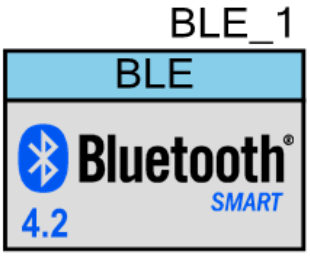
**Please note that Cypress is an Infineon Technologies Company.**
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

# Bluetooth Low Energy (BLE)

3.53




## Features

- Bluetooth v4.2 compliant protocol stack
- Generic Access Profile (GAP) Features
  - Broadcaster, Observer, Peripheral and Central roles
  - Supports role reversal between Peripheral and Central
  - User-defined advertising data
  - Bonding support for up to four devices
  - Security modes 1 and 2
- Generic Attribute Profile (GATT) Features
  - GATT Client and Server
  - 16-, 32-, and 128-bit UUIDs
- Special Interest Group (SIG) adopted GATT-based Profiles and Services, and quick prototype of new profile design through intuitive GUI Custom Profile development; Support of Bluetooth Developer Studio Profile format
- Security Manager features
  - Pairing methods: Just works, Passkey Entry, Out of Band, Numeric Comparison
  - Authenticated man-in-the-middle (MITM) protection and data signing
- Logical Link Adaption Protocol (L2CAP) Connection Oriented Channel
- Link Layer (LL) Features
  - Master and Slave role
  - 128-bit AES encryption
  - Low Duty Cycle Advertising
  - LE Ping

# General Description

The Bluetooth Low Energy (BLE) Component provides a comprehensive GUI-based configuration window to facilitate designing applications requiring BLE connectivity. The Component incorporates a Bluetooth Core Specification v4.2 compliant protocol stack and provides APIs to enable user applications to access the underlying hardware via the stack.

## When to use the BLE 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 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.

The component can also support several Profiles at a time by adding the required Services of a Profile to a base Profile. For example, you can select HID as a base Profile. Then to add a Find Me Profile, add the Immediate Alert Service to the HID Profile.

See BLE Service-Specific APIs for a list of supported Profiles and Services.

## Comprehensive APIs

The BLE Component provides application-level APIs to design solutions without requiring manual stack level configuration. The BLE Component API documentation is also provided in a separate HTML-based file.

## Custom Profiles

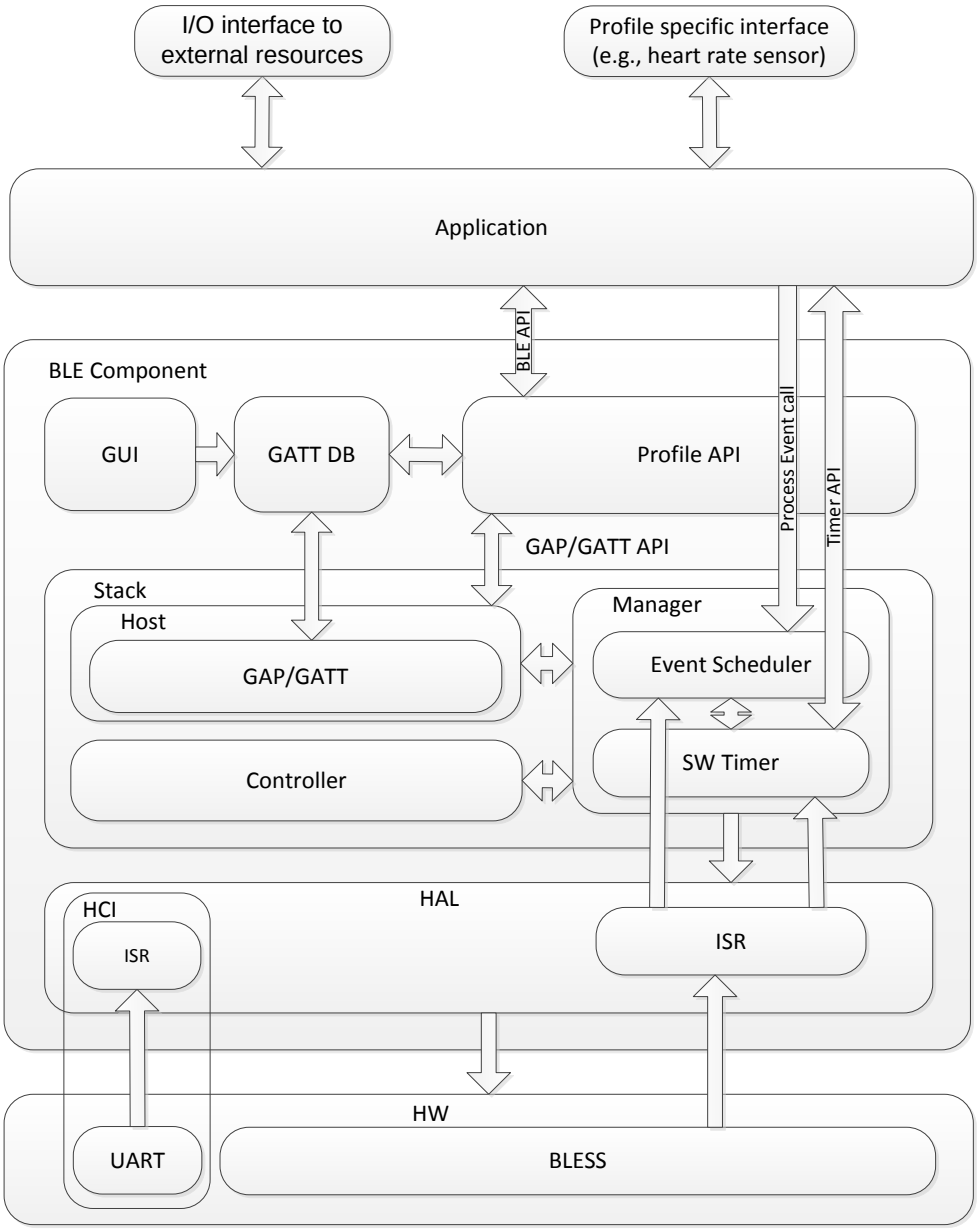
You can create custom Profiles that use existing Services, and you can create custom Services with custom Characteristics and Descriptors. There are no restrictions for GAP roles for a custom Profile.

## Debug Support

For testing and debugging, the Component can be configured to HCI mode through a Component embedded UART. For over-the-air verification, Cypress CySmart Central Emulation Tool can be used for generic Bluetooth host stack emulation. To launch this tool, right click on the Component to bring up the context menu, and choose to deploy the CySmart Central Emulation Tool.

## BLE Component Architecture

The BLE Component consists of the BLE Stack, BLE Profile, BLE Component Hardware Abstraction Layer (HAL), and the Link Layer. The following figure shows a high-level architecture of the BLE 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.

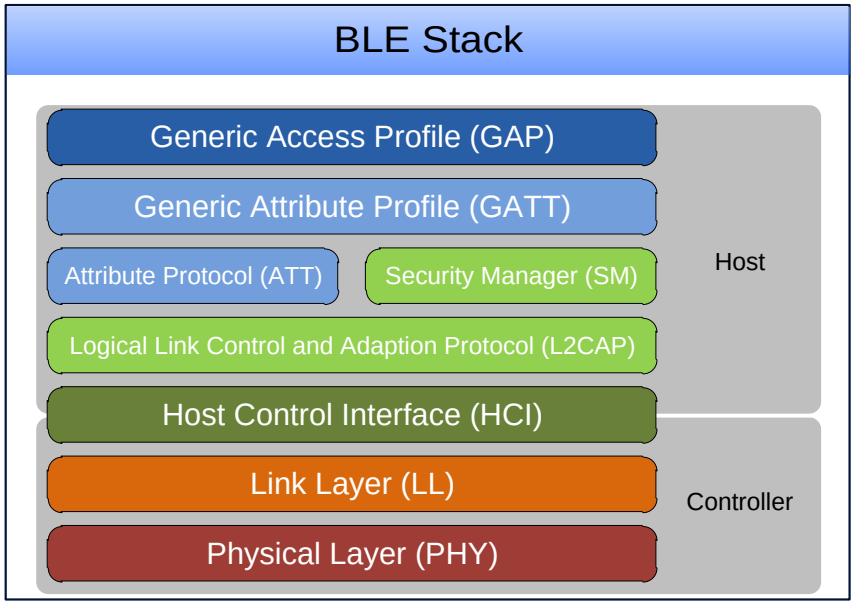
### BLE Stack

The BLE stack implements the core BLE functionality as defined in the Bluetooth Core Specification 4.2. The stack is included as a precompiled library and it is embedded inside the BLE Component.

The BLE stack implements all the mandatory and optional features of Low Energy Single Mode compliant to Bluetooth Core Specification 4.2. The following table shows which Bluetooth Core Specification 4.2 features are supported by different devices.

| Features | Devices with Bluetooth 4.1 | Devices with Bluetooth 4.2 |
|---|---|---|
| LE Secure connection | ✓ | ✓ |
| LL Privacy | - | ✓ |
| LE Data Length Extension | - | ✓ |

The BLE Stack implements a layered architecture of the BLE protocol stack as shown in the following figure.




*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 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 Profile Role in Profile Mode or HCI Mode. Refer to the Generic Tab section for selection of stack configuration.

| BLE Component Configuration | GAP Role | BLE Stack Library |
|---|---|---|
| BLE Profile | Central + Peripheral | CyBLEStack_BLE_SOC_CENTRAL_PERIPHERAL.a |
| BLE Profile | Central | CyBLEStack_BLE_SOC_CENTRAL.a |
| BLE Profile | Peripheral | CyBLEStack_BLE_SOC_PERIPHERAL.a |
| Broadcaster/Observer | Broadcaster | CyBLEStack_BLE_SOC_PERIPHERAL.a |
| Broadcaster/Observer | Observer | CyBLEStack_BLE_SOC_CENTRAL.a |
| HCI Mode | N/A | CyBLEStack_HCI_MODE_CENTRAL_PERIPHERAL.a |

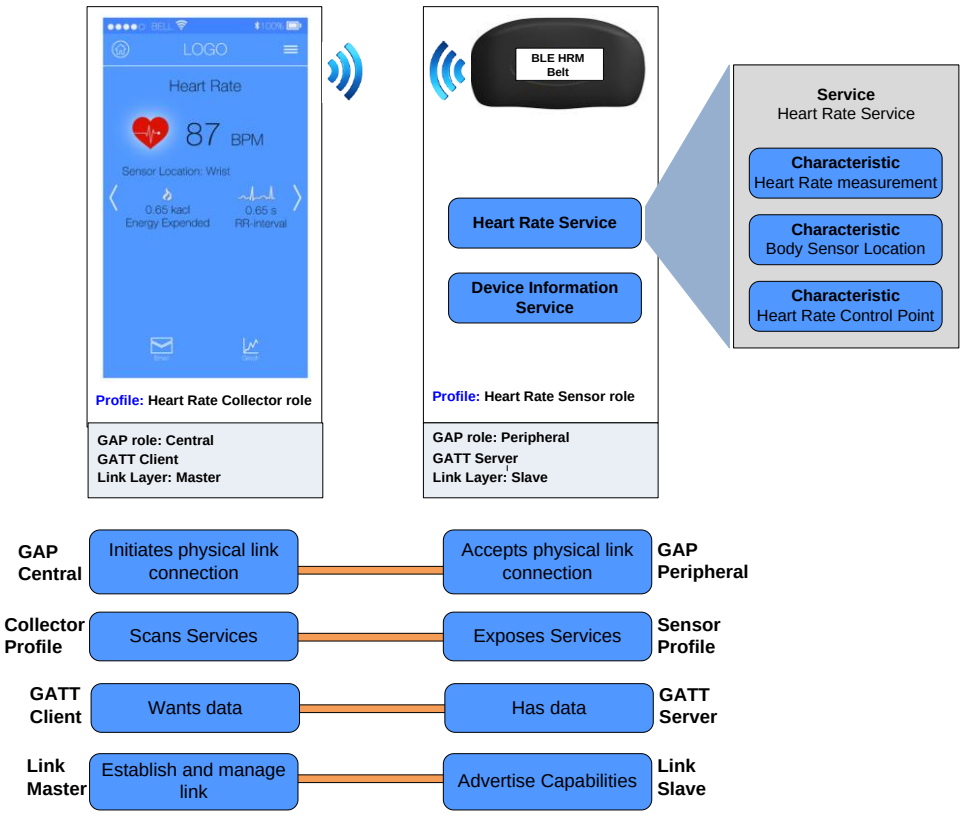
### Profile Layer

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 Profile parameter for how to configure the BLE 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.

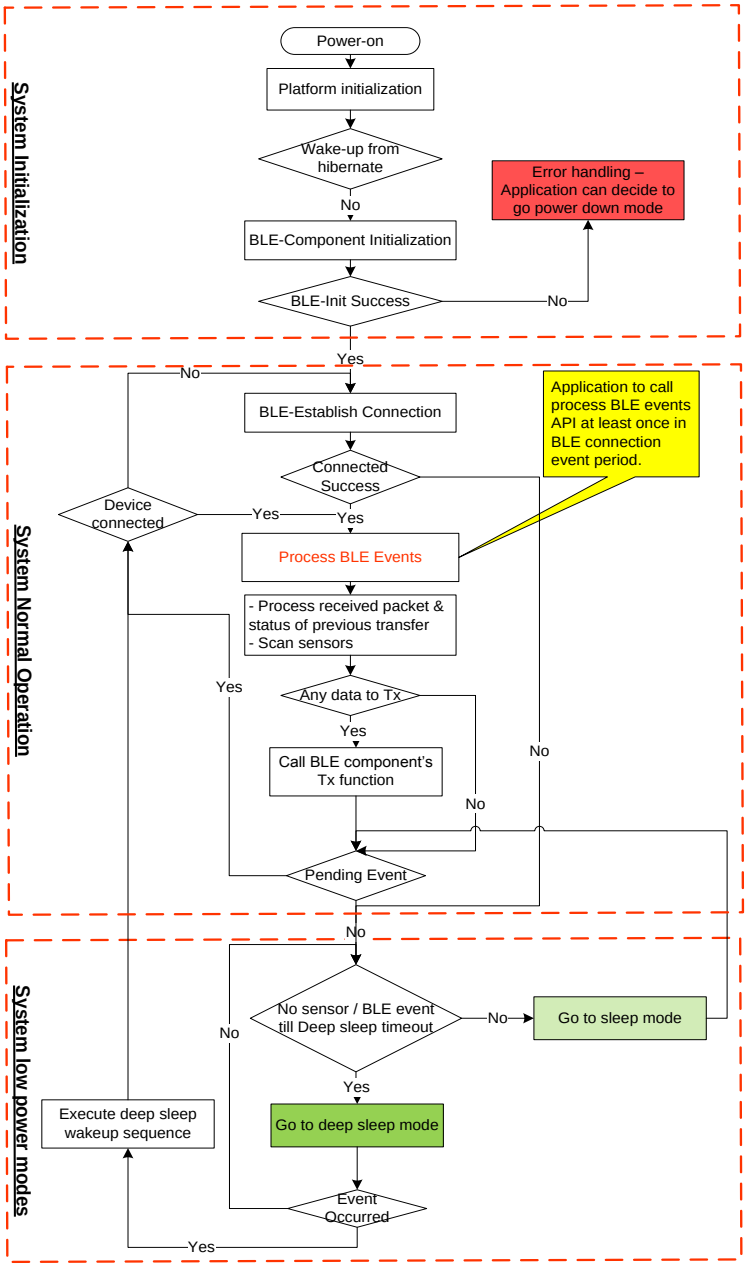
### Hardware Abstraction Layer (HAL)

The HAL implements the interface between the BLE stack and the underlying hardware. This layer is meant for the stack only and 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.




Once the Component is initialized, it enters normal operation and periodically enters various degrees of low power operation to conserve power. Hence 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 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 CyBle_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 DeepSleep 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 | PSoC 4200-BL, PRoC 4200-BL MCUs Power Modes | | | | |
|---|---|---|---|---|---|
| | Active | Sleep | Deep Sleep | Hibernate | Stop |
| Active (idle/Tx/Rx) | ✓ | ✓ | | | |
| Sleep | ✓ | ✓ | | | |
| Deep Sleep (ECO off) | ✓ | ✓ | ✓ | | |
| Off | | | | ✓ | ✓ |

## Callback Functions

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

```
void (* CYBLE_CALLBACK_T)(uint32 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. Refer to the BLE Common Events section for the BLE stack events.

The BLE stack triggers the application event handler callback for any pending events generated by the link layer, after calling the CyBle_ProcessEvents() API method. However, other BLE 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 CYBLE_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. Refer to the BLE Service-Specific Events section for the BLE Service-specific events.

## Device Bonding

The BLE 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 CyBle_StoreBondingData() API to write the bonding data from RAM to the dedicated Flash location, as defined by the Component. Refer to the BLE_HID_Keyboard example project for usage details.

### Notes

- For BLE devices with 128 K of Flash memory, the Flash write modifies the IMO of the chip to 48 MHz temporarily during the write cycle. Therefore, you should only perform the bonding data Flash storage while the BLE devices are disconnected, because the change in IMO may disrupt the BLE communication link. Likewise, you should either temporarily halt all peripherals running off of the IMO or compensate for the brief frequency change during the Flash write cycle.

- If BLE device with 128 K of Flash memory, is configured to run at 48 MHz, then the IMO does not change and does not affect other peripherals. However, the Flash write is a blocking call and may disrupt the BLE communication. Therefore, it is advisable to perform the Flash write while the devices are disconnected.

## LFCLK configuration

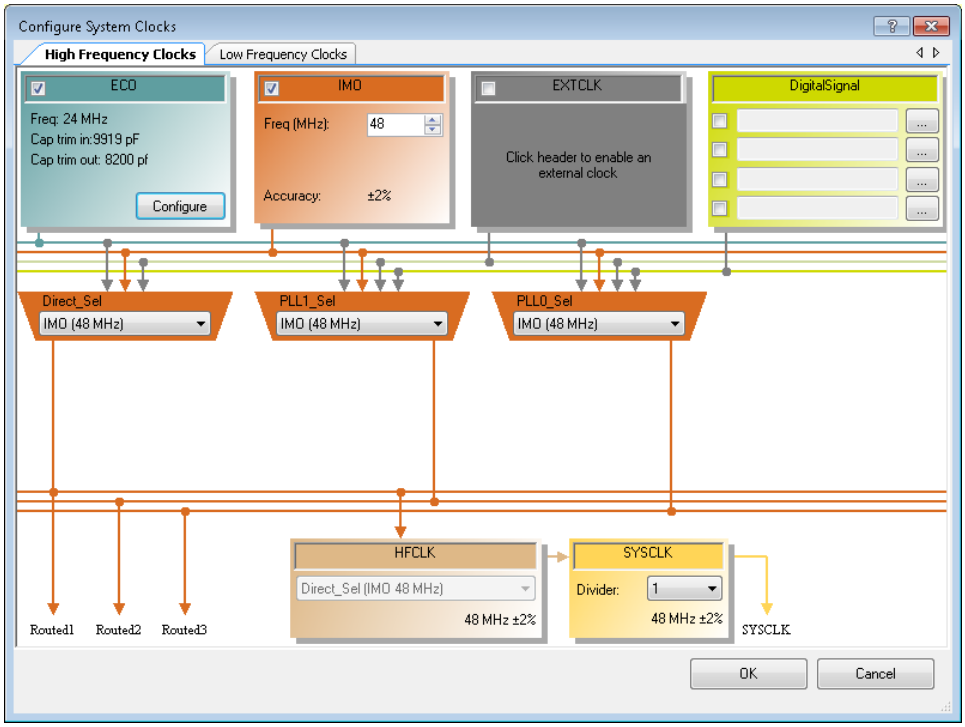
The LFCLK configuration as set in the **Clocks** tab of the Design-Wide Resources (*<project>.cydwr*) file affects the BLE Component's ability to operate in Deep Sleep Mode. If the WCO is chosen, then the Component Deep Sleep Mode is available for use. However, if the ILO is chosen, then the Component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

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

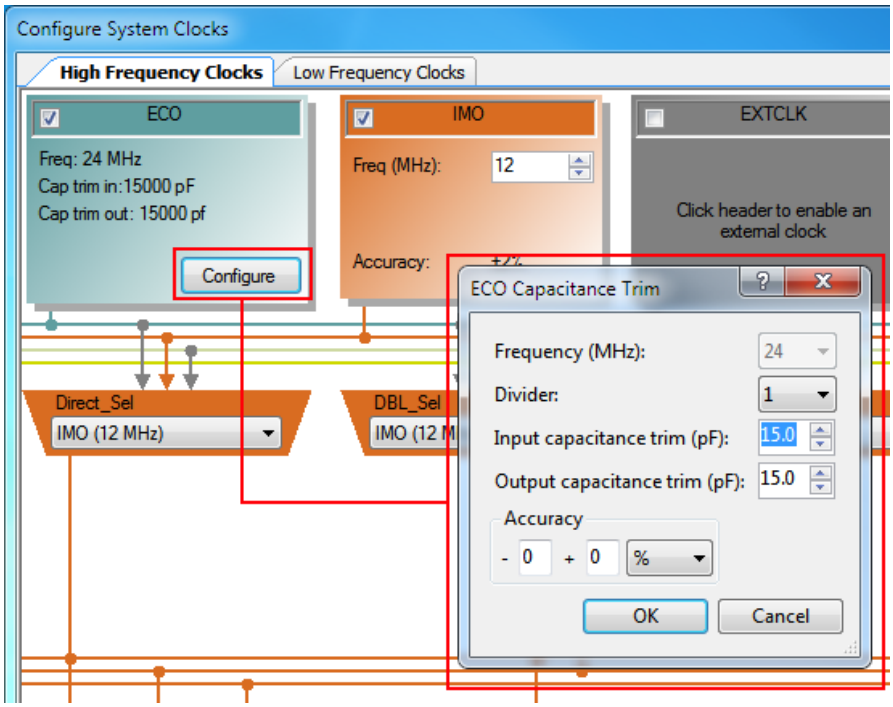
## ECO capacitance trim

ECO capacitance trim is required to supply the correct capacitance load to achive 0 ppm for the ECO. By default, the ECO capacitance trim values are set to the trim values of PSoC 4 BLE / PRoC® 4 BLE modules. If any other BLE module is used (such as, EZ-BLE module), the trim values should be changed to the values provided by that module's datasheet.

1. If needed, to configure the ECO capacitance trim, open the Design-Wide Resources Clock Editor (*<project>.cydwr*).
2. Click **Edit Clock** To open the Configure System Clocks dialog.



3. Under the **High Frequency Clocks** tab, in the **ECO** section, click the **Configure…** button to open the ECO Capacitance Trim dialog.



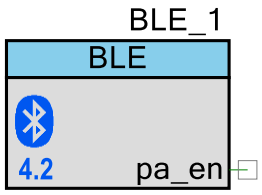

## Unsupported Features

The BLE Component stack does not support the following optional Bluetooth v4.2 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)

# 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_en – Output *

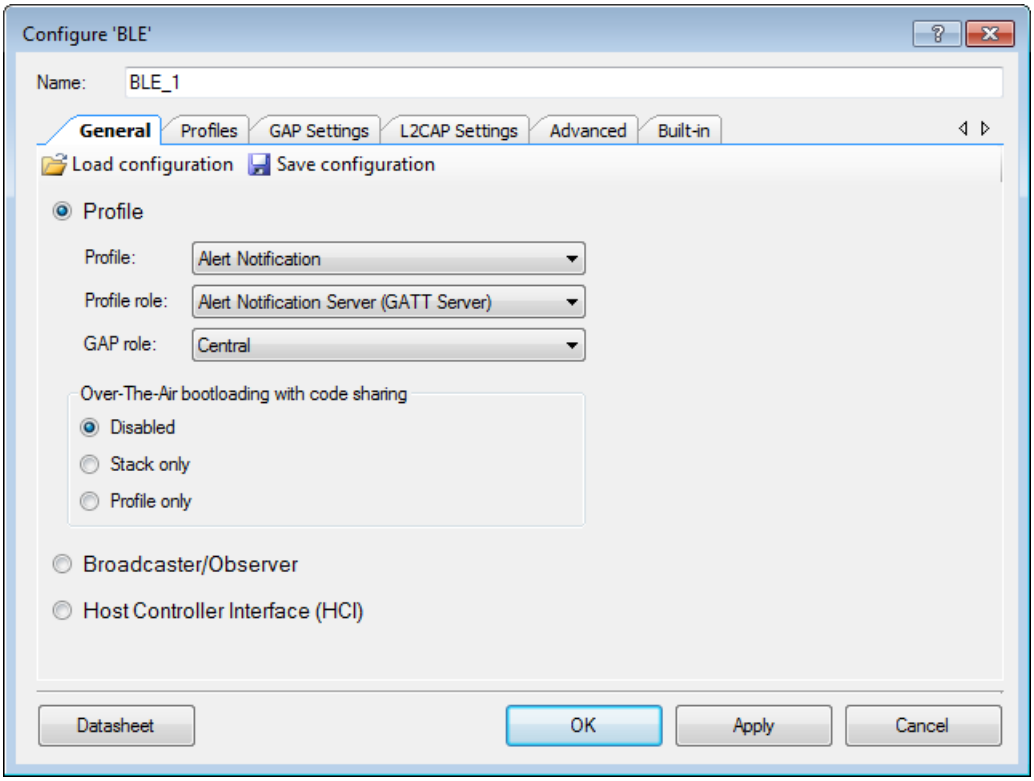
The power amplifier enable (pa_en) output allows you to connect a high active external power amplifier to the device. This output can be routed to the P5[0] digital output pin only. This output is visible if the **Enable external Power Amplifier control** parameter is selected on the **Advanced** tab.

# Component Parameters

Drag a BLE 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 Component. This tab contains tools to load and save configurations as also three main areas for the type of configuration.




### 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 **Profiles** tab.

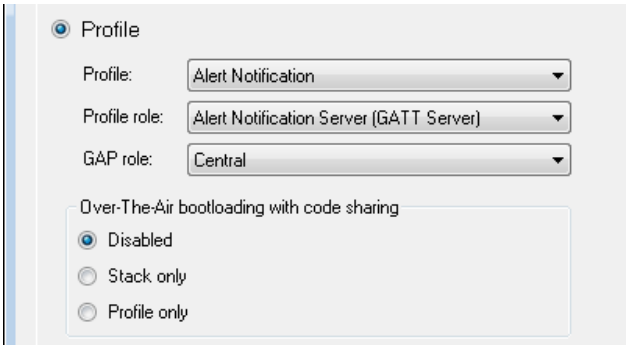
### Mode Selection

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

- Profile
- Broadcaster/Observer
- Host Controller Interface

## General Tab – Profile

The Profile mode is used to select the target Profile, Profile role, and GAP role, as well as Over-The-Air (OTA) Bootloading options.




### Profile

The **Profile** option is used to choose the target Profile from a list of supported Profiles. See Profile, Service, and Characteristic. 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**.
  - Central GAP role
  - Peripheral and Central GAP role
- **Alert Notification Client** Profile role – Specified as a GATT Client.
  - Peripheral GAP role
  - Peripheral and Central GAP role

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**.
  - Peripheral GAP role
- **Automation IO Client** Profile role – Specified as a GATT Client.
  - Central GAP role
  - Peripheral and Central GAP role

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**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role

- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role
- **Cycling Power Sensor and Broadcaster** Profile role. Requires the following Service: **Cycling Power Service**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles
- **Cycling Power Observer** Profile role. Can only talk to a device with the **Cycling Power Broadcaster** role.
  - Central GAP 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**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role
  - Central GAP role
  - Peripheral and Central GAP roles
- **Find Me Locator** Profile role – Specified as a GATT Client. Requires support of the following Service: **Immediate Alert Service**.
  - Peripheral GAP role
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Glucose Service**. Support of **Device Information Service** is optional.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Peripheral GAP role
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Health Thermometer Service**. Support of **Device Information Service** is optional.
  - Central GAP role
  - Peripheral and Central GAP roles

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 **Profile** drop-down list. It can be added to **Custom Profile** (or other) on the **Profiles** tab.

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**.
  - Peripheral GAP role
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Service: **Heart Rate Service**. Support of **Device Information Service** is optional.
  - Central GAP role
  - Peripheral and Central GAP roles

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.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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 to the Custom Profile (or other) on the **Profiles** tab.

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

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**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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**.
  - Central GAP role
  - Peripheral and Central GAP role
- **Phone Alert Client** Profile role – Specified as a GATT Client. Requires support of the following Service: **Phone Alert Service**.
  - Peripheral GAP role
  - Peripheral and Central GAP role

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.
  - Peripheral GAP role
  - Central GAP role
- **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**.
  - Central GAP role
  - Peripheral GAP role
  - Peripheral and Central GAP role

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

*Pulse Oximeter*

This Profile is used to enable communications between a Pulse Oximeter (PLX) and a Collector. It contains guidance for finding, connecting to, receiving measurements from, and configuring a pulse oximeter that supports this profile.

- **Pulse Oximeter Sensor** Profile role – Specified as a GATT Server. Requires the following Services: **Pulse Oximeter Service** and **Device Information Service**. Optionally may include **Bond Management Service, Current Time Service** and/or **Battery Service**.
  - Peripheral GAP role
- **Collector** Profile role – Specified as a GATT Client. Requires support of the following Services: **Pulse Oximeter Service** and **Device Information Service**. Support of **Bond Management Service, Current Time Service** and/or **Battery Service** is optional.
  - Central GAP role
  - Peripheral and Central GAP roles

*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**.
  - Peripheral GAP role
- **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.
  - Central GAP role
  - Peripheral and Central GAP roles

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

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**.
  - Central GAP role
  - Peripheral and Central GAP role
- **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.
  - Peripheral GAP role
  - Peripheral and Central GAP role

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**.

  - Peripheral GAP role

- **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.

  - Central GAP role
  - Peripheral and Central GAP roles

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

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

*Custom*

This is used to create a custom Profile. It allows you to add a **Custom Service** and gives control over the Service types. Custom Services cannot be used in stand-alone mode; they need to be used in a Profile. For example, the Device Information Service is used in the Heart Rate Profile. It can be used in a custom Profile, or it can be added to any of existing Profiles.

**Note** The Apple Notification Center Service is not included into any Bluetooth SIG adopted Profiles, so it can be used only within custom Profile.

- **Server (GATT Server)** Profile role
  - Peripheral GAP role

- Central GAP role
- Peripheral and Central GAP roles

- **Client (GATT Client)** Profile role
  - Peripheral GAP role
  - Central GAP role
  - Peripheral and Central GAP roles
- **Client and Server (GATT Client and Server)** Profile role
  - Peripheral GAP role
  - Central GAP role
  - Peripheral and Central GAP roles

*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).

The Bootloader Service is designed to be used with the Cypress Bootloader/Bootloadable components and therefore it uses the characteristic structure compatible with the Bootloader component command format.

### Profile Role

The **Profile role** parameter configuration depends on the chosen **Profile**, and the **Profile role** selection affects the **GAP role** parameter. These parameters affect the options available on the **Profiles** tab.

- **GATT Server –** Defines the role of the device that contains a specific data in a structured form. The device in this role is usually a sensor that gets the data. The data is structured in the GATT database. BLE Profiles can introduce their own names to identify GATT Server device (e.g. Find Me Profile uses “Find Me Target”). GATT Server devices usually utilize the GAP Peripheral role.
- **GATT Client –** Defines the role of the device that generates requests to the GATT Server device to fetch data. BLE Profiles can introduce their own names to identify GATT Client device (e.g. Find Me Profile uses “Find Me Locator”). GATT Client devices usually utilize the GAP Central role.

- **Client and Server –** Defines the role of the device that concurrently can perform functionality of a GATT Client and Server Profile role. For example, a peripheral device can act as a GATT Client and start discovering the iOS device's (acting as GATT Server) Services (Battery, Time and Apple Notification Central Service).

### 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.
- **Peripheral and Central** – In this role, the application can perform role reversal between Peripheral and Central roles at run time. For example, Bluetooth Smart watch (Peripheral) can connect to a smartphone (Central device). The same sports watch can then switch to the Central device mode to obtain data from other Peripheral devices such as a heart rate monitor and a blood pressure sensor.

  **Note** The BLE device can also be configured to simultaneously support both Peripheral and Broadcaster or Central and Observer roles. This option is not exposed in the GUI, but can be dynamically configured in the firmware. Refer to the BLE Cycling Sensor code example for an implementation of simultaneous Peripheral and Broadcaster roles.

### 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 only** – When this option is selected, the component represents only the stack portion of BLE along with a Bootloader Service. It is used to isolate the stack from the profiles.

  The **Stack only** mode is used in the BLE OTA Upgradable Stack Example.

  **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 property can be modified in the PSoC Creator Design-Wide Resources System Editor. See the PSoC Creator Help for more information.

- **Profile only** – This option makes the component only have the profile-specific code. Stack is excluded.
  - Stack dependency – This field allows you to associate a **Profiles only** project with the **Stack only** project. Each project configured in the **Stack only** mode during the build generates the .CYCSA file located in the *Generated_Source* project folder. This file needs to be referenced from the **Profiles only** project using this field.

## General Tab – Broadcaster/Observer

The **Broadcaster/Observer** mode allows you to configure the device directly into one of the non-connectable GAP roles that does not require a Profile definition.

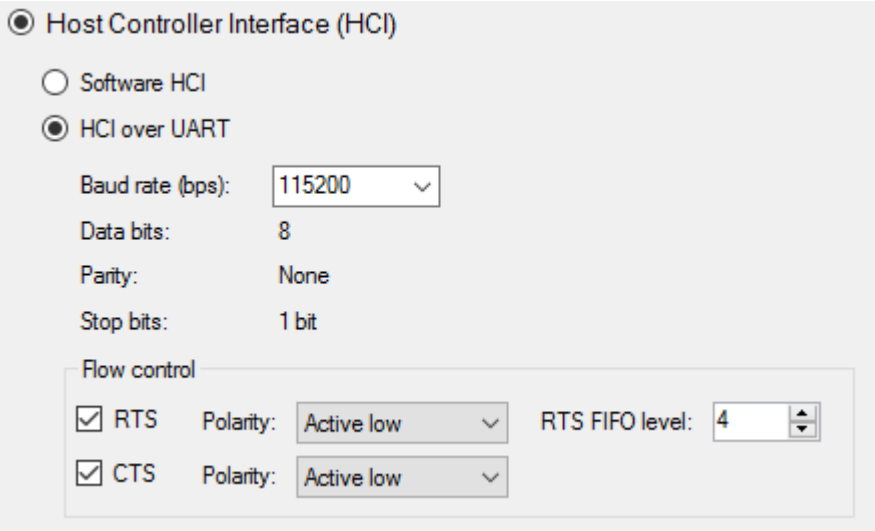



Two GAP roles are available for selection:

- **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.

## General Tab – Host Controller Interface

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 **Profiles** tab, **GAP Settings** tab, and **L2CAP Settings** tab become unavailable.




Two HCI interfaces are available for selection:

- **Software HCI**

  In this mode the BLE host will communicate with the BLE controller using the software transport.

  Use CyBle_HciSendPacket() to send the HCI commands to the BLE controller. The BLE controller will generate CYBLE_EVT_HCI_PKT event with the responses for HCI commands in the registered application callback. The same event should be used to receive commands from the remote devices in the HCI format.

- **HCI over UART**

  In this mode the BLE host will communicate with the BLE controller via the UART.

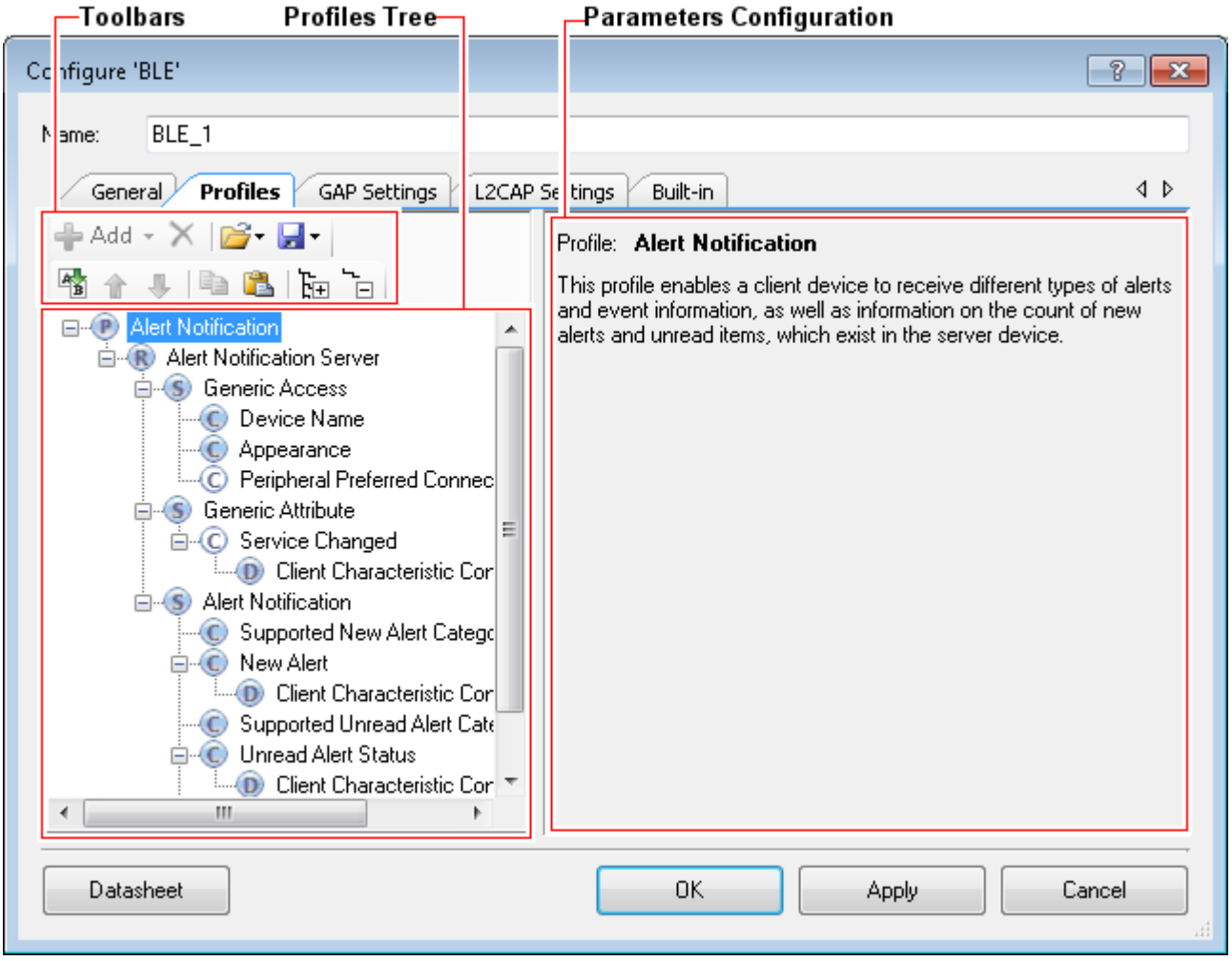
  The UART is a full-duplex 8 data bit, 1 stop bit, no parity with Flow control interface.

  - 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 is only available for PSoC 4100 BLE / PSoC 4200 BLE / PSoC 4100M / PSoC 4200M / PSoC 4200L / PSoC 4000S / PSoC 4100S / PSoC Analog Coprocessor devices. It 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: 4.
- 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.

## Profiles Tab

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




### Toolbars

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

- **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 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 Services, Characteristics, and Descriptors in the selected Profile. 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.

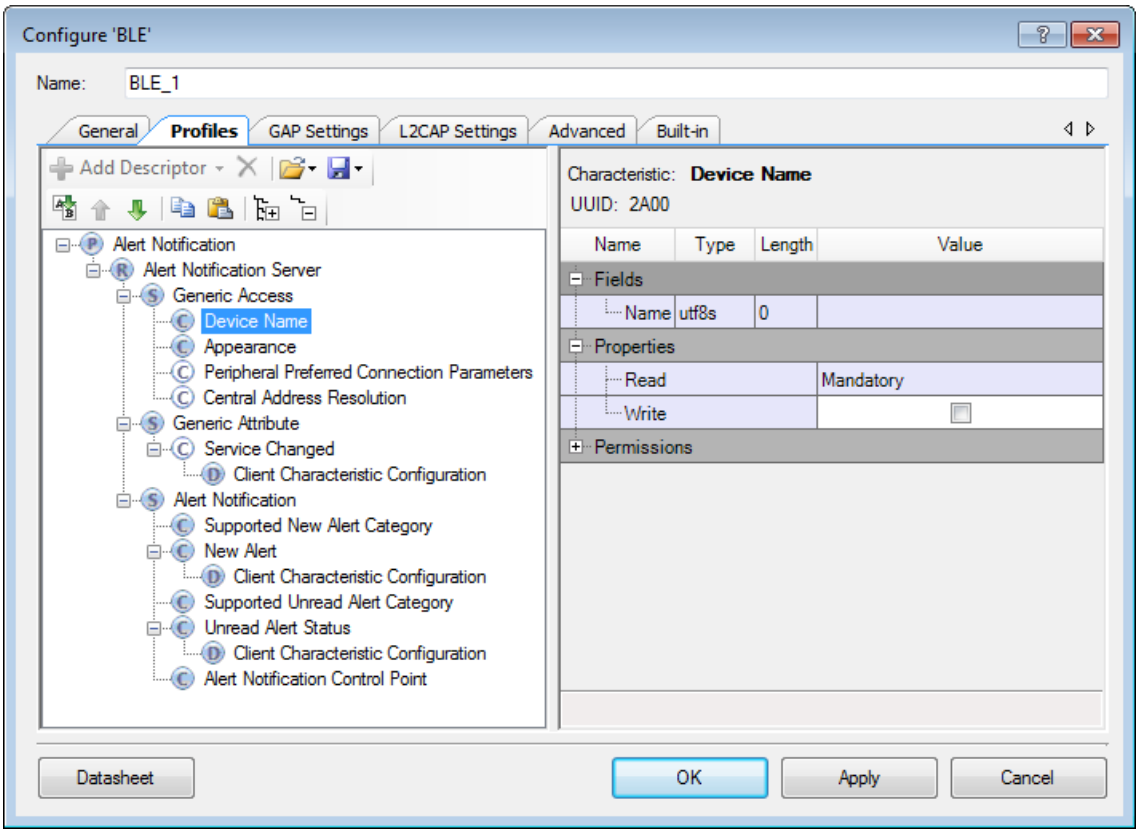
### Parameters Configuration

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

### Notes

- All Profiles must have a **Generic Access Service** and a **Generic Attribute Service**.
- The Service Characteristics are configurable only when the device is a GATT Server.
- 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*



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 is read-only and is derived from the **GAP Settings** tab, **Peripheral Preferred Connection Parameters** node. It 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.

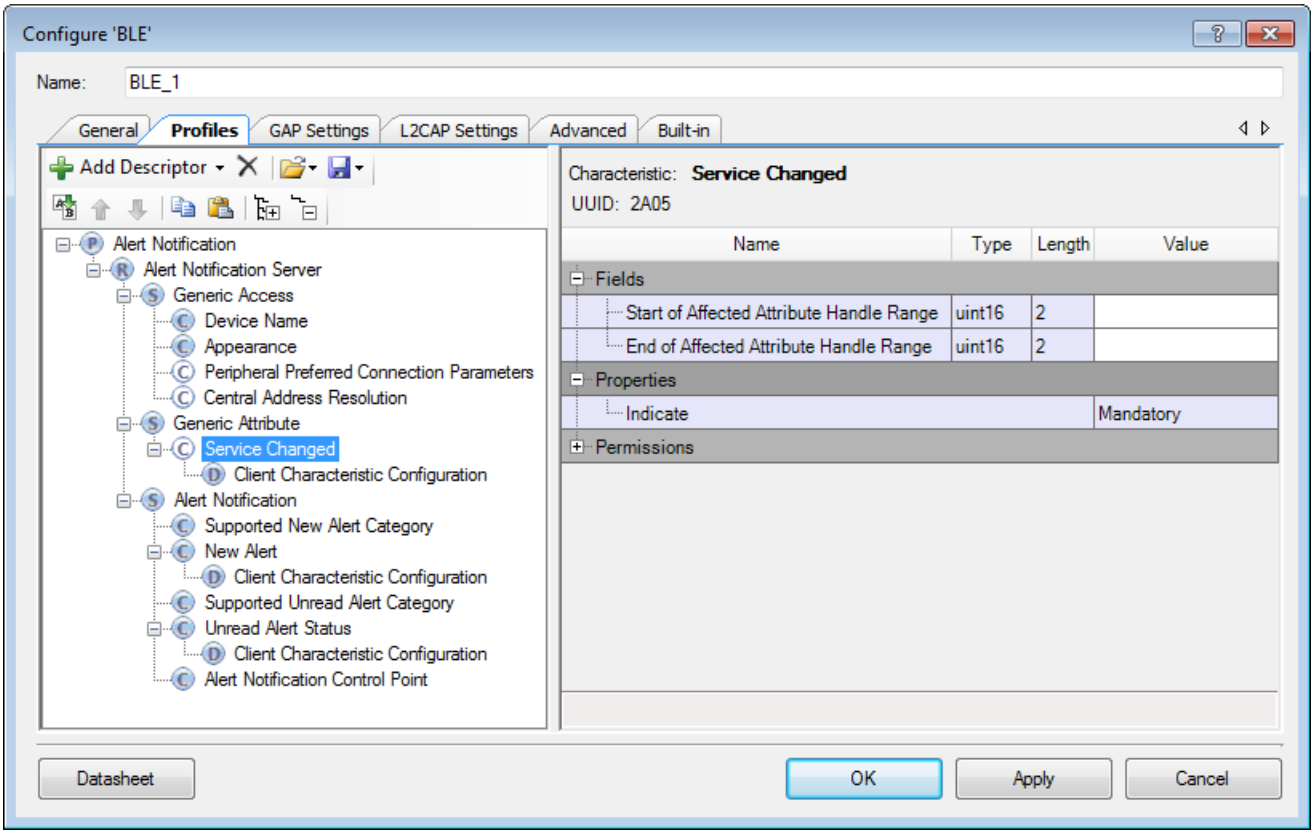
  - **Minimum Connection Interval:** This is a 2-byte parameter that denotes the minimum permissible connection time.
  - **Maximum Connection Interval**: This is a 2-byte parameter that denotes the maximum permissible connection time.
  - **Slave Latency**: This is a 2-byte value and defines the latency 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** 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).

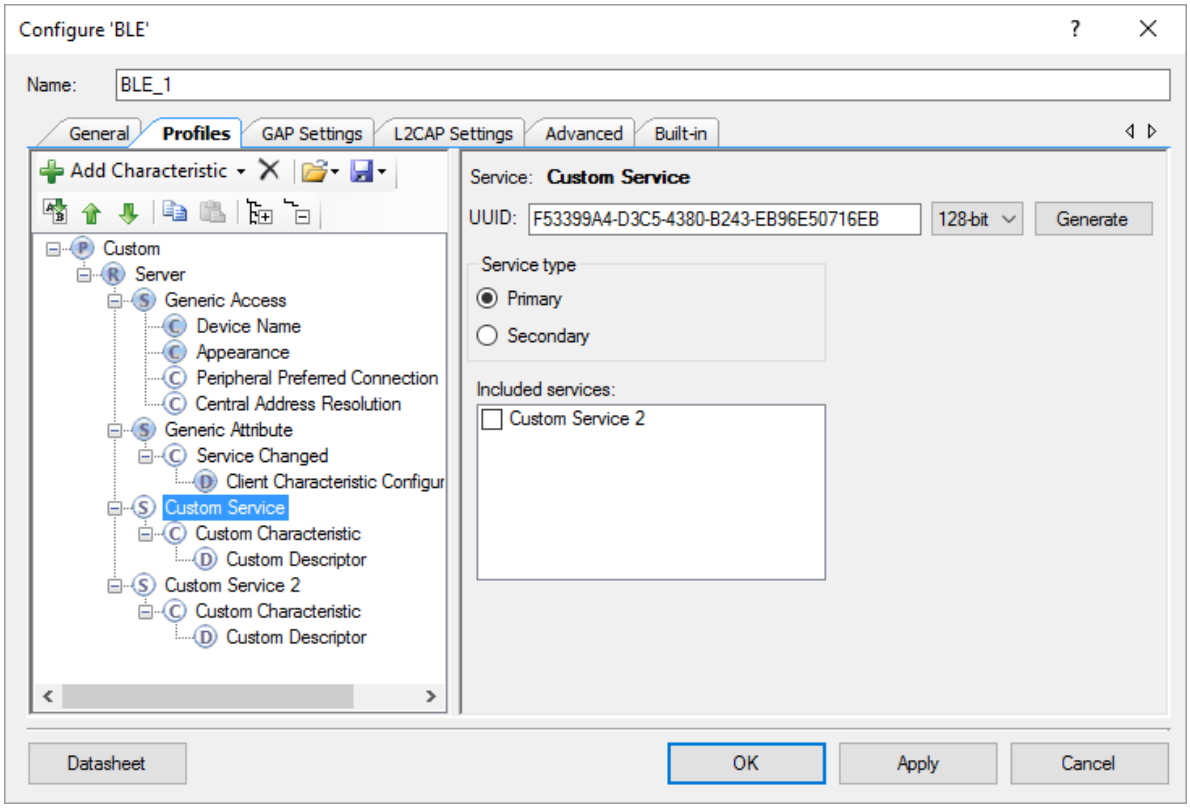
*Generic Attribute Service*




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




*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.

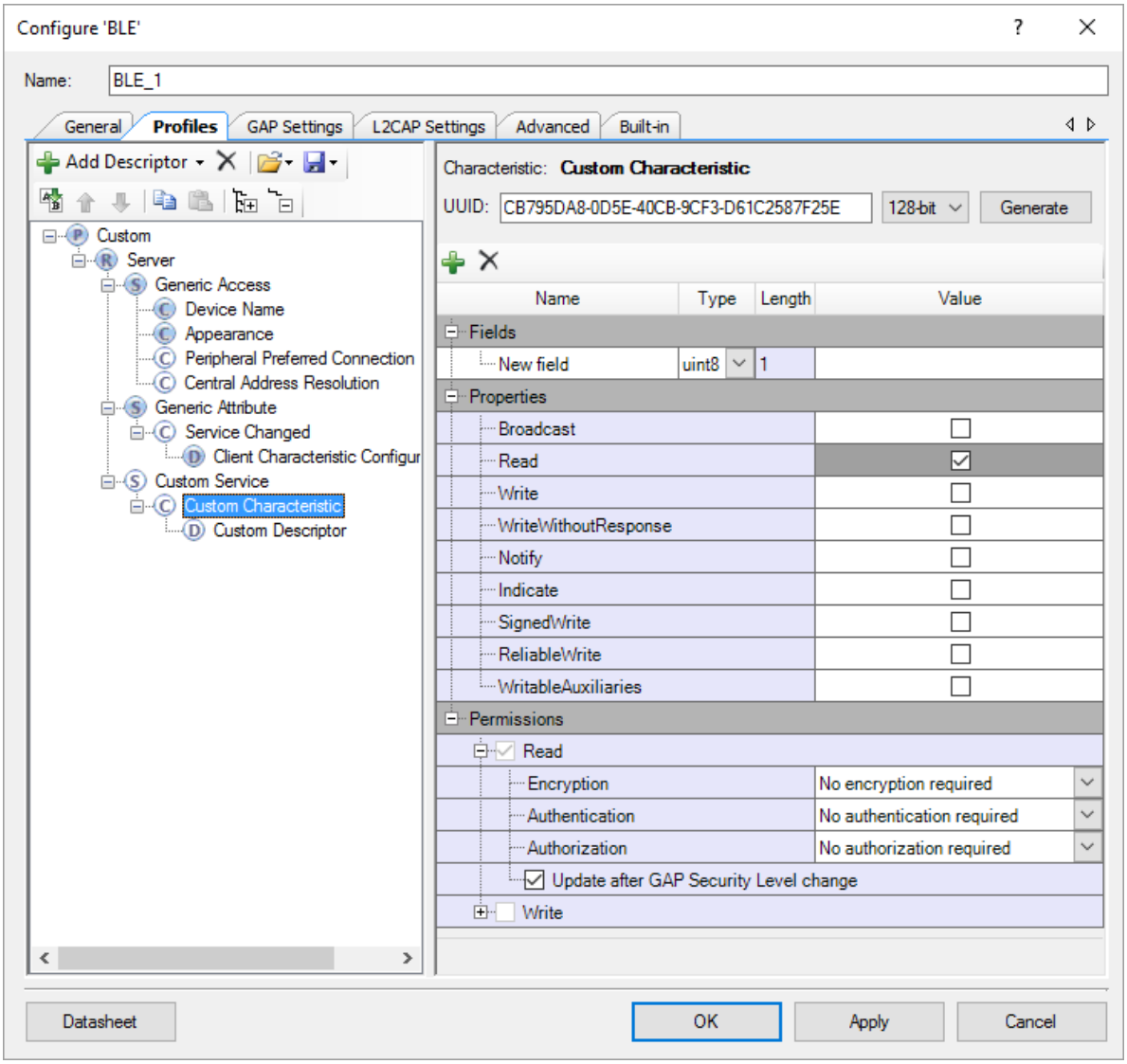
*Service type*

- **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




*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.

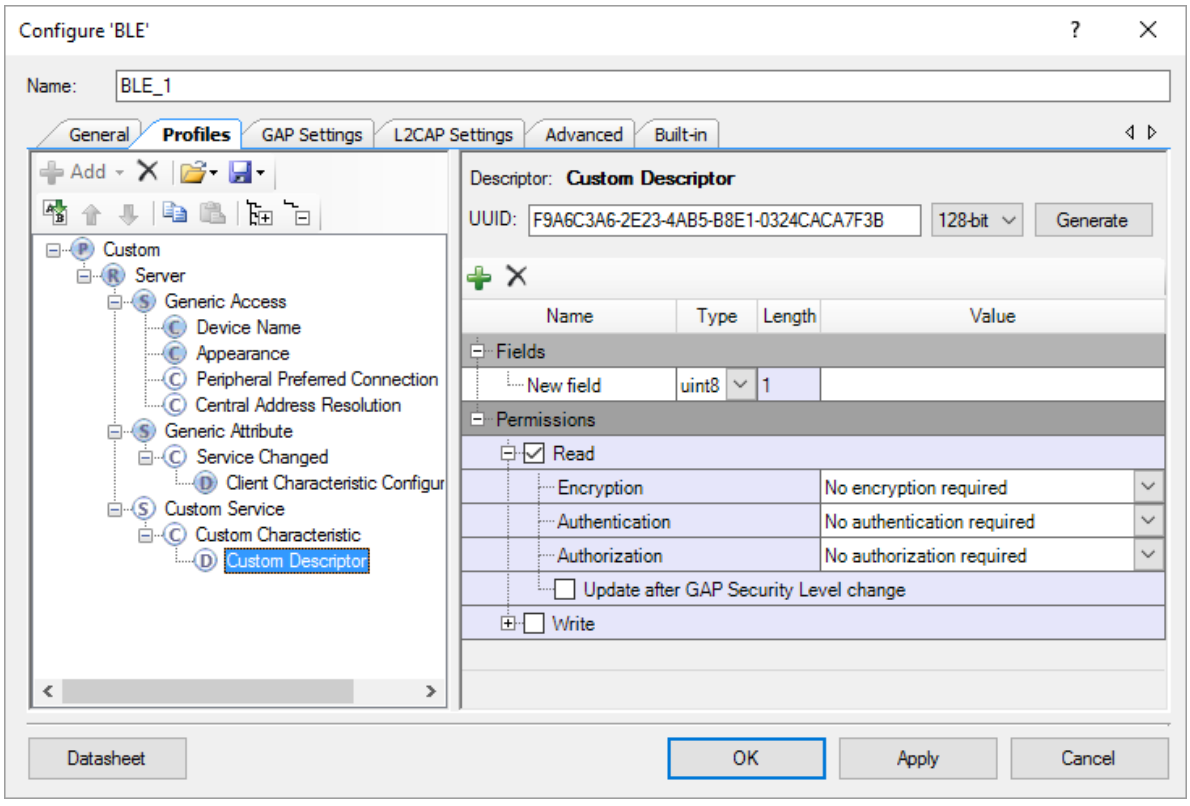
*Properties*

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 on the GAP Settings tab.

### Custom Descriptor Configuration




*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.

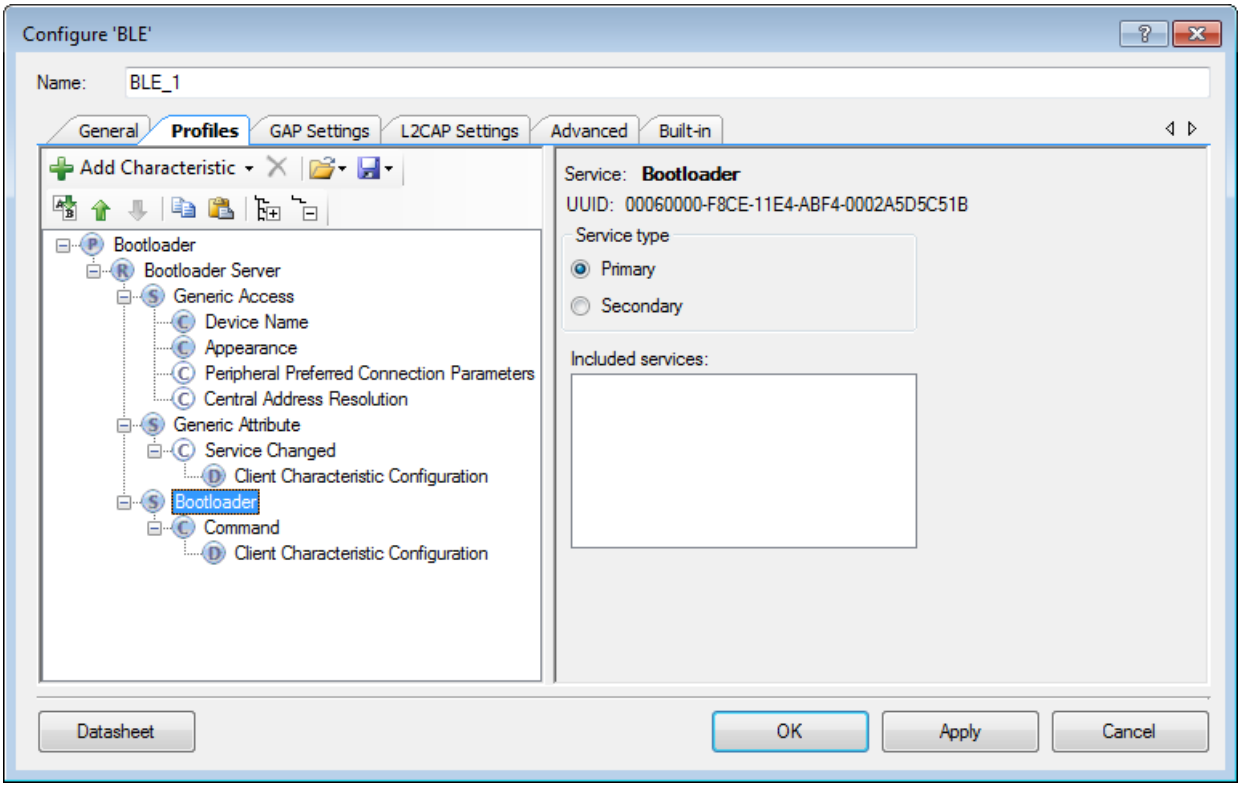
*Fields*

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




*UUID*

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

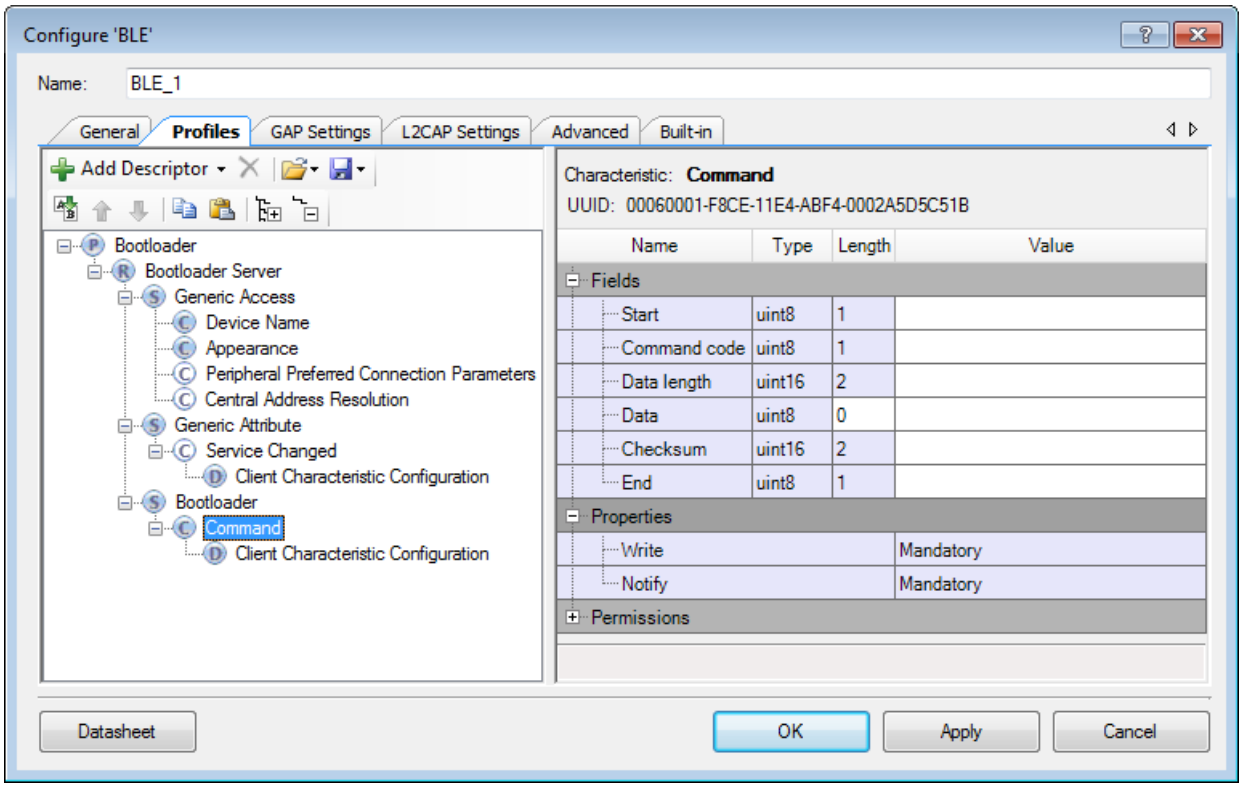
*Service type*

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

### Command Characteristic Configuration




*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 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

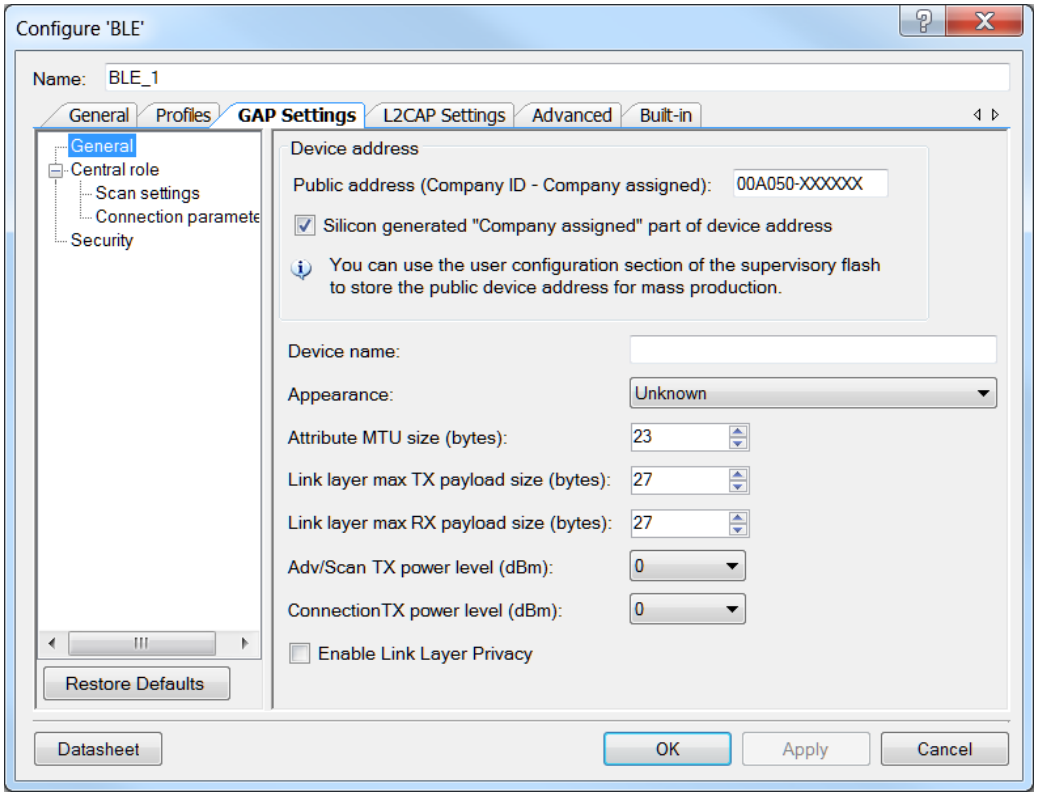
The GAP parameters define the general connection settings required when connecting Bluetooth devices. It contains various sections of parameters based on the item you select in the tree.

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.

### GAP Settings Tab – General

This section contains general GAP parameters:

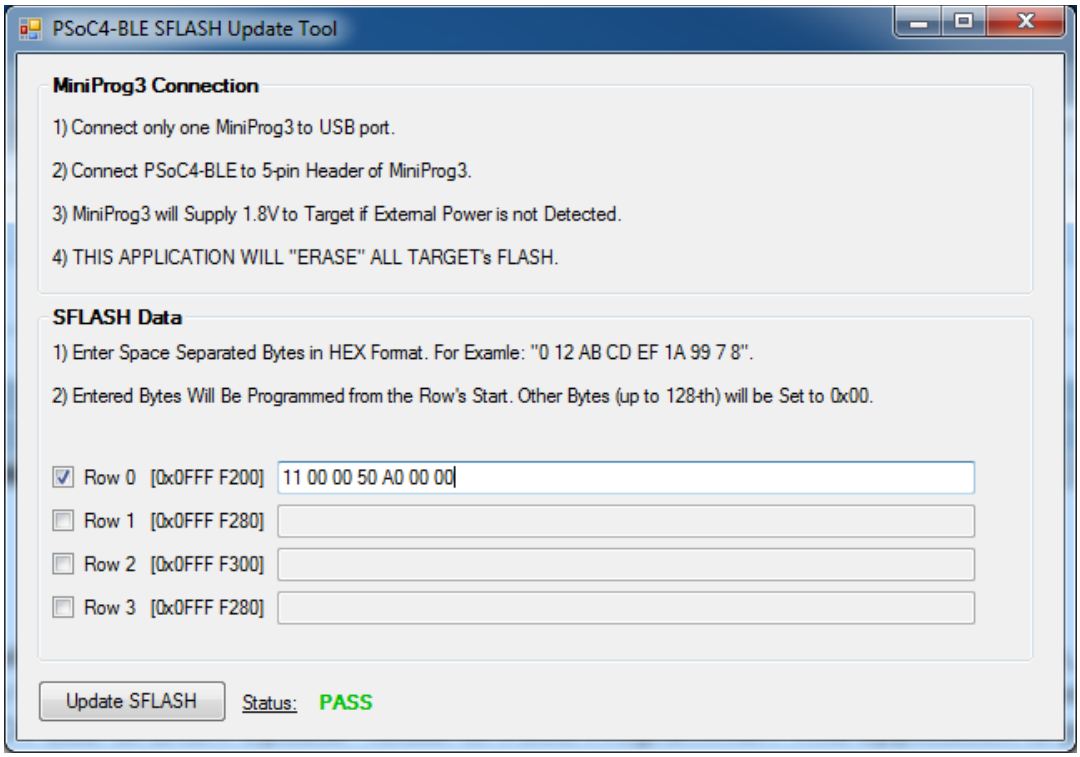



*Public device address (Company ID – Company assigned)*

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 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), device address can be programmed into the user's SFLASH location using MiniProg3 and the sample application installed in the "C:\Program Files (x86)\Cypress\Programmer\Example\Misc\PSoC4-BLE-SFLASH-Update\Executable" folder of PSoC Programmer. Enter device address structure of type CYBLE_GAP_BD_ADDR_T in the Row 0 line to store it in the SFLASH.




Row 1, Row 2 and Row 3 are not used by the component and available for user information storage. Note that row addresses and length (128 or 256 bytes) depend on the flash memory size of the selected device. Row 0 address is: 0x0FFF F200 for device with 128 KB Flash or 0x0FFF F400 for device with 256 KB Flash.

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.

*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.

*Link Layer Max Tx Payload Size*

The maximum link layer transmit 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 'CYBLE_EVT_GAP_DATA_LENGTH_CHANGE' event. Valid range is from 27 to 251 bytes. This option is available only for the devices supporting Bluetooth 4.2.

*Link Layer Max Rx Payload Size*

The maximum link layer receive 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 'CYBLE_EVT_GAP_DATA_LENGTH_CHANGE' event. Valid range is from 27 to 251 bytes. This option is available only for the devices supporting Bluetooth 4.2.

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.

*Adv/Scan TX power level*

The initial transmitter power level (dBm) of the advertisement or scan channels upon startup. Default: 0 dBm. Possible values: -18 dBm, -12 dBm, -6 dBm, -3 dBm, -2 dBm, -1 dBm, 0 dBm, 3 dBm.

*Connection TX power level*

The initial transmitter power level (dBm) of the connection channels upon startup. Default: 0 dBm. Possible values: -18 dBm, -12 dBm, -6 dBm, -3 dBm, -2 dBm, -1 dBm, 0 dBm, 3 dBm.

**Note** Due to hardware limitations, the 3 dBm value can be set only for both Adv/Scan TX power level and Connection TX power level simultaneously.

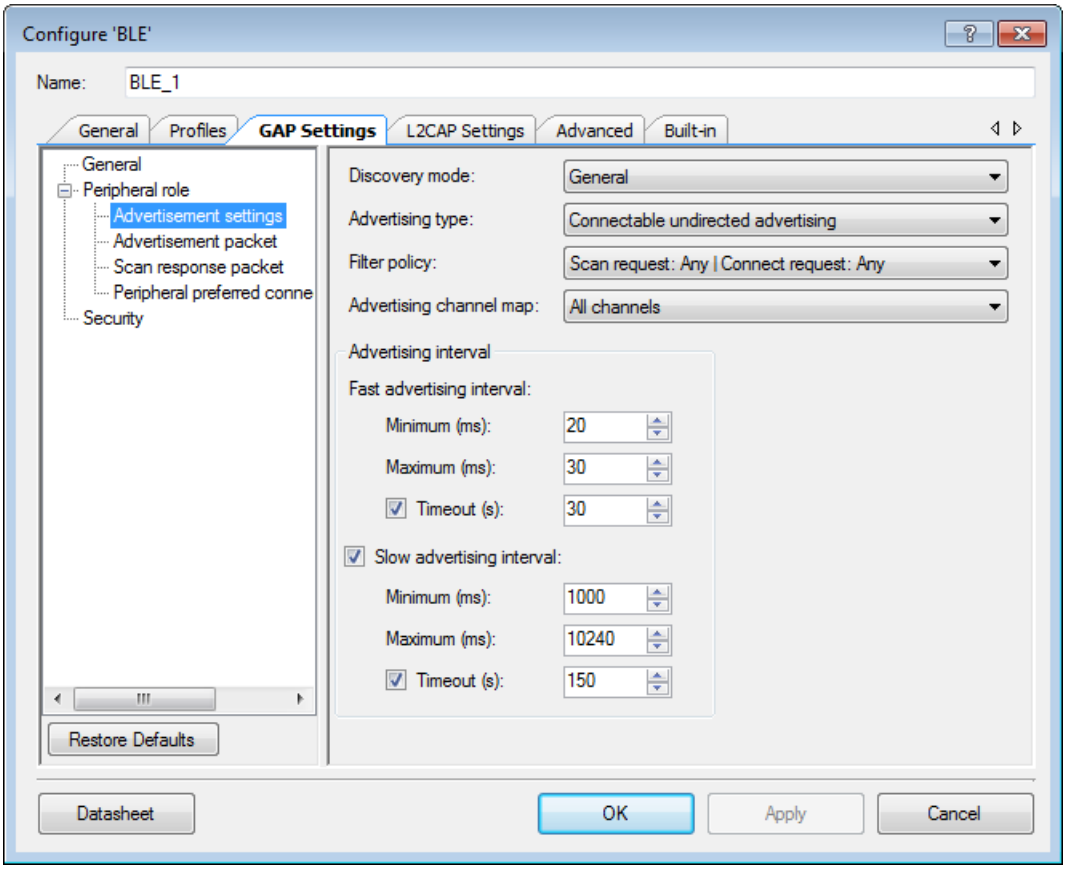
*Enable Link Layer Privacy*

Enables LL Privacy 1.2 feature of Bluetooth 4.2 and enables generation of CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE and CYBLE_EVT_GAPC_DIRECT_ADV_REPORT events.

Note that CYBLE_EVT_GAP_DEVICE_CONNECTED event is not generated when this feature is enabled. This option is available only for devices supporting Bluetooth 4.2.

### GAP Settings Tab – Advertisement Settings

These parameters are available when the device is configured as "Peripheral," "Peripheral and Central," or "Broadcaster" **GAP role**.



*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. The timeout duration is defined by the applicable advertising timeout parameter.

*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

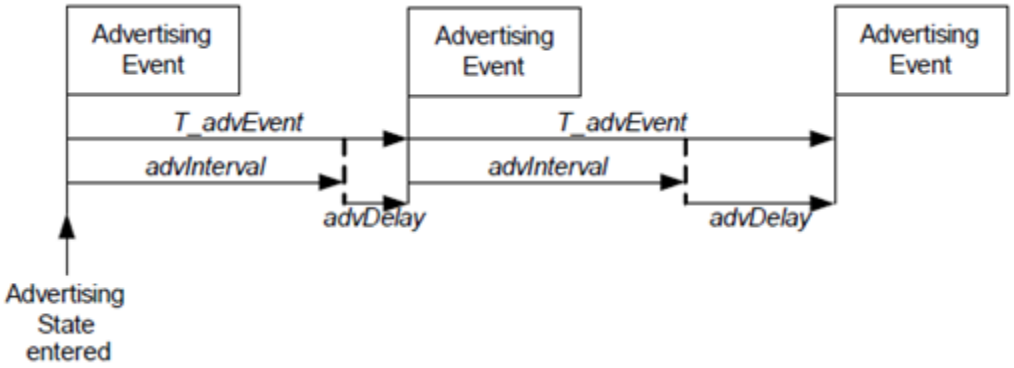
*Advertising Interval*

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, 'CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP' event is generated.

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

- **Fast advertising interval** – This advertisement interval results in faster LE Connection. The BLE 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.

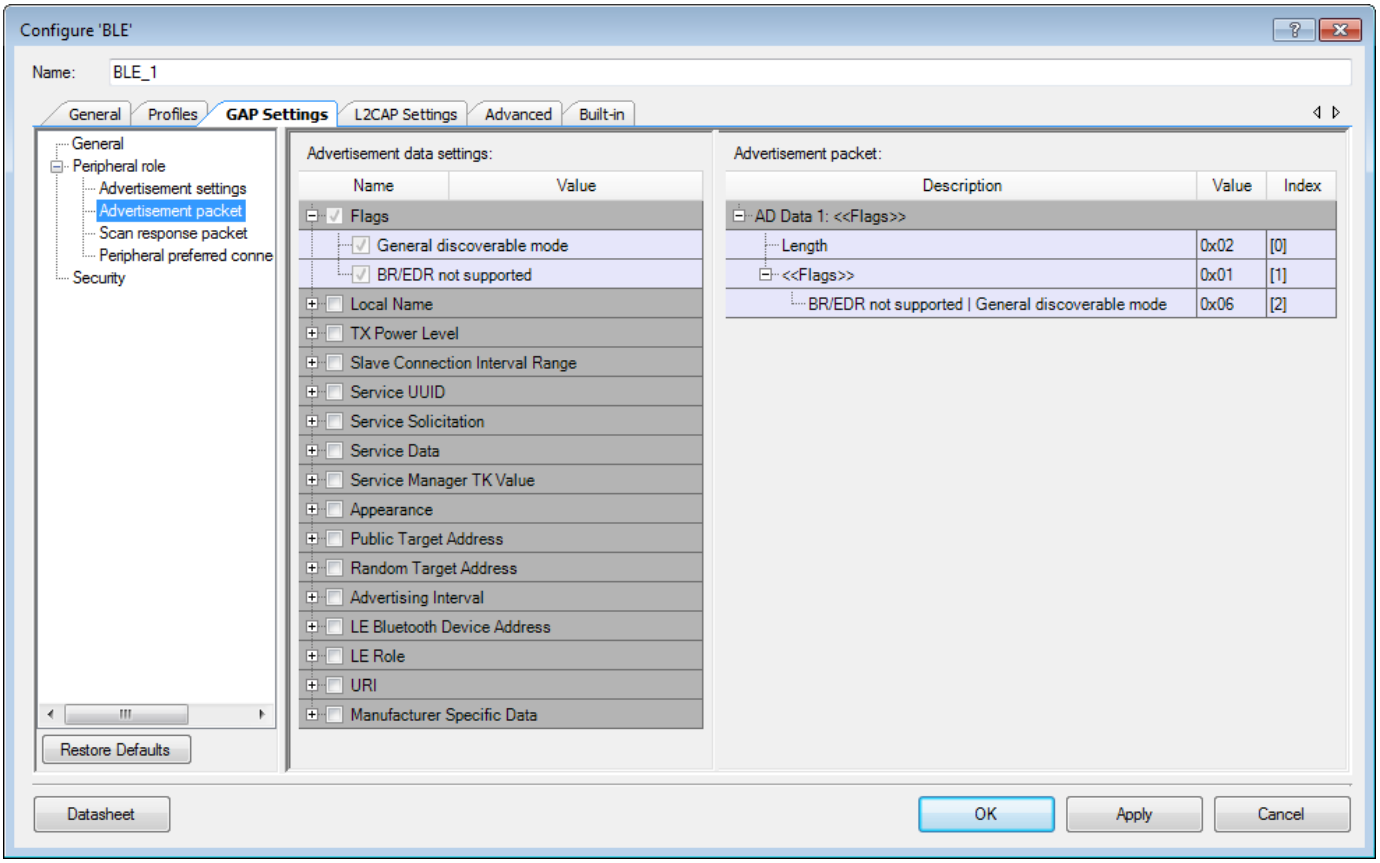
- **Slow advertising interval** – Defines the advertising interval for slow advertising. This is an optional parameter which, if enabled, allows to implement advertising with a lower duty cycle to save battery life. The Slow advertising interval parameters are applied to the device after the internal 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.
  - 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 Pakcage Data Unit (PDU) sent on each of used advertising channels.

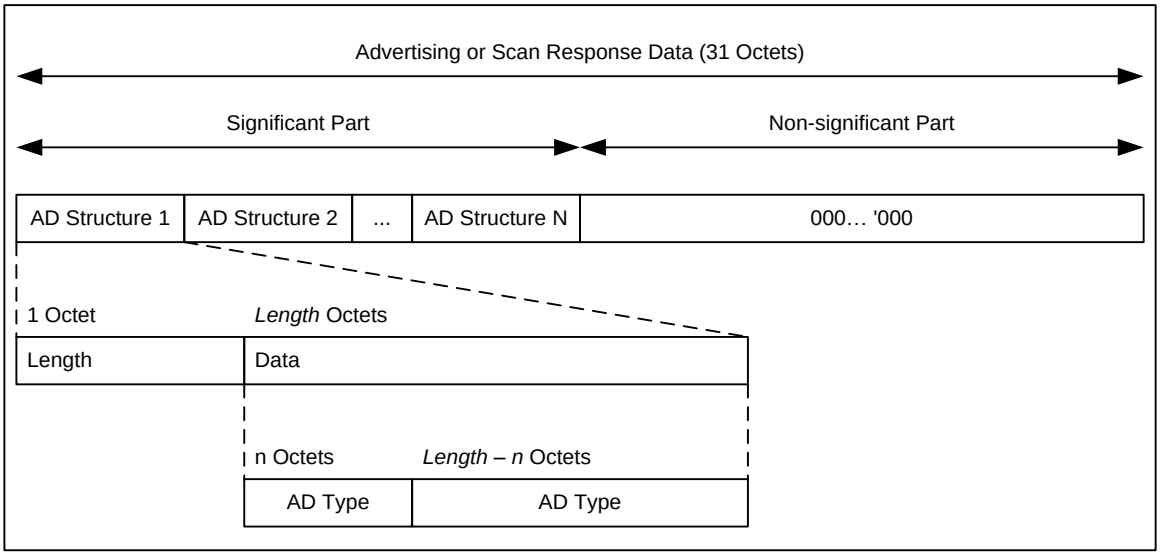
### GAP Settings Tab – Advertisement packet

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



*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)

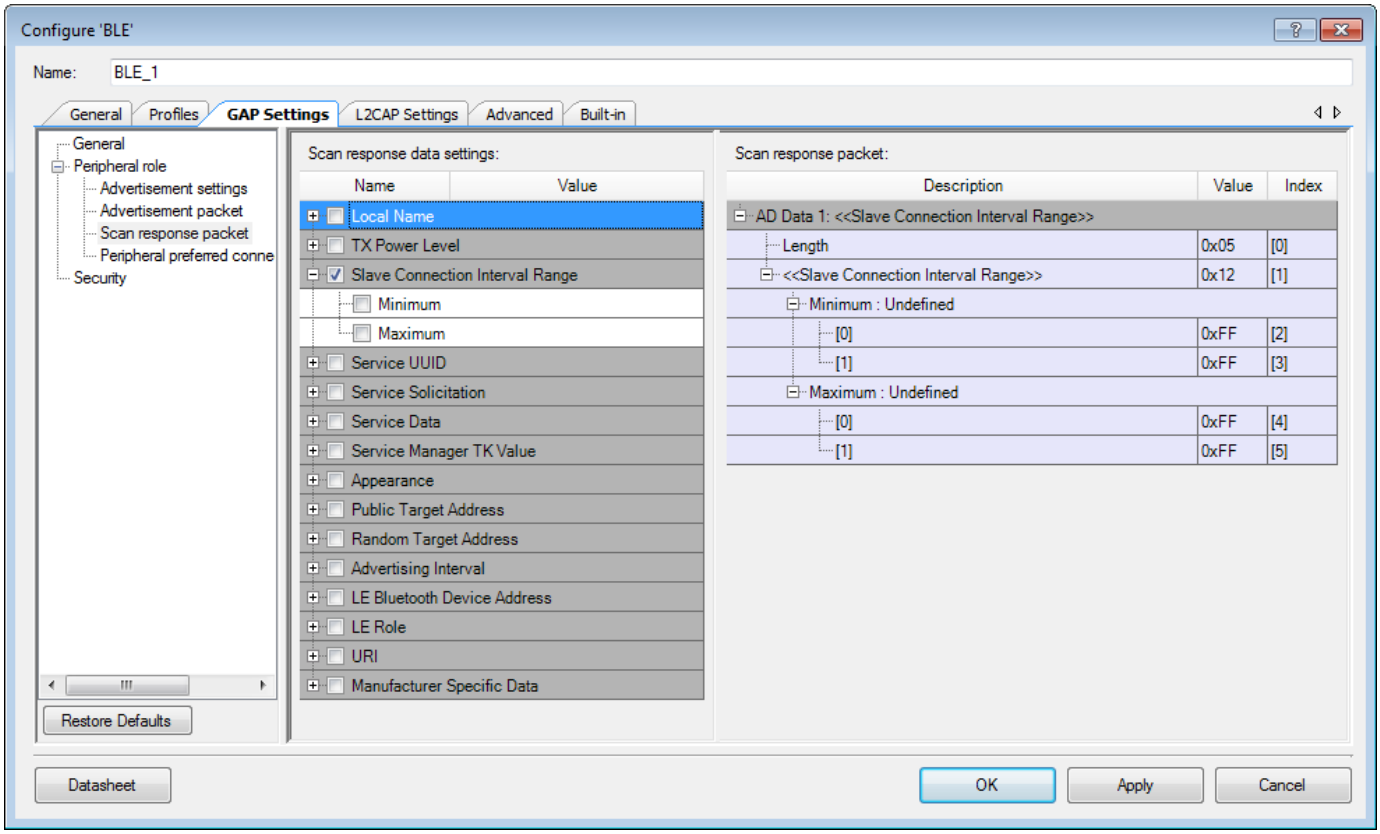
The following table shows the **AD Types**.

| AD Type | Description |
|---|---|
| Flags | Flags to broadcast underlying BLE transport capability such as Discoverable mode, LE only, etc. |

| AD Type | Description |
|---|---|
| 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

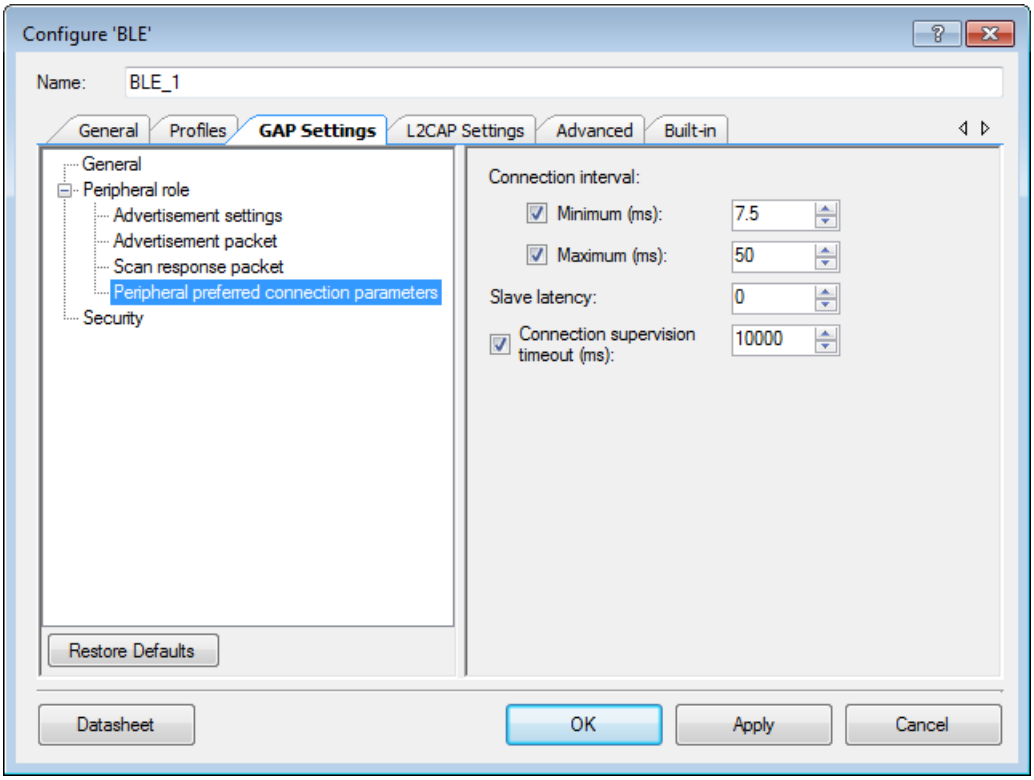
This section displays when the device is configured to contain a "Peripheral," "Broadcaster," or "Peripheral and Central" **GAP role**. It is used to configure the Scan response data packet to be used in response to device scanning performed by a GATT Client device.




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 – Peripheral preferred connection parameters

These parameters define the preferred BLE interface connection settings of the Peripheral. After establishing a connection, the Central device may read these settings and update the BLE interface connection parameters accordingly.




**Note** The scaled values of these parameters used internally by the BLE stack are also shown in the **Peripheral Preferred Connection Parameters** on the **Profiles** tab. These are the actual values sent over the air.

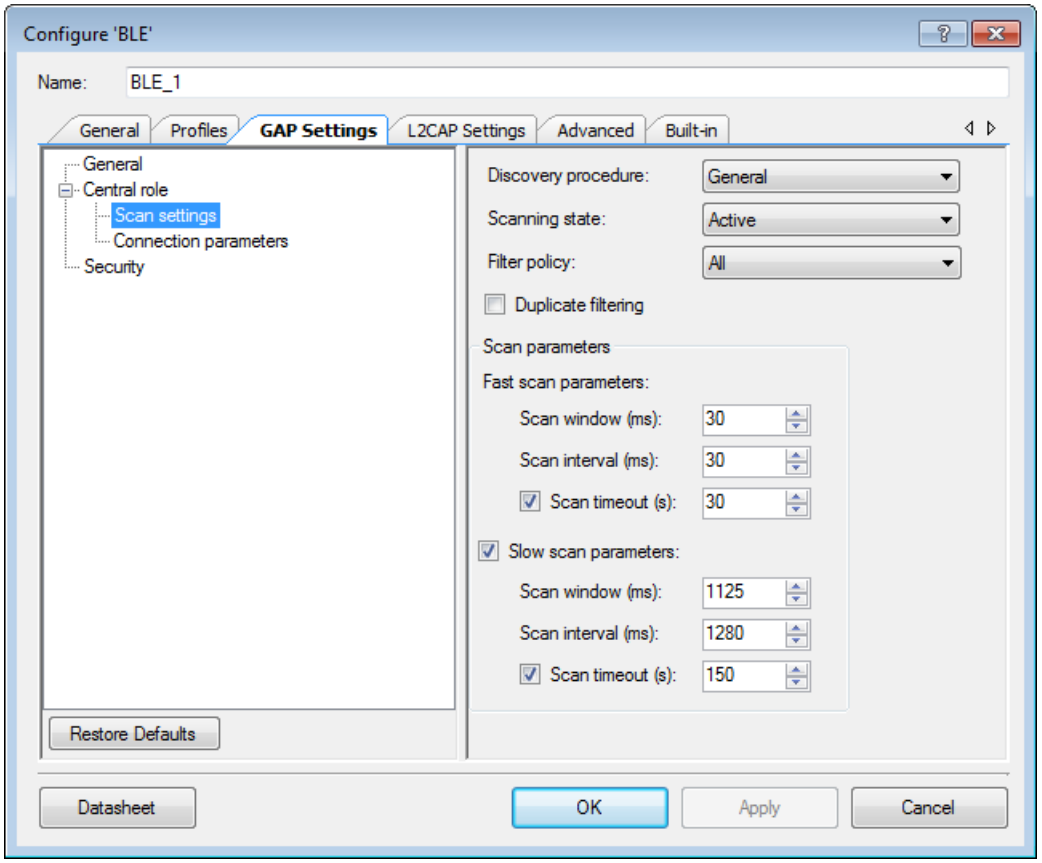
- **Connection interval –** The Central device connecting to a Peripheral device needs to define the time interval for a connection to happen.
  - Minimum (ms): This parameter is the minimum permissible connection time value to be used during a connection event. It is configured in steps of 1.25 ms. 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.
- **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** that for proper operation the Connection Supervision Timeout must be larger than **(1 + Slave latency) * Connection Interval * 2** (ms). Refer to Bluetooth Core Specification 4.2 Volume 6, Part B, Chapter 4.5.2 for more information on Connection Supervision Timeout.

### GAP Settings Tab – Scan settings

These parameters are available when the device is configured as a "Central," "Peripheral and 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**.




**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.

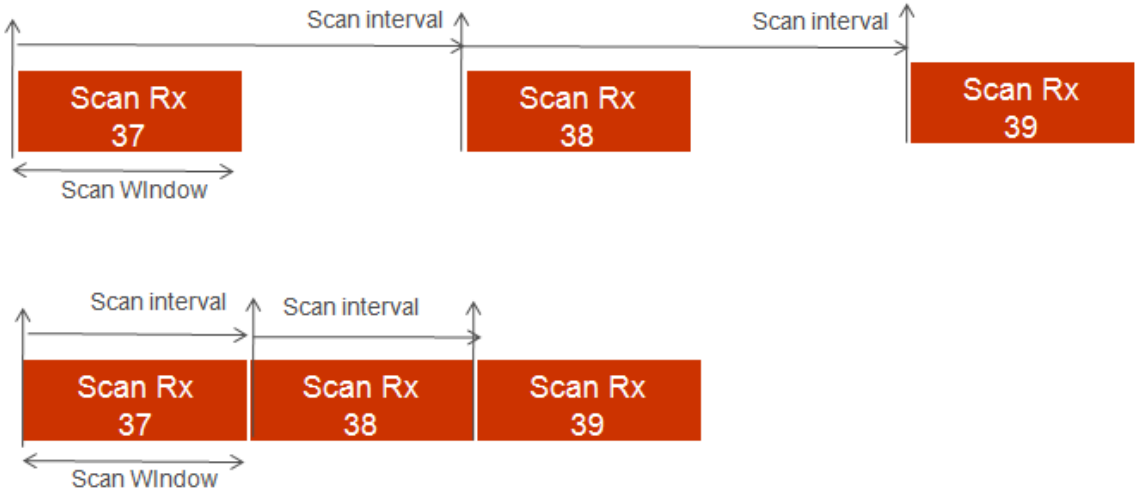
*Scan parameters*

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, 'CYBLE_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.

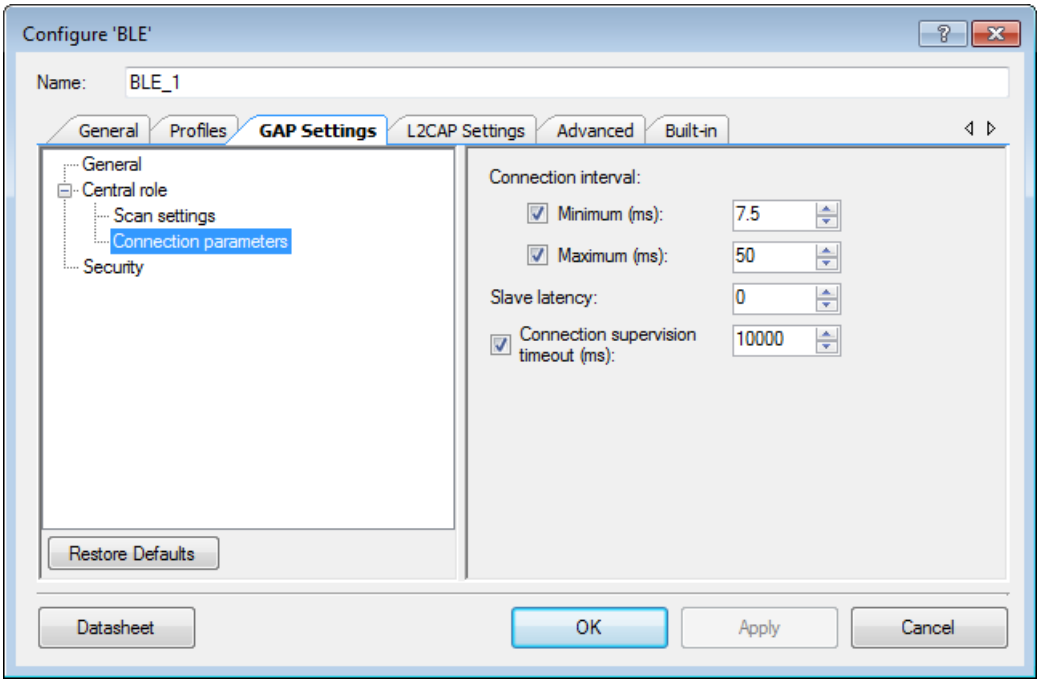
- **Scan Timeout**: The timeout value of scanning with fast scan parameters. Default: 30 s. When unchecked, the device is scanning continuously. The timeout cannot occur before the scanning interval is expired, that is why if a timeout value is less than slow scanning interval minimum value, a warning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



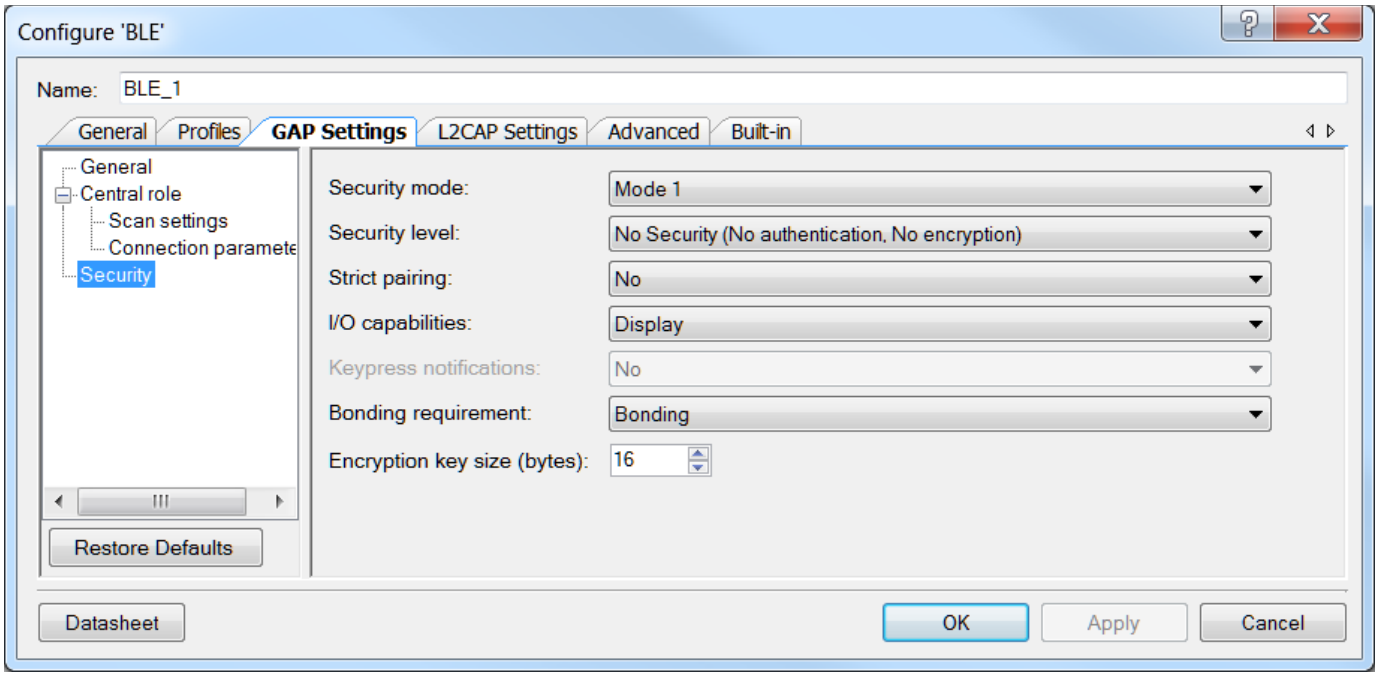
### GAP Settings Tab – Connection parameters

This section is the same as Peripheral Preferred Connection Parameters for Advertisement Settings. The only difference is that Central connection parameters will not be shown on the **Peripheral Preferred Connection parameters** on the **Profile** tab.



### GAP Settings Tab – Security

This section contains several parameters to configure the global security options for the Component. These parameters are configurable only in **Profile** mode. If the device is configured as a GATT Server, you can optionally set each Characteristic using its own unique security setting in the **Profile Tree**.




*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.

*Strict Pairing*

Provides an option to use only the selected security features and doesn't automatically fallback to an unsecure connection if the peer device doesn't support the selected security features.

*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.

- 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.
- Display Only – Used in devices with display capability and may display authentication data. GAP authentication is required.
- Keyboard Only – Used in devices with numeric keypad. 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 and Display – Used in devices like PCs and tablets. GAP authentication is required.

*Keypress notifications*

Provides an option for a keyboard-only 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.

*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 four.

- **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.

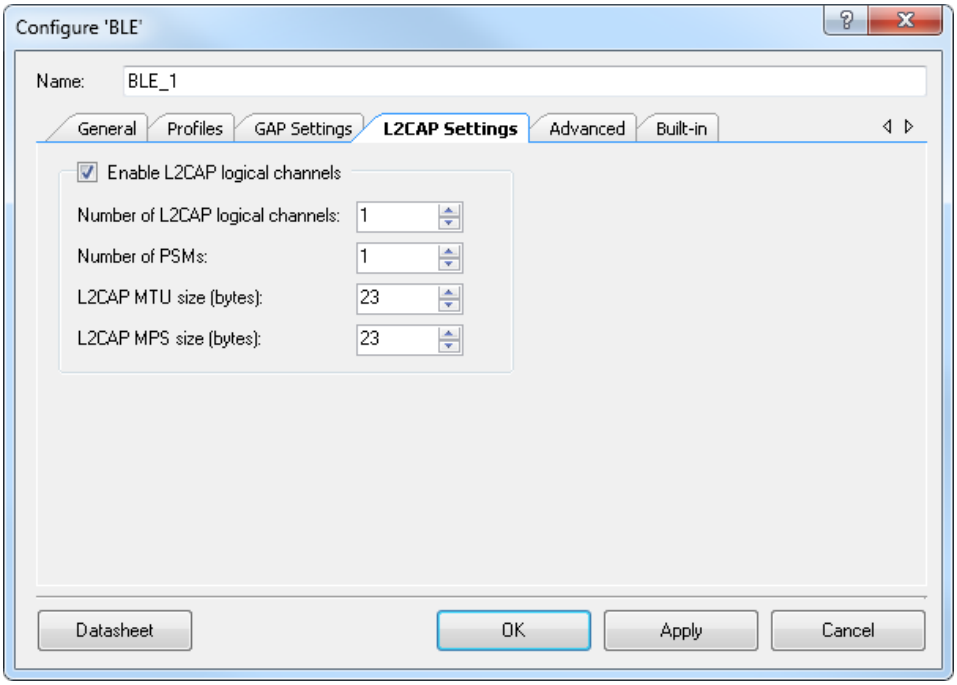
*Other Parameters*

Other parameters that are not exposed in the GUI and have fixed values:

- Maximum Bonded Devices – The maximum number of bonded devices that is supported by this device. Value: 4.
- Maximum Whitelist Size – The maximum number of devices that can be added to the whitelist. Value: 8.
- Maximum Resolvable Devices – The maximum number of peer devices whose addresses should be resolved by this device. Value: 8.

## L2CAP Settings Tab

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




### 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

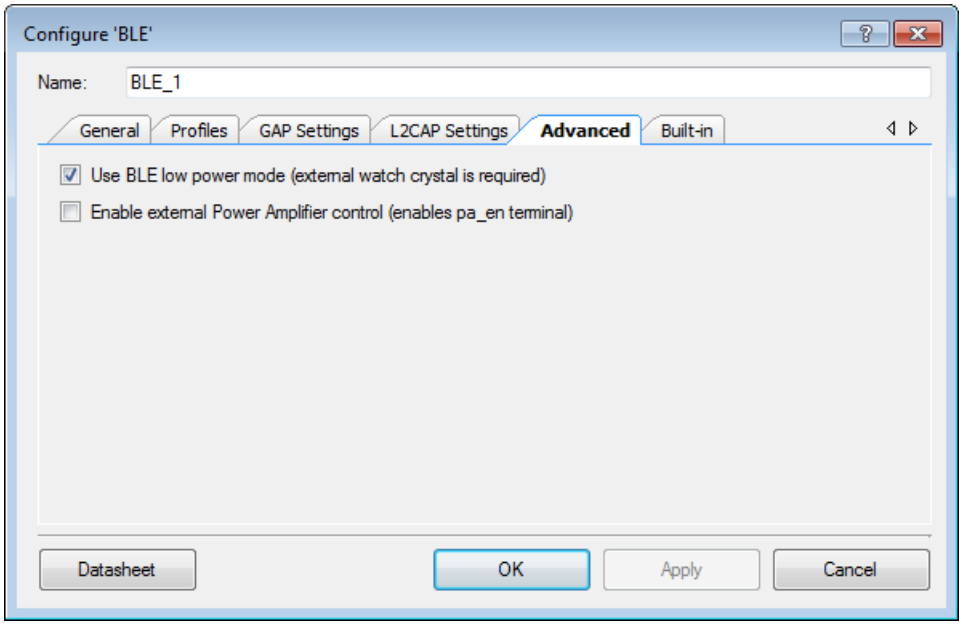
This parameter defines the maximum SDU size of an L2CAP packet. Valid range is from 23 to 65488 bytes. Default: 1280 bytes when **Internet Protocol Support Service** is supported and 23 bytes otherwise.

### L2CAP MPS size

This parameter defines the maximum size of payload data that the L2CAP layer is capable of accepting. **L2CAP MPS size** should be less than or equal to the **L2CAP MTU size** parameter. Valid range is from 23 to 65488 bytes. Default: 23 bytes.

## Advanced Tab

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




### Use BLE low power mode

This parameter identifies if the low power mode support is required for the BLE component. Default: true.

When this parameter is set, WCO 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 control

This parameter enables the high active external power amplifier control signal (pa_en) on a GPIO. This signal is set high just before the BLE RF transmission is enabled and is set low immediately after the BLE RF transmission.

Default: false.

# BLE Component APIs

The BLE Component 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.

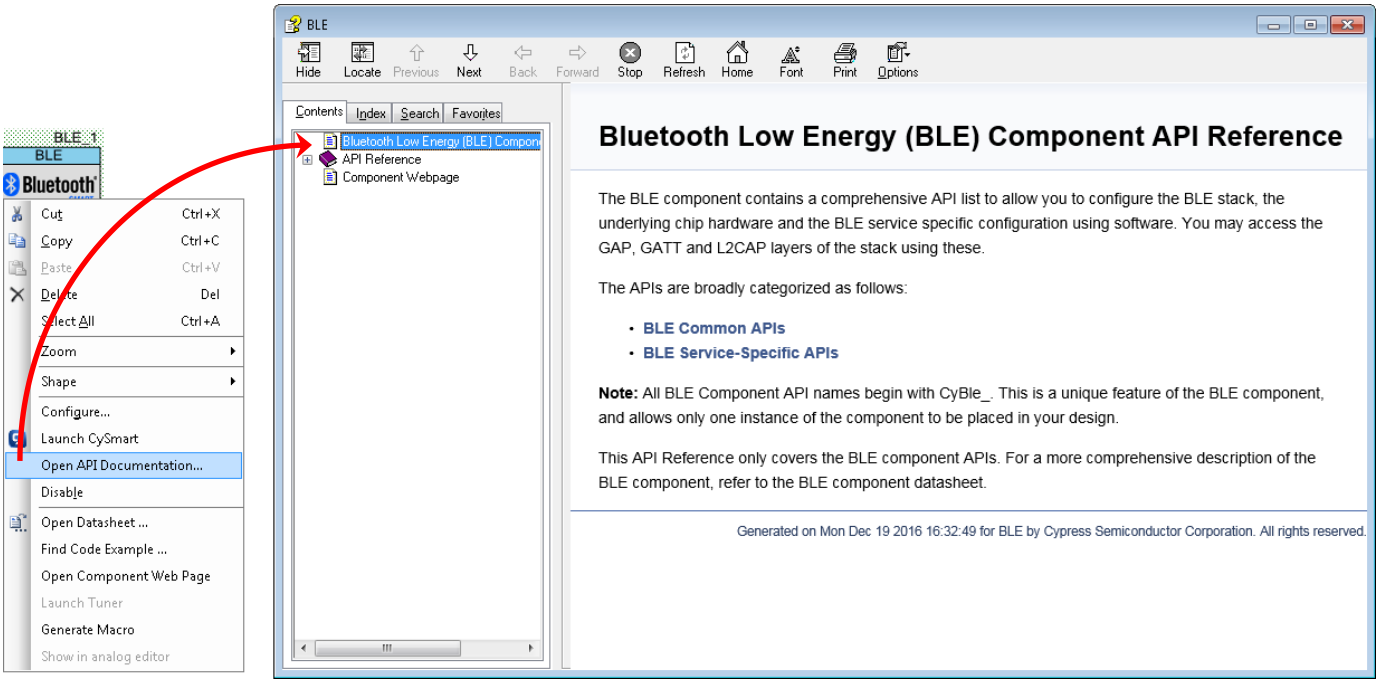
The APIs are broadly categorized as follows:

- BLE Common APIs
- BLE Service-Specific APIs

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

## HTML-Based API Document

Because the BLE Component has numerous APIs, Cypress has also provided a separate HTML-based API reference document (CHM file). To open this file, right-click on the BLE Component on the design canvas, and select **Open API Documentation…**



# BLE Common APIs

## Description

The common APIs act as a general interface between the BLE application and the BLE Stack module. The application may use these APIs to control the underlying hardware such as radio power, data encryption and device bonding via the stack. It may also access the GAP, GATT and L2CAP layers of the stack.

## Modules

- BLE Common Core Functions
  *The common core APIs are used for general BLE component configuration. These include initialization, power management, and utilities.*
- GAP Functions
  *The GAP APIs allow access to the Generic Access Profile (GAP) layer of the BLE stack. Depending on the chosen GAP role in the GUI, you may use a subset of the supported APIs.*
- GATT Functions
  *The GATT APIs allow access to the Generic Attribute Profile (GATT) layer of the BLE stack. Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.*
- L2CAP Functions
  *The L2CAP APIs allow access to the Logical link control and adaptation protocol (L2CAP) layer of the BLE stack.*
- BLE Common Events
  *The BLE stack generates events to notify the application on various status alerts concerning the stack. These can be generic stack events or can be specific to GAP, GATT or L2CAP layers. The service specific events are handled separately in BLE Service-Specific Events.*
- BLE Common Definitions and Data Structures
  *Contains definitions and structures that are common to all BLE common APIs. Note that some of these are also used in Service-specific APIs.*

# BLE Common Core Functions

## Description

The common core APIs are used for general BLE component configuration. These include initialization, power management, and utilities.

## Macros

- #define CyBle_SetState(state) (cyBle_state = (state))
- #define CyBle_GetState() (cyBle_state)
- #define CyBle_GattGetBusyStatus() (cyBle_busyStatus)
- #define CyBle_SetGattError(gattError) (cyBle_gattError = (gattError))

## Functions

- CYBLE_API_RESULT_T CyBle_Start (CYBLE_CALLBACK_T callbackFunc)
- void CyBle_Stop (void)
- CYBLE_API_RESULT_T CyBle_StoreBondingData (uint8 isForceWrite)
- CYBLE_API_RESULT_T CyBle_GapRemoveBondedDevice (CYBLE_GAP_BD_ADDR_T *bdAddr)
- uint8 CyBle_IsDeviceAddressValid (const CYBLE_GAP_BD_ADDR_T *deviceAddress)

- CYBLE_API_RESULT_T CyBle_SoftReset (void)
- CYBLE_LP_MODE_T CyBle_EnterLPM (CYBLE_LP_MODE_T pwrMode)
- CYBLE_LP_MODE_T CyBle_ExitLPM (void)
- void CyBle_ProcessEvents (void)
- CYBLE_API_RESULT_T CyBle_SetDeviceAddress (CYBLE_GAP_BD_ADDR_T *bdAddr)
- CYBLE_API_RESULT_T CyBle_GetDeviceAddress (CYBLE_GAP_BD_ADDR_T *bdAddr)
- int8 CyBle_GetRssi (void)
- CYBLE_API_RESULT_T CyBle_GetTxPowerLevel (CYBLE_BLESS_PWR_IN_DB_T *bleSsPwrLvl)
- CYBLE_API_RESULT_T CyBle_SetTxPowerLevel (CYBLE_BLESS_PWR_IN_DB_T *bleSsPwrLvl)
- CYBLE_API_RESULT_T CyBle_GetBleClockCfgParam (CYBLE_BLESS_CLK_CFG_PARAMS_T *bleSsClockConfig)
- CYBLE_API_RESULT_T CyBle_SetBleClockCfgParam (CYBLE_BLESS_CLK_CFG_PARAMS_T *bleSsClockConfig)
- CYBLE_API_RESULT_T CyBle_GenerateRandomNumber (uint8 *randomNumber)
- CYBLE_API_RESULT_T CyBle_AesEncrypt (uint8 *plainData, uint8 *aesKey, uint8 *encryptedData)
- CYBLE_API_RESULT_T CyBle_SetCeLengthParam (uint8 bdHandle, uint8 mdBit, uint16 ceLength)
- CYBLE_API_RESULT_T CyBle_WriteAuthPayloadTimeout (uint8 bdHandle, uint16 authPayloadTimeout)
- CYBLE_API_RESULT_T CyBle_ReadAuthPayloadTimeout (uint8 bdHandle, uint16 *authPayloadTimeout)
- CYBLE_API_RESULT_T CyBle_GetStackLibraryVersion (CYBLE_STACK_LIB_VERSION_T *stackVersion)
- CYBLE_API_RESULT_T CyBle_IsStackIdle (void)
- CYBLE_BLESS_STATE_T CyBle_GetBleSsState (void)
- void CyBle_AesCcmInit (void)
- CYBLE_API_RESULT_T CyBle_AesCcmEncrypt (uint8 *key, uint8 *nonce, uint8 *in_data, uint8 length, uint8 *out_data, uint8 *out_mic)
- CYBLE_API_RESULT_T CyBle_AesCcmDecrypt (uint8 *key, uint8 *nonce, uint8 *in_data, uint8 length, uint8 *out_data, uint8 *in_mic)
- CYBLE_API_RESULT_T CyBle_GenerateAesCmac (CYBLE_AES_CMAC_GENERATE_PARAM_T *cmacGenParam)
- CYBLE_API_RESULT_T CyBle_SetAppEventMask (uint32 UserEventMask)
- CYBLE_API_RESULT_T CyBle_RegisterBlessInterruptCallback (CYBLE_BLESS_EVENT_PARAM_T *BlessEventParams)
- void CyBle_SetTxGainMode (uint8 bleSsGainMode)
- void CyBle_SetRxGainMode (uint8 bleSsGainMode)
- CYBLE_API_RESULT_T CyBle_SetSlaveLatencyMode (uint8 bdHandle, uint8 setForceQuickTransmit)
- void CyBle_SetSeedForRandomGenerator (uint32 seed)
- CYBLE_API_RESULT_T CyBle_IsLLControlProcPending (void)
- CYBLE_API_RESULT_T CyBle_StartTransmitterTest (CYBLE_TRANSMITTER_TEST_PARAMS_T *TransmitterTestParams)
- CYBLE_API_RESULT_T CyBle_StartReceiverTest (uint8 RxFreq)
- CYBLE_API_RESULT_T CyBle_TestEnd (uint16 *PacketCount)
- CYBLE_API_RESULT_T CyBle_HciSendPacket (CYBLE_HCI_PKT_PARAMS_T *HciPktParams)
- CYBLE_API_RESULT_T CyBle_StoreStackData (uint8 isForceWrite)
- CYBLE_API_RESULT_T CyBle_StoreAppData (uint8 *srcBuff, const uint8 destAddr[], uint32 buffLen, uint8 isForceWrite)

## Macro Definition Documentation

**#define CyBle_SetState( state) (cyBle_state = (state))**

Used to set the component state machine's state.

**Parameters:**

| | |
|---|---|
| *state* | The desired state of type CYBLE_STATE_T that the event handler's state machine should be set to. For detailed information refer to CyBle_GetState() API function description. |

**#define CyBle_GetState() (cyBle_state)**

This function is used to determine the current state of the component state machine.

The component is in the state CYBLE_STATE_INITIALIZING after CyBle_Start() function is called and until CYBLE_EVT_STACK_ON event is not received. After successful initialization the state is changed to CYBLE_STATE_DISCONNECTED. For GAP Peripheral role if CyBle_GappStartAdvertisement() is called and CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event received the state is changed to the CYBLE_STATE_ADVERTISING. For GAP Central role if CyBle_GapcStartScan() API function is called and CYBLE_EVT_GAPC_SCAN_START_STOP event received the state is changed to the CYBLE_STATE_SCANNING. When CyBle_GapcConnectDevice() is called the state is changed to CYBLE_STATE_CONNECTING. After successfully connection indicated by CYBLE_EVT_GAP_DEVICE_CONNECTED or CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE event the state is changed to CYBLE_STATE_CONNECTED. If CyBle_GapDisconnect() API function is called and EVT_GAP_DEVICE_DISCONNECTED event received the state is changed to the CYBLE_STATE_DISCONNECTED. If CyBle_Stop() is called state of component is changed to the CYBLE_STATE_STOPPED.

**Returns:**

CYBLE_STATE_T: The current state.

- CYBLE_STATE_STOPPED - BLE is turned off
- CYBLE_STATE_INITIALIZING, - Initializing state
- CYBLE_STATE_CONNECTED - Peer device is connected
- CYBLE_STATE_ADVERTISING - Advertising process
- CYBLE_STATE_SCANNING - Scanning process
- CYBLE_STATE_CONNECTING - Connecting
- CYBLE_STATE_DISCONNECTED - Essentially idle state

**#define CyBle_GattGetBusyStatus() (cyBle_busyStatus)**

This function returns the status of BLE stack (busy or not busy). The status is changed after CYBLE_EVT_STACK_BUSY_STATUS event.

**Returns:**

uint8: Busy status

- CYBLE_STACK_STATE_BUSY - BLE stack busy
- CYBLE_STACK_STATE_FREE - BLE stack not busy

**#define CyBle_SetGattError( gattError) (cyBle_gattError = (gattError))**

Sets the GATT Error Code after the Authorization Code check on the application layer on the CYBLE_EVT_<service initials>_WRITE_CHAR event for the Bond Management Control Point characteristic.

This API function function is useful only within the registered service callback on the CYBLE_EVT_<service initials>_CHAR event for the certain services:

BMS: Check the Authorization Code of the Bond Management Control Point characteristic. CTS: To set GATT error in case if one or several data fields was/were ignored by the Server. ESS: Used by user to indicate the unsupported condition of ES Trigger Descriptor. CGMS: Check CRC and the length of the characteristics.

CYBLE_GATT_ERR_CODE_T gattError: GATT Error Code, possible values are:

- CYBLE_GATT_ERR_NONE - if the application layer decides the Authorization Code is correct for this OpCode.
- For the BMS:

  - CYBLE_GATT_ERR_OP_CODE_NOT_SUPPORTED - if the application layer decides the OpCode is not supported.
  - CYBLE_GATT_ERR_INSUFFICIENT_AUTHORIZATION - if the application layer decides the Authorization Code is not correct for this OpCode.
- For the CTS: CYBLE_GATT_ERR_CTS_DATA_FIELD_IGNORED - one or several data fields was/were ignored.
- For the ESS:
  - CYBLE_GATT_ERR_CONDITION_NOT_SUPPORTED - to indicate that the requested condition is not supported.
- For the CGMS:
  - CYBLE_GATT_ERR_MISSING_CRC - when the CRC is missed.
  - CYBLE_GATT_ERR_INVALID_CRC - when the CRC is incorrect.
  - CYBLE_GATT_ERR_INVALID_PDU - when the length of the attribute is incorrect.

## Function Documentation

### CYBLE_API_RESULT_T CyBle_Start (CYBLE_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 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 the generation of CYBLE_EVT_STACK_ON event on successful initialization of the BLE Stack.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | Event callback function to receive events from BLE stack. CYBLE_CALLBACK_T is a function pointer type. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On passing a NULL pointer to the function. |
| CYBLE_ERROR_REPEATED_ATTEMPTS | On invoking this function more than once without calling CyBle_Shutdown() function between calls to this function. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | There is insufficient memory available. |

**Global Variables**

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

### void CyBle_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 CyBle_Start().

**Returns:**

None

### CYBLE_API_RESULT_T CyBle_StoreBondingData (uint8 *isForceWrite*)

This function writes the new bonding data from RAM to the dedicated Flash location as defined by the component. It performs data comparing between RAM and Flash before writing to Flash. If there is no change between RAM and Flash data, then no write is performed. It writes only one flash row in one call. Application should keep calling this function till it return CYBLE_ERROR_OK. This function is available only when Bonding requirement is selected in Security settings.

**Parameters:**

| | |
|---|---|
| *isForceWrite* | If value is set to 0, then stack will check if flash write is permissible. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_FLASH_WRITE_NOT_PERMITED | Flash Write is not complete |
| CYBLE_ERROR_INVALID_PARAMETER | Invalid input parameter |
| CYBLE_ERROR_FLASH_WRITE | Error in flash Write |

**Side Effects**

For BLE devices with 128K of Flash memory this function will automatically modify the clock settings for the device. Writing to flash requires changes to be done to the IMO (set to 48 MHz) and HFCLK (source set to IMO) settings. The configuration is restored before returning. This will impact the operation of most of the hardware in the device.

**Global Variables**

The cyBle_pendingFlashWrite variable is used to detect status of pending write to flash operation for stack data and CCCD. This function automatically clears pending bits after write operation complete.

### CYBLE_API_RESULT_T CyBle_GapRemoveBondedDevice (CYBLE_GAP_BD_ADDR_T **bdAddr*)

This function marks the device untrusted. It removes the bonding information of the device including CCCD values. This function removes device from the white list also when autopopulate white list with bonded devices option is enabled.

This function is available only when Bonding requirement is selected in Security settings.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Pointer to peer device address, of type CYBLE_GAP_BD_ADDR_T. If device address is set to 0, then all devices shall be removed from trusted list and white list. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'bdAddr'. |
| CYBLE_ERROR_INVALID_OPERATION | Whitelist is already in use or there is pending write to flash operation. |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device does not exist in the bond list. |

**Global Variables**

The bdHandle is set in cyBle_pendingFlashWrite variable to indicate that data should be stored to flash by CyBle_StoreBondingData() afterwards.

**uint8 CyBle_IsDeviceAddressValid (const CYBLE_GAP_BD_ADDR_T **deviceAddress*)**

This function verifies that BLE public address has been programmed to SFLASH during manufacture. It could be used to verify if public device address is programmed to flash memory.

**Parameters:**

| | |
|---|---|
| *deviceAddress* | the pointer to the BD address of type CYBLE_GAP_BD_ADDR_T. |

**Returns:**

Non zero value when a device address differs from the default SFLASH content.

**CYBLE_API_RESULT_T CyBle_SoftReset (void )**

This function resets the BLE Stack, including BLE sub-system hardware registers. BLE Stack transitions to idle mode. This function can be used to reset the BLE Stack if the BLE Stack turns unresponsive due to incomplete transfers with the peer BLE device.

A call to this function results in the generation of CYBLE_EVT_STACK_ON event on successful BLE Stack Reset.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_OPERATION | This error occurs if this function is invoked before invoking CyBle_StackInit function. |

**CYBLE_LP_MODE_T CyBle_EnterLPM (CYBLE_LP_MODE_T *pwrMode*)**

This function requests the underlying BLE modules such as BLE Controller, BLE Host Stack and BLE Stack manger to enter into one of the supported low power modes. Application should use this function to put Bluetooth Low Energy Sub-System (BLESS) to Low Power Mode (LPM).

BLE Stack enters and exits low power modes based on its current state and hence the application should consider the BLE Stack LPM state before putting the CPU or the overall device into LPM. This function attempts to set the requested low power mode and if that is not possible, it tries to set the next higher low-power-mode. This behavior is due to the requirement that the application will always try to use the lowest power mode when there is nothing that it needs to process. Note that the CPU will not be able to access the BLESS registers when BLESS is in deep sleep mode.

BLE Stack has the following power modes:

1. Active
2. Sleep (Low Power Mode)
3. DeepSleep with ECO Off (Low Power Mode)
4. Hibernate (Low Power Mode)

Note that certain conditions may prevent BLE sub system from entering a particular low power mode.

**Active Mode**

Bluetooth Low Energy Sub System (BLESS) has three sub-modes in Active mode:

1. Idle
2. Transmit Mode, and
3. Receive Mode

These modes draw full current from the device and the CPU has full access to its registers.

**Sleep Mode**

The clock to the link layer engine and digital modem is gated and the (External Crystal Oscillator) ECO continues to run to maintain the link layer timing. The application cannot enter sleep mode if a Transmit or Receive is in progress.

**Deep Sleep with ECO Off Mode**

The ECO is stopped and Watch Crystal Oscillator (WCO) is used to maintain link layer timing. All the regulators in the Radio Frequency (RF) transceiver are turned off to reduce leakage current and BLESS logic is kept powered ON from the System Resources Sub System (SRSS) deep-sleep regulator for retention of current BLESS state information. This mode can be entered from either Idle (Active) or Sleep mode. It should be entered when the next scheduled activity instant in time domain is greater than the Deep Sleep total wakeup time (typically 2ms).

NOTE: If application is using ECO as source of HFCLK for higher clock accuracy and calls this API function to move BLESS to Deep Sleep mode then HFCLK accuracy and frequency would be impacted as this API function switches HFCLK source from ECO to IMO. On BLESS wakeup, the HFCLK source would be switched back to ECO.

Recommendation is that application turns on IMO and sets it as HFCLK source before calling this API function. Upon wakeup due to sources other than BLESS, application can turn on ECO and switch HFCLK source to ECO. Pseudo code of recommendation is given below.

Pseudo Code: //Turn on IMO and switch HFCLK to IMO CyBle_EnterLPM(CYBLE_BLESS_DEEPSLEEP); CySysPmDeepSleep(); //If exit is not due to BLE and application need to use ECO //then turn on ECO and switch HFCLK source to ECO.

**Hibernate mode**

The application layer should invoke this function with the Hibernate Mode option to put the BLE Stack in to hibernate mode. If this mode is set, the micro-controller can be put in to Hibernate Mode by the application layer. This mode ensures that BLE Sub-system is completely idle and no procedures such ADV, SCAN and CONNECTION are active.

The following table indicates the allowed sleep modes for the complete system (BLE Sub-system and the micro-controller). Modes marked In 'X' are the allowed combinations. The application layer should make sure that the invalid modes are not entered in to:

| BLE Stack LPM / PSoC4 A-BLE LPM | Active | Sleep | DeepSleep | Hibernate |
|---|---|---|---|---|
| Active | X | | | |
| Sleep | X | X | | |
| DeepSleep (ECO OFF) | X | X | X | |
| Hibernate | | | | X |

The application layer is responsible for putting the BLE Sub-system and the micro-controller in to the desired sleep modes. Upon entering the requested sleep mode combination, the BLE Sub-system and the micro-controller are woken up by an interrupt every advertisement interval(in case of a GAP Peripheral) or connection interval (in case of GAP Central). On wakeup, if the application needs to transmit some data, appropriate function(s) including the

Stack functions need to be invoked. This needs to be followed by a call to the function CyBle_ProcessEvents, which handles all pending transmit and receive operations. The application can now put the complete system back in to one of the sleep modes. The application should ensure that the above invalid states are never encountered.

Application shall also ensure that BLE Sub-system's low power entry and low power exit interrupts are processed in realtime and not blocked. It is recommended that BLE Sub-system interrupt should be of higher priority. If BLE Sub-system interrupts are blocked for longer time ( > 200us ), BLE Sub-system can violate Bluetooth specification timing for wakeup where ECO is required to perform BLE radio operation. It can also result in race condition where BLE Stack waits for interrupt as ECO is not started correctly and BLE Sub system enters in unknown state, BLE Stack gets stuck in busy loop.

This is a blocking function. In process of entering in BLESS Deep Sleep Mode, BLE Stack puts CPU in Sleep Mode to save power while polling for entry indication to BLESS DSM. No event is generated on calling this function. Based on the return code from this function, the application layer should decide on the sleep mode for the complete system. For example, if the return code is CYBLE_BLESS_DEEPSLEEP, the application can choose to call system wide DeepSleep mode function.

**Parameters:**

| | |
|---|---|
| *pwrMode* | The power mode that the component is intended to enter. The allowed values are,<br>• CYBLE_BLESS_SLEEP<br>• CYBLE_BLESS_DEEPSLEEP |

**Returns:**

CYBLE_LP_MODE_T: The actual power mode that BLE stack is now set to.

### CYBLE_LP_MODE_T CyBle_ExitLPM (void )

Application can asynchronously wake up the BLE Stack from low power using this function. The wake up is not performed for the entire chip. This is a blocking call and returns when BLE Stack has come out of LPM, and in process of waking up from BLESS Deep Sleep Mode, BLE Stack puts CPU in Sleep Mode to save power while polling for wakeup indication from BLESS. No event is generated on calling this function. It has no effect if it is invoked when the BLE Stack is already in active mode.

**Returns:**

CYBLE_LP_MODE_T: The actual power mode that BLE stack is now set to. Expected return value is CYBLE_BLESS_ACTIVE.

### void CyBle_ProcessEvents (void )

This function checks the internal task queue in the BLE Stack, and pending operation of the BLE Stack, if any. This needs to be called at least once every interval 't' where:

1. 't' is equal to connection interval or scan interval, whichever is smaller, if the device is in GAP Central mode of operation, or
2. 't' is equal to connection interval or advertisement interval, whichever is smaller, if the device is in GAP Peripheral mode of operation.

On calling every interval 't', all pending operations of the BLE Stack are processed. This is a blocking function and returns only after processing all pending events of the BLE Stack Care should be taken to prevent this call from any kind of starvation; on starvation, events may be dropped by the stack. All the events generated will be propagated to higher layers of the BLE Stack and to the Application layer only after making a call to this function.

Call to this function can wakeup BLESS from Low Power Mode, and in process of waking up from BLESS Deep Sleep Mode, BLE Stack puts CPU in Sleep Mode to save power while polling for wakeup indication from BLESS. This can occur if the caller function has pending data or control transactions to be performed in BLE Stack that need to be programmed to BLESS in CyBle_ProcessEvents() context and BLESS is in Low Power Mode.

**Returns:**

None

### CYBLE_API_RESULT_T CyBle_SetDeviceAddress (CYBLE_GAP_BD_ADDR_T *bdAddr)

This function sets the Bluetooth device address into BLE Stack's memory. This address shall be used for all BLE procedures unless explicitly changed by application. The application layer needs to call this function every time an address change is required. Bluetooth 4.1 Core specification [3.12] specifies that the Bluetooth device can change its private address periodically, with the period being decided by the application; there are no limits specified on this period. The application layer should maintain its own timers in order to do this.

User should call 'CyBle_GapSetIdAddress' API function to set identity address if 'CyBle_SetDeviceAddress' API function is used to set public or random static address. This is a blocking function. No event is generated on calling this function. This API function will be obsolete in future.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Bluetooth Device address retrieved from the BLE stack gets stored to a variable pointed to by this pointer. The variable is of type CYBLE_GAP_BD_ADDR_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |
| CYBLE_ERROR_INVALID_OPERATION | Operation is not permitted when device is in connected state. |

### CYBLE_API_RESULT_T CyBle_GetDeviceAddress (CYBLE_GAP_BD_ADDR_T *bdAddr)

This API function reads the BD device address from BLE Controller's memory. This address shall be used for BLE procedures unless explicitly indicated by BLE Host through HCI commands. This is a blocking function and it returns immediately with the required value.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Pointer to the CYBLE_GAP_BD_ADDR_T structure variable. It has two fields where,<br>• bdAddr.addr: Bluetooth Device address buffer that is populated with the device address data from BLE stack.<br>• bdAddr.type: Caller function should fill the "address type" to retrieve appropriate address.<br>Caller function should use bdAddr.type = 0x00 to get the "Public Device Address" which is currently set.<br>Caller function use bdAddr.type = 0x01 to get the "Random Device Address" which is currently set. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**int8 CyBle_GetRssi (void )**

This function reads the recorded Received Signal Strength Indicator (RSSI) value for the last successfully received packet from the BLE radio sub-system. This is a blocking function. No event is generated on calling this function.

**Returns:**

int8: The RSSI value of the responding device.

| Information | Description |
|---|---|
| Range | -85 <= N <= 5 |
| Note | The value is in dBm. |

**CYBLE_API_RESULT_T CyBle_GetTxPowerLevel (CYBLE_BLESS_PWR_IN_DB_T *bleSsPwrLvl)**

This function reads the transmit power of the BLE radio for the given BLE sub-system channel group. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bleSsPwrLvl* | Pointer to a variable of type CYBLE_BLESS_PWR_IN_DB_T where,<br>• bleSsPwrLvl -> blePwrLevelInDbm indicates Output Power level in dBm returned by the function.<br>• bleSsPwrLvl -> bleSsChId indicates Channel group for which power level is to be read. This needs to be set before calling the function. The value can be advertisement channels (CYBLE_LL_ADV_CH_TYPE) or data channels (CYBLE_LL_CONN_CH_TYPE).<br>• If bleSsPwrLvl->blePwrLevelInDbm is greater than 0dBm, then the power level is applicable to both advertisement and connection channel. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter |

**CYBLE_API_RESULT_T CyBle_SetTxPowerLevel (CYBLE_BLESS_PWR_IN_DB_T *bleSsPwrLvl)**

This function sets the transmit power of the BLE radio for given BLE sub-system channel group. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bleSsPwrLvl* | Pointer to a variable of type 'CYBLE_BLESS_PWR_IN_DB_T' where,<br>• bleSsPwrLvl -> blePwrLevelInDbm indicates Output Power level in dBm to be set by the function.<br>• bleSsPwrLvl -> bleSsChId indicates Channel group for which power level is to be set. The value can be advertisement channels (CYBLE_LL_ADV_CH_TYPE) or data channels (CYBLE_LL_CONN_CH_TYPE). |

NOTE: The set power level is applicable to both advertisement and connection channel for the following scenarios

- bleSsPwrLvl->blePwrLevelInDbm is greater than 0dB

- Before calling this API function Tx power level is 3dB

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_GetBleClockCfgParam (CYBLE_BLESS_CLK_CFG_PARAMS_T *bleSsClockConfig)**

This function reads the clock configuration parameter of BLE sub-system. This is a blocking function. No event is generated on calling this function. The following parameters related to the BLE sub-system clock are set by this function:

**Sleep Clock accuracy**

Sleep clock accuracy (SCA) in PPM. This parameter indicates the sleep clock accuracy in PPM as described in the following table. It is set in the BLE Stack and is used for BLE Connection operation while creating LE connection with the peer device.

| Sleep Clock Accuracy Enum Field | PPM Range Translation (PPM) |
|---|---|
| CYBLE_LL_SCA_251_TO_500_PPM | 251 - 500 |
| CYBLE_LL_SCA_151_TO_250_PPM | 151 - 250 |
| CYBLE_LL_SCA_101_TO_150_PPM | 101 - 150 |
| CYBLE_LL_SCA_076_TO_100_PPM | 76 - 100 |
| CYBLE_LL_SCA_051_TO_075_PPM | 51 - 75 |
| CYBLE_LL_SCA_031_TO_050_PPM | 31 - 50 |
| CYBLE_LL_SCA_021_TO_030_PPM | 21 - 30 |
| CYBLE_LL_SCA_000_TO_020_PPM | 0 - 20 |

Refer to Bluetooth Core Specification 4.1 Volume 6, Chapter 4.5.7 for more details on how the SCA is used.

**Link Layer clock divider**

This input decides the frequency of the clock to the link layer. A lower clock frequency results in lower power consumption. Default clock frequency for the operation is 24 MHz. BLESS supports 24 MHz, 12 MHz and 8 MHz clock configurations. Based on the end application requirement (how frequent the communication is expected to be), this parameter needs to be set.

**ecoXtalStartUpTime** ECO startup time specifies the value in the unit of 62.5 us (16 KHz clock cycles). This value is programmed in BLESS WAKE_UP config register, to configure the wakeup time required by ECO. Max value for ECO startup time field can be 79u units = (79 * 62.5) us

**Parameters:**

| | |
|---|---|
| *bleSsClockConfig* | Pointer to a variable of type CYBLE_BLESS_CLK_CFG_PARAMS_T to which the existing clock configuration is stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_SetBleClockCfgParam (CYBLE_BLESS_CLK_CFG_PARAMS_T **bleSsClockConfig*)**

This function sets the clock configuration parameter of BLE sub-system. This is a blocking function. No event is generated on calling this function. The following parameters related to the BLE sub-system clock are set by this function:

**Sleep Clock accuracy**

Sleep clock accuracy (SCA) in PPM. This parameter indicates the sleep clock accuracy in PPM as described in the following table. It is set in the BLE Stack and is used for BLE Connection operation while creating LE connection with the peer device.

| Sleep Clock Accuracy Enum Field | PPM Range Translation (PPM) |
|---|---|
| CYBLE_LL_SCA_251_TO_500_PPM | 251 - 500 |
| CYBLE_LL_SCA_151_TO_250_PPM | 151 - 250 |
| CYBLE_LL_SCA_101_TO_150_PPM | 101 - 150 |
| CYBLE_LL_SCA_076_TO_100_PPM | 76 - 100 |
| CYBLE_LL_SCA_051_TO_075_PPM | 51 - 75 |
| CYBLE_LL_SCA_031_TO_050_PPM | 31 - 50 |
| CYBLE_LL_SCA_021_TO_030_PPM | 21 - 30 |
| CYBLE_LL_SCA_000_TO_020_PPM | 0 - 20 |

Refer to Bluetooth Core Specification 4.1 Volume 6, Chapter 4.5.7 for more details on how the SCA is used.

**Link Layer clock divider**

This input decides the frequency of the clock to the link layer. A lower clock frequency results in lower power consumption. Default clock frequency for the operation is 24MHz. BLESS supports 24MHz, 12MHz and 8MHz clock configurations. Based on the end application requirement (how frequent the communication is expected to be), this parameter needs to be set.

**ecoXtalStartUpTime** ECO startup time specifies the value in the unit of 62.5us (16KHz clock cycles). This value is programmed in BLESS WAKE_UP config register, to configure the wakeup time required by ECO. Max value for ECO startup time field can be 79u units = (79 * 62.5) us

**Parameters:**

| | |
|---|---|
| *bleSsClockConfig* | Pointer to a variable of type CYBLE_BLESS_CLK_CFG_PARAMS_T from which the existing clock configuration is taken. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_GenerateRandomNumber (uint8 **randomNumber*)**

This function generates 8-byte random number which complies with pseudo random number generation in accordance with [FIPS PUB 140-2]. Random number generation function is used during security procedure documented in Bluetooth 4.1 core specification, Volume 3, Part H.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *randomNumber* | Pointer to a buffer of size 8 bytes in which the generated random number gets stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_AesEncrypt (uint8 **plainData*, uint8 **aesKey*, uint8 **encryptedData*)**

This function uses BLE sub-system AES engine to encrypt 128-bit of plain text using the given AES key. The output of AES processing is copied to encryptedData buffer. Refer Bluetooth 4.1 core specification, Volume 3, Part H, section 2.2 for more details on usage of AES key.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *plainData* | Pointer to the data containing plain text (128-bit) that is to be encrypted. |
| *aesKey* | Pointer to the AES Key (128-bit) that is to be used for AES encryption. |
| *encryptedData* | Pointer to the encrypted data (128-bit) that is output of AES module for given plainData and aesKey. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter |

**CYBLE_API_RESULT_T CyBle_SetCeLengthParam (uint8 *bdHandle*, uint8 *mdBit*, uint16 *ceLength*)**

This function sets the connection event duration related parameters that can result in extension or truncation of LE connection event based on more data (mdBit) bit status and 'ceLength' duration. Refer Bluetooth 4.1 core specification, Volume 6, Part B, section 4.5 for more details on connection states of BLE Link Layer.

This is a blocking function. No event is generated on calling this function.

BLE Stack uses the BLESS hardware (AES module) to encrypt/decrypt the data. BLESS must be initialized before using this function. This function can safely be used by the application in "single thread/task system" which is the case with the current implementation of the BLE Stack. For multitasking systems, this function must be used within the BLE task to ensure atomic operation.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device bdHandle. |
| *mdBit* | 'More Data' bit to select more number of data packets in BLE Stack buffer. A value of 0x01 indicates extension and a value of 0x00 indicates truncation. |
| *ceLength* | CE length of connection event that can extend the connection event. Details on this parameter are as given below:<br>• Value Range = 0x0000 to 0xFFFF<br>• Time Calculation = N x 0.625 ms<br>• Time Range = 0 ms to 40.959 ms |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | One of the input parameters is invalid |
| CYBLE_ERROR_NO_CONNECTION | When controller can't find active connection using given bdHandle |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Invalid bdHandle or LE connection doesn't exist for link identified by bdHandle. |

**CYBLE_API_RESULT_T CyBle_WriteAuthPayloadTimeout (uint8 *bdHandle*, uint16 *authPayloadTimeout*)**

This function sets the Authentication Payload timeout in BLE Controller for LE_PING feature. Refer Bluetooth 4.1 core specification, Volume 6, Part B, section 4.6.5 for LE Ping operation.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *authPayloadTimeout* | Variable containing authentication timeout value to be written to BLE Controller. Details on this parameter are as given below:<br>• Value Range = 0x0001 to 0xFFFF<br>• Default Value (N) = 3000 (30 seconds)<br>• Time Calculation = N x 10 ms<br>• Time Range = 10 ms to 655,350 ms |

Note: The time at which PING packet transmitted over the air is determined from the following formula (only in case of SlaveLatency is enabled) (authPayloadTimeout - (4 * ((1 + SlaveLatency) * Connection Interval)))

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | One of the input parameters is invalid |
| CYBLE_ERROR_INVALID_OPERATION | Operation is not permitted |
| CYBLE_ERROR_NO_CONNECTION | When controller can't find active connection using given bdHandle |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Invalid bdHandle or LE connection doesn't exist for link identified by bdHandle. |

**CYBLE_API_RESULT_T CyBle_ReadAuthPayloadTimeout (uint8 *bdHandle*, uint16 **authPayloadTimeout*)**

This function reads the Authentication Payload timeout set in BLE Controller for LE_PING feature Refer Bluetooth 4.1 core specification, Volume 6, Part B, section 4.6.5 for LE Ping operation.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |

| | |
|---|---|
| *authPayload Timeout* | Pointer to a variable to which authentication timeout value, read from BLE Controller, is written. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | One of the input parameters is invalid. |
| CYBLE_ERROR_INVALID_OPERATION | Operation is not permitted. |
| CYBLE_ERROR_NO_CONNECTION | When controller can't find active connection using given bdHandle |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Invalid bdHandle or LE connection doesn't exist for link identified by bdHandle. |

### CYBLE_API_RESULT_T CyBle_GetStackLibraryVersion (CYBLE_STACK_LIB_VERSION_T *stackVersion)

This function retrieves the version information of the BLE Stack library. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *stackVersion* | Pointer to a variable of type CYBLE_STACK_LIB_VERSION_T containing the version information of the CYBLE Stack library. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | stackVersion is NULL. |

### CYBLE_API_RESULT_T CyBle_IsStackIdle (void )

This function is used to check BLE stack is idle or not. This API function returns CYBLE_ERROR_OK if BLE Stack is idle. This function returns CYBLE_ERROR_STACK_BUSY if L2CAP TX data is queued for transmission, or any tasks are pending or hardware is busy. This function will not consider Rx path to decide stack is idle or not.

Note: This API function should not be called from BLE Stack callback context.

Use case example: Application can check before shut-down, BLE stack is idle or not.

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | If Stack is idle |
| CYBLE_ERROR_STACK_BUSY | If Stack is not idle. |

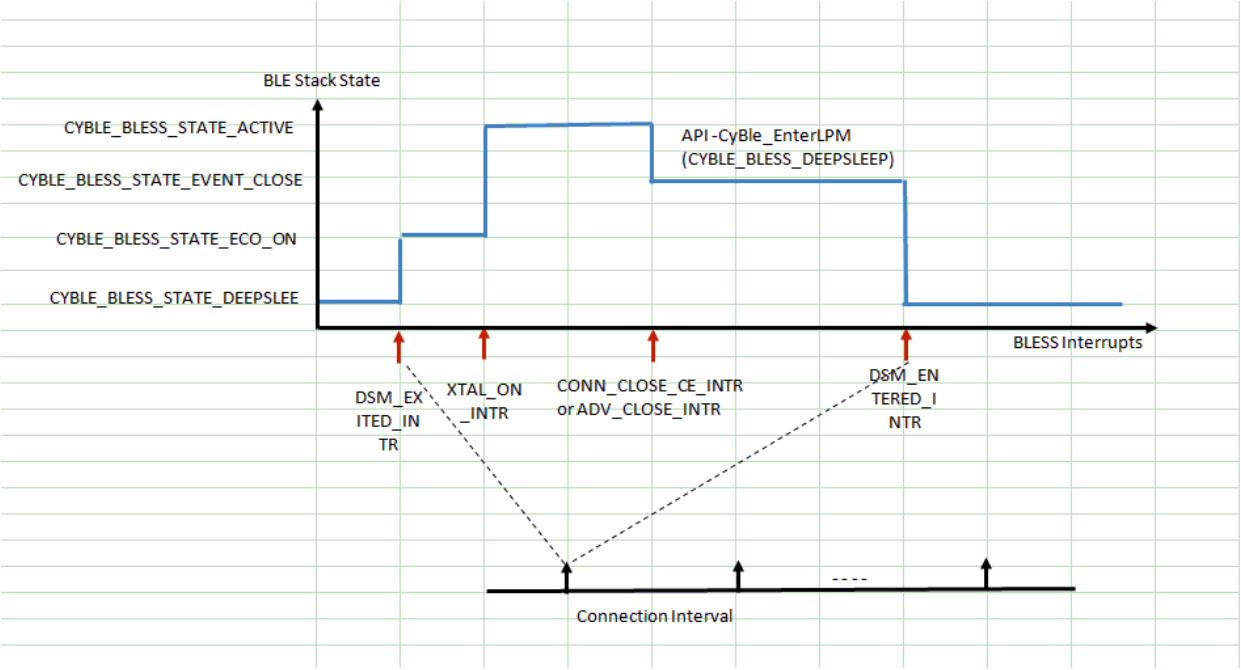
### CYBLE_BLESS_STATE_T CyBle_GetBleSsState (void )

This function gets the BLE Subsystem's current operational mode. This state can be used to manage system level power modes based on return value.

**Returns:**

CYBLE_BLESS_STATE_T bleStackMode: CYBLE_BLESS_STATE_T has one of the following modes

| BLE Stack Mode | Description |
|---|---|
| CYBLE_BLESS_STATE_ACTIVE | BLE Sub System is in active mode, CPU can be in active mode or sleep mode. |
| CYBLE_BLESS_STATE_EVENT_CLOSE | BLE Sub System radio and Link Layer hardware finishes Tx/Rx. After this state application can try putting BLE to Deep Sleep State to save power in rest of the BLE transmission event. |
| CYBLE_BLESS_STATE_SLEEP | BLE Sub System is in sleep mode, CPU can be in sleep mode. |
| CYBLE_BLESS_STATE_ECO_ON | BLE Sub System is in process of wakeup from Deep Sleep Mode and ECO(XTAL) is turned on. CPU can be put in Deep Sleep Mode. |
| CYBLE_BLESS_STATE_ECO_STABLE | BLE Sub System is in process of wakeup from Deep Sleep Mode and ECO(XTAL) is stable. CPU can be put in sleep mode. |
| CYBLE_BLESS_STATE_DEEPSLEEP | BLE Sub System is in Deep Sleep Mode. CPU can be put in Deep Sleep Mode. |
| CYBLE_BLESS_STATE_HIBERNATE | BLE Sub System is in Hibernate Mode. CPU can be put in Deep Sleep Mode. |




**void CyBle_AesCcmInit (void )**

This function initializes the clocks and registers needed to used AEC CCM encryption / decryption functionality without initializing the complete BLE Stack. This function must be called before calling CyBle_AesCcmEncrypt and/or CyBle_AesCcmDecrypt function. This is a blocking function. No event is generated on calling this function.

**Returns:**

None

**CYBLE_API_RESULT_T CyBle_AesCcmEncrypt (uint8 **key*, uint8 **nonce*, uint8 **in_data*, uint8 *length*, uint8 **out_data*, uint8 **out_mic*)**

This function encrypts the given data. This function can only be invoked after invoking 'CyBle_AesCcmInit' function. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *key* | Pointer to an array of bytes holding the key. The array length to be allocated by the application should be 16 bytes. |
| *nonce* | Pointer to an array of bytes. The array length to be allocated by the application is 13 Bytes. |
| *in_data* | Pointer to an array of bytes to be encrypted. Size of the array should be equal to the value of 'length' parameter. |
| *length* | Length of the data to be encrypted, in Bytes. Valid value range is 1 to 27. |
| *out_data* | Pointer to an array of size 'length' where the encrypted data is stored. |
| *out_mic* | Pointer to an array of bytes (4 Bytes) to store the MIC value generated during encryption. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | One of the inputs is a null pointer or the 'length' value is invalid |

**CYBLE_API_RESULT_T CyBle_AesCcmDecrypt (uint8 **key*, uint8 **nonce*, uint8 **in_data*, uint8 *length*, uint8 **out_data*, uint8 **in_mic*)**

This function decrypts the given data. This function can only be invoked after invoking 'CyBle_AesCcmInit' function. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *key* | Pointer to an array of bytes holding the key. The array length to be allocated by the application should be 16 bytes. |
| *nonce* | Pointer to an array of bytes. The array length to be allocated by the application is 13 Bytes. |
| *in_data* | Pointer to an array of bytes to be decrypted. Size of the array should be equal to the value of 'length' parameter. |
| *length* | Length of the data to be decrypted, in Bytes. Valid value range is 1 to 27. |
| *out_data* | Pointer to an array of size 'length' where the decrypted data is stored. |
| *in_mic* | Pointer to an array of bytes (4 Bytes) to provide the MIC value generated during encryption. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | One of the inputs is a null pointer or the 'length' value is invalid |
| CYBLE_ERROR_MIC_AUTH_FAILED | Data decryption has been done successfully but MIC based authorization check has failed. |

| Error codes | Description |
|---|---|
| | This error can be ignored if MIC based authorization was not intended. |

**CYBLE_API_RESULT_T CyBle_GenerateAesCmac (CYBLE_AES_CMAC_GENERATE_PARAM_T **cmacGenParam*)**

This API function enables the application to generate the AES CMAC of 16 bytes, for given variable length message and CMAC Key.

After this API function call, if the return value is CYBLE_ERROR_OK, then callback given in the input parameter is called when the cmac generation is completed. Once this callback is called, check the output parameter cmac to get the generated cmac value.

**Parameters:**

| | |
|---|---|
| *cmacGenParam* | pointer to structure containing parameters required for AES CMAC Generation. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | cmacGenParam is NULL or key is NULL or mac, output parameter is NULL or appl_callback is NULL or if buffer is NULL when size is greater than zero |
| CYBLE_ERROR_STACK_INTERNAL | An error occurred in BLE stack |

**CYBLE_API_RESULT_T CyBle_SetAppEventMask (uint32 *UserEventMask*)**

This API function enables the application to Mask which Events user wants to receive

Currently supporting maskable events CYBLE_EVT_GAP_CONN_ESTB CYBLE_EVT_GAP_SCAN_REQ_RECVD

**Parameters:**

| | |
|---|---|
| *UserEventMask* | User Event Mask |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | UserEventMask is ZERO |

**CYBLE_API_RESULT_T CyBle_RegisterBlessInterruptCallback (CYBLE_BLESS_EVENT_PARAM_T **BlessEventParams*)**

This API function will registers the callback function for BLESS Events and sets Event mask which BLESS Events user wants to receive

Currently supporting events CYBLE_ISR_BLESS_CONN_CLOSE_CE CYBLE_ISR_BLESS_ADV_CLOSE

Note: Application has to pay utmost care about not doing delayed processing in event handler as the registered callback will get called from BLESS Interrupt Service Routine.

Application can set/clear flag which can be used for further processing outside of the ISR context.

Event received through callback represents events received as a whole at that point i.e., application won't receive individual events.

**Parameters:**

| | |
|---|---|
| *BlessEventParams* | pointer to structure CYBLE_BLESS_EVENT_PARAM_T |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | If NULL passed |

### void CyBle_SetTxGainMode (uint8 *bleSsGainMode*)

This function configures the Tx gain mode for BLESS radio for Tx operation.

**Parameters:**

| | |
|---|---|
| *bleSsGainMode* | Gain mode setting for the output power |

| BLESS RD Gain Mode | Description |
|---|---|
| CYBLE_BLESS_NORMAL_GAIN_MODE | 0x00u - BLESS Normal Gain Mode Tx Pwr Range -18dBm to 0 dBm Normal Rx Sensitivity |
| CYBLE_BLESS_HIGH_GAIN_MODE | 0x01u - BLESS High Gain Mode Tx Pwr Range -18dBm to 3 dBm 3 dBm Additional Rx Sensitivity |

**Returns:**

none

### void CyBle_SetRxGainMode (uint8 *bleSsGainMode*)

This function configures the Rx gain mode to select Higher or Lower Receive Sensitivity for BLESS radio.

**Parameters:**

| | |
|---|---|
| *bleSsGainMode* | Gain mode setting for the Receiver Sensitivity. |

| BLESS RD Gain Mode | Description |
|---|---|
| CYBLE_BLESS_NORMAL_GAIN_MODE | 0x00u - BLESS Normal Gain Mode. Rx Sensitivity of -89dBm. |
| CYBLE_BLESS_HIGH_GAIN_MODE | 0x01u - BLESS High Gain Mode. Rx Sensitivity of -91dBm. |

**Returns:**

none

**CYBLE_API_RESULT_T CyBle_SetSlaveLatencyMode (uint8 *bdHandle*, uint8 *setForceQuickTransmit*)**

This function overrides the default BLE Stack behavior for LE connection that is established with non zero slave latency. This API function can be used by application to force set quick transmission for a link related to specified 'bdHandle' during slave latency period.

If the force quick transmit option is selected, the device will always respond all the Connection Events (CE) ignoring the slave latency. To re-enable BLE Stack control quick transmit behavior application should call this API function with force quick transmit option set to zero.

BLE Stack Control Policy: BLE Stack enables quick transmission whenever any data packet is queued in link layer. Upon successful transmission of data packet BLE Stack resets the quick transmit to enable latency for power save.

BLE Stack also enables quick transmit whenever any real time LL Control PDU is received. Once the acknowledgment of the PDU is processed the quick transmit option is reset.

**Parameters:**

| | |
|---|---|
| *bdHandle* | bdHandle identifying LE connection for which force quick transmit option is to be set or reset. |
| *setForceQuickTransmit* | This parameter is used to set or reset the force quick transmit configuration in BLE Stack.<br>• '1': Set the quick transmit behavior, it gets set immediately and disables over the air slave latency . This quick transmit setting remains true until application gives control to BLE Stack for controlling quick transmit bit.<br>• '0': Reset the force quick transmit behavior in BLESS to allow BLE Stack to control quick transmit behavior when slave latency is applied. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_NO_CONNECTION | When controller can't find active connection using given bdHandle |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Invalid bdHandle or LE connection doesn't exist for link identified by bdHandle. |

**void CyBle_SetSeedForRandomGenerator (uint32 *seed*)**

As per security specification of Bluetooth, BLE stack uses pseudo random number generator (Bluetooth core specification 4.2, Vol.2 Part H, Sec-2). Application can generate random number using API function CyBle_GenerateRandomNumber. Seed for random number generator with better entropy for randomness can be provided by application using this API function. This function sets application specific seed for DRBG (Deterministic Random number generator).

**Parameters:**

| | |
|---|---|
| *seed* | Seed for DRBG. Setting the seed to zero is functionally equivalent to not setting the application specific seed. |

**Returns:**

None.

**CYBLE_API_RESULT_T CyBle_IsLLControlProcPending (void )**

This function checks the Link Layer state for any pending real time control (LL_CHANNEL_MAP, LL_CONNECTION_UPDATE) procedure. When any such procedure is pending in Link layer busy state it is indicated by Link Layer.

Application using specific GAP API functions or L2CAP API functions that can result in initiation of real time procedures such as LL_CHANNEL_MAP, LL_CONNECTION_UPDATE can check the state of Link Layer to avoid any such rejection from BLE Stack.

BLE Stack can reject the new request If any LL control procedure is pending for completion this API function will return CYBLE_ERROR_CONTROLLER_BUSY.

**Returns:**

CYBLE_API_RESULT_T: Return value indicates the Link Layer status for any pending real time procedure.

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | Link Layer is Free. |
| CYBLE_ERROR_CONTROLLER_BUSY | Link Layer Control Procedure is pending, no new LL control procedure can be initiated. |

**CYBLE_API_RESULT_T CyBle_StartTransmitterTest (CYBLE_TRANSMITTER_TEST_PARAMS_T **TransmitterTestParams*)**

This API function Programs direct test mode TX test command parameters.

**Parameters:**

| | |
|---|---|
| *TransmitterTestParams* | pointer to structure CYBLE_TRANSMITTER_TEST_PARAMS_T. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | TransmitterTestParams is NULL |

**CYBLE_API_RESULT_T CyBle_StartReceiverTest (uint8 *RxFreq*)**

This API function Programs direct test mode RX test command parameters.

**Parameters:**

| | |
|---|---|
| *RxFreq* | Frequency for reception. N = (F 2402)/2 Range: 0x00 0x27. Frequency Range : 2402 MHz to 2480 MHz. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | RxFreq is Out of Range |

**CYBLE_API_RESULT_T CyBle_TestEnd (uint16 **PacketCount*)**

This API function Programs the direct test end command to the hardware, it reads number of successful packtes received from ll hardware.

**Parameters:**

| | |
|---|---|
| *PacketCount* | Pointer to a buffer of size 16 bytes in which the received number of successful packets will be stored. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | PacketCount is NULL |

### CYBLE_API_RESULT_T CyBle_HciSendPacket (CYBLE_HCI_PKT_PARAMS_T *HciPktParams)

This API function Sends HCI packet to Controller

User should deallocate memory buffer passed as an input parameter, after receiving an Event from the controller for command packet and after recieving Number Of Completed Packets event for data packet transmitted.

**Parameters:**

| | |
|---|---|
| *HciPktParams* | pointer to structure CYBLE_HCI_PKT_PARAMS_T. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | HciCmdParams is NULL |
| CYBLE_ERROR_INVALID_OPERATION | Operation not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_StoreStackData (uint8 *isForceWrite*)

This function instructs Stack to backup Stack internal RAM data into flash. This API function must be called by application to backup stack data. If this API function is not called appropriately, stack internal data structure will not be available on power cycle.

**Parameters:**

| | |
|---|---|
| *isForceWrite* | If value is set to 0, then stack will check if flash write is permissible. If value is set to 1, application should exit low power mode by calling CyBle_ExitLPM(). |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_FLASH_WRITE_NOT_PERMITED | Flash Write is not permitted or not completely written |

**CYBLE_API_RESULT_T CyBle_StoreAppData (uint8 **srcBuff*, const uint8 *destAddr*[], uint32 *buffLen*, uint8 *isForceWrite*)**

This function instructs the Stack to backup application specific data into flash. This API function must be called by application to backup application specific data.

**Parameters:**

| | |
|---|---|
| *srcBuff* | Source buffer |
| *destAddr* | Destination address |
| *buffLen* | Length of srcData |
| *isForceWrite* | If value is set to 0, then stack will check if flash write is permissible. If value is set to 1, application should exit low power mode by calling CyBle_ExitLPM() |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_FLASH_WRITE_NOT_PERMITED | Flash Write is not permitted |
| CYBLE_ERROR_INVALID_PARAMETER | Invalid input parameter |
| CYBLE_ERROR_FLASH_WRITE | Error in flash Write |

# GAP Functions

## Description

The GAP APIs allow access to the Generic Access Profile (GAP) layer of the BLE stack. Depending on the chosen GAP role in the GUI, you may use a subset of the supported APIs.

The GAP API names begin with CyBle_Gap. In addition to this, the APIs also append the GAP role initial letter in the API name.

## Modules

- GAP Central and Peripheral Functions
  *These are APIs common to both GAP Central role and GAP Peripheral role. You may use them in either roles.*
- GAP Central Functions
  *APIs unique to designs configured as a GAP Central role.*
- GAP Peripheral Functions
  *APIs unique to designs configured as a GAP Peripheral role.*
- GAP Definitions and Data Structures
  *Contains the GAP specific definitions and data structures used in the GAP APIs.*

# GAP Central and Peripheral Functions

## Description

These are APIs common to both GAP Central role and GAP Peripheral role. You may use them in either roles.

No letter is appended to the API name: CyBle_Gap

## Functions

- CYBLE_API_RESULT_T CyBle_GapSetLocalName (const char8 name[])
- CYBLE_API_RESULT_T CyBle_GapGetLocalName (char8 name[])
- CYBLE_API_RESULT_T CyBle_GapSetIoCap (CYBLE_GAP_IOCAP_T ioCap)
- CYBLE_API_RESULT_T CyBle_GapSetSecurityRequirements (uint8 secReq, uint8 encKeySize)
- CYBLE_API_RESULT_T CyBle_GapSetOobData (uint8 bdHandle, uint8 oobFlag, uint8 *key, uint8 *oobData, uint8 *oobDataLen)
- CYBLE_API_RESULT_T CyBle_GapGetPeerBdAddr (uint8 bdHandle, CYBLE_GAP_BD_ADDR_T *peerBdAddr)
- CYBLE_API_RESULT_T CyBle_GapGetPeerBdHandle (uint8 *bdHandle, CYBLE_GAP_BD_ADDR_T *peerBdAddr)
- CYBLE_API_RESULT_T CyBle_GapGetPeerDevSecurity (uint8 bdHandle, CYBLE_GAP_AUTH_INFO_T *security)
- CYBLE_API_RESULT_T CyBle_GapDisconnect (uint8 bdHandle)
- CYBLE_API_RESULT_T CyBle_GapDisconnectWithReason (uint8 bdHandle, uint8 reason)
- CYBLE_API_RESULT_T CyBle_GapGetPeerDevSecurityKeyInfo (uint8 bdHandle, uint8 *keysFlag, CYBLE_GAP_SMP_KEY_DIST_T *keyInfo)
- CYBLE_API_RESULT_T CyBle_GapGenerateDeviceAddress (CYBLE_GAP_BD_ADDR_T *bdAddr, CYBLE_GAP_ADDR_TYPE_T addrType, uint8 *irk)
- CYBLE_API_RESULT_T CyBle_GapSetSecurityKeys (uint8 keysFlag, CYBLE_GAP_SMP_KEY_DIST_T *keyInfo)
- CYBLE_API_RESULT_T CyBle_GapGenerateKeys (uint8 keysFlag, CYBLE_GAP_SMP_KEY_DIST_T *keyInfo)
- CYBLE_API_RESULT_T CyBle_GapAuthReq (uint8 bdHandle, CYBLE_GAP_AUTH_INFO_T *authInfo)
- CYBLE_API_RESULT_T CyBle_GapAuthPassKeyReply (uint8 bdHandle, uint32 passkey, uint8 accept)
- CYBLE_API_RESULT_T CyBle_GapRemoveDeviceFromWhiteList (CYBLE_GAP_BD_ADDR_T *bdAddr)
- CYBLE_API_RESULT_T CyBle_GapAddDeviceToWhiteList (CYBLE_GAP_BD_ADDR_T *bdAddr)
- CYBLE_API_RESULT_T CyBle_GapGetBondedDevicesList (CYBLE_GAP_BONDED_DEV_ADDR_LIST_T *bondedDevList)
- CYBLE_API_RESULT_T CyBle_GapRemoveOldestDeviceFromBondedList (void)
- CYBLE_API_RESULT_T CyBle_GapGetDevSecurityKeyInfo (uint8 *keyFlags, CYBLE_GAP_SMP_KEY_DIST_T *keys)
- CYBLE_API_RESULT_T CyBle_GapGetDevicesFromWhiteList (uint8 *count, CYBLE_GAP_BD_ADDR_T *addr)
- CYBLE_API_RESULT_T CyBle_GapGetChannelMap (uint8 bdHandle, uint8 *channelMap)
- CYBLE_API_RESULT_T CyBle_GapSetSecureConnectionsOnlyMode (uint8 state)
- CYBLE_API_RESULT_T CyBle_GapGenerateLocalP256Keys (void)
- CYBLE_API_RESULT_T CyBle_GapSetLocalP256Keys (CYBLE_GAP_SMP_LOCAL_P256_KEYS *localP256Keys, uint8 isValidateKeys)
- CYBLE_API_RESULT_T CyBle_GapAuthSendKeyPress (uint8 bdHandle, CYBLE_GAP_KEYPRESS_NOTIFY_TYPE notificationType)
- CYBLE_API_RESULT_T CyBle_GapGenerateOobData (const uint8 *pRand)
- CYBLE_API_RESULT_T CyBle_GapSetDataLength (uint8 bdHandle, uint16 connMaxTxOctets, uint16 connMaxTxTime)
- CYBLE_API_RESULT_T CyBle_GapSetRxDataLength (CYBLE_GAP_RX_DATA_LENGTH_T *RxDleParams)
- CYBLE_API_RESULT_T CyBle_GapSetSuggestedDataLength (uint16 suggestedTxOctets, uint16 suggestedTxTime)
- CYBLE_API_RESULT_T CyBle_GapGetDataLength (CYBLE_GAP_DATA_LENGTH_T *readParam)
- CYBLE_API_RESULT_T CyBle_GapConvertOctetToTime (CYBLE_GAP_PHY_TYPE_T phy, uint16 octets, uint16 *pTime)

- CYBLE_API_RESULT_T CyBle_GapAddDeviceToResolvingList (const CYBLE_GAP_RESOLVING_DEVICE_INFO_T *rpaInfo)
- CYBLE_API_RESULT_T CyBle_GapRemoveDeviceFromResolvingList (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr)
- CYBLE_API_RESULT_T CyBle_GapClearResolvingList (void)
- CYBLE_API_RESULT_T CyBle_GapReadPeerResolvableAddress (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr, uint8 *peerResolvableAddress)
- CYBLE_API_RESULT_T CyBle_GapReadLocalResolvableAddress (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr, uint8 *localResolvableAddress)
- CYBLE_API_RESULT_T CyBle_GapSetResolvablePvtAddressTimeOut (uint16 rpaTimeOut)
- CYBLE_API_RESULT_T CyBle_GapReadResolvingList (CYBLE_GAP_RESOLVING_LIST_T *resolvingList)
- CYBLE_API_RESULT_T CyBle_GapSetAddressResolutionEnable (uint8 enableDisable)
- CYBLE_API_RESULT_T CyBle_GapSetPrivacyMode (CYBLE_GAP_PRIVACY_MODE_INFO_T *privacyModeInfo)
- CYBLE_API_RESULT_T CyBle_GapGetBondedDevicesByRank (CYBLE_GAP_DEVICE_ADDR_LIST_T *bondedDevList)
- CYBLE_API_RESULT_T CyBle_GapSetLeEventMask (uint8 *hciLeEventMask)
- CYBLE_API_RESULT_T CyBle_GapSetIdAddress (const CYBLE_GAP_BD_ADDR_T *bdAddr)
- CYBLE_API_RESULT_T CyBle_GapGenerateAndSetIrk (uint8 keysFlag, uint8 *irk)
- CYBLE_API_RESULT_T CyBle_GapFixAuthPassKey (uint8 isFixed, uint32 fixedPassKey)
- CYBLE_API_RESULT_T CyBle_GappSetNumOfAdvPkts (CYBLE_GAPP_DISC_MODE_INFO_T *advInfo, uint16 NumOfAdvPkts)

## Function Documentation

### CYBLE_API_RESULT_T CyBle_GapSetLocalName (const char8 *name*[])

This function is used to set the local device name - a Characteristic of the GAP Service. If the characteristic length entered in the component customizer is shorter than the string specified by the "name" parameter, the local device name will be cut to the length specified in the customizer.

**Parameters:**

| | |
|---|---|
| *name* | The local device name string. The name string to be written as the local device name. It represents a UTF-8 encoded User Friendly Descriptive Name for the device. The length of the local device string is entered into the component customizer and it can be set to a value from 0 to 248 bytes. If the name contained in the parameter is shorter than the length from the customizer, the end of the name is indicated by a NULL octet (0x00). |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | Function completed successfully. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter |

### CYBLE_API_RESULT_T CyBle_GapGetLocalName (char8 *name*[])

This function is used to read the local device name - a Characteristic of the GAP Service.

**Parameters:**

| | |
|---|---|
| *name* | The local device name string. Used to read the local name to the given string array. It represents a UTF-8 encoded User Friendly Descriptive Name for the device. The length of the local device string is entered into the component customizer and it can be set to a value from 0 to 248 bytes. If the name contained in the parameter is shorter than the length from the customizer, the end of the name is indicated by a NULL octet (0x00). |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | Function completed successfully. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter |

**CYBLE_API_RESULT_T CyBle_GapSetIoCap (CYBLE_GAP_IOCAP_T *ioCap*)**

This function sets the input and output capability of the BLE Device that is used during authentication procedure. This is a blocking function. No event is generated on calling this function. The input capabilities are described in the following table:

| Capability | Description |
|---|---|
| No input | Device does not have the ability to indicate "yes" or "no" |
| Yes/No | Device has at least two buttons that can be easily mapped to "yes" and "no" or the device has a mechanism whereby the user can indicate either "yes" or "no". |
| Keyboard | Device has a numeric keyboard that can input the numbers "0" through "9" and a confirmation. Device also has at least two buttons that can be easily mapped to "yes" and "no" or the device has a mechanism whereby the user can indicate either "yes" or "no". |

The output capabilities are described in the following table:

| Capability | Description |
|---|---|
| No output | Device does not have the ability to display or communicate a 6 digit decimal number. |
| Numeric output | Device has the ability to display or communicate a 6 digit decimal number. |

Combined capability is defined in the following table:

| Input Capability | No Output | Numeric Output |
|---|---|---|
| No input | NoInputNoOutput | DisplayOnly |
| Yes/No | NoInputNoOutput | DisplayYesNo |
| Keyboard | KeyboardOnly | KeyboardDisplay |

Refer Bluetooth 4.1 core specification, Volume 3, Part C, section 5.2.2.4 for more details on the IO capabilities. IO capabilities of the BLE devices are used to determine the pairing method. Please refer Bluetooth 4.1 core specification, Volume 3, Part H, section 2.3.5.1 for more details on the impact of IO capabilities on the pairing method chosen.

**Parameters:**

| | |
|---|---|
| *ioCap* | IO Capability of type CYBLE_GAP_IOCAP_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying invalid input parameter |

**CYBLE_API_RESULT_T CyBle_GapSetSecurityRequirements (uint8 *secReq*, uint8 *encKeySize*)**

This function is used to set the security requirements of local device and encryption key size requirement of the local device. This is a blocking function. No event is generated on calling this function. It is expected to call this API function on host stack on, though can be called at any point except when any of SMP procedure is in progress. Security requirements are defined in the following table:

| Security Requirement | |
|---|---|
| CYBLE_GAP_NO_SECURITY_REQUIREMENTS | Default :security requirement specifies there are no security requirements |
| CYBLE_GAP_SEC_UNAUTH_PAIRING | Bit 0: Legacy pairing with NO MITM protection |
| CYBLE_GAP_SEC_AUTH_PAIRING | Bit 1: Legacy pairing with MITM protection |
| CYBLE_GAP_SEC_SC_PAIRING_WITH_NO_MITM | Bit 2: Secured Connection pairing with NO MITM protection |
| CYBLE_GAP_SEC_SC_PAIRING_WITH_MITM | Bit 3: Secured Connection pairing with MITM protection |
| CYBLE_GAP_SEC_OOB_IN_LEGACY_PAIRING | Bit 4: Legacy pairing with OOB method |
| CYBLE_GAP_SEC_OOB_IN_SC_PAIRING | Bit 5: Secured Connection pairing with OOB method |

After this API function is called, BLE Stack will check whether the received security request or pairing request or pairing response satisfies local device security requirements that are set using this API function. If local device security requirements are not met then pairing is rejected by the BLE stack.

Eg: CyBle_GapSetSecurityRequirements() is called with secReq as CYBLE_GAP_SEC_SC_PAIRING_WITH_MITM. Now if BLE Stack receives any pairing request with SC bit and MITM bit are not set, then that pairing request will be rejected by the stack.

Note: If the secured connection only mode is set, then these security requirements are not considered during pairing procedure. This is to maintain BWC for SC Only mode.

**Parameters:**

| | |
|---|---|
| *secReq* | Security requirement is a bit-field parameter. Application can set this value with the above defined values in the table. Application can set multiple security requirements by ORing them in this parameter. Eg: If secReq is (CYBLE_GAP_SEC_UNAUTH_PAIRING \| CYBLE_GAP_SEC_SC_PAIRING_WITH_NO_MITM), then stack |

| | |
|---|---|
| | allows pairing only if received pairing request is either Legacy pairing with NO MITM or Secured Connection pairing with NO MITM. |
| *encKeySize* | Encryption key size requirement of the local device. This parameter does not affect anything on central side. At peripheral side, when encryption key size is set using this API function, then after during pairing if negotiated key size is less than the key size set by this API function, then BLE Stack will reject that pairing request. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying invalid input parameter |

**CYBLE_API_RESULT_T CyBle_GapSetOobData (uint8 *bdHandle*, uint8 *oobFlag*, uint8 **key*, uint8 **oobData*, uint8 **oobDataLen*)**

This function sets OOB presence flag and data. This function should be used by the application layer if it wants to enable OOB bonding procedure for any specific device identified by "bdHandle". This function should be called before initiating authentication or before responding to authentication request to set OOB flag and data. For more details on OOB, please refer Bluetooth 4.1 core specification, Volume 1, Part A, section 5.2.4.3. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device for which the Out Of Band signaling (OOB) configuration is to be used. |
| *oobFlag* | OOB data presence flag. Allowed value are,<br>• CYBLE_GAP_OOB_DISABLE<br>• CYBLE_GAP_OOB_ENABLE |
| *key* | 16 Octet Temporary Key, to be used for OOB authentication. |
| *oobData* | Pointer to OOB data. In case of Legacy Pairing this parameter is not used for OOB authentication. |
| *oobDataLen* | Pointer to a variable to store OOB data length. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter |
| CYBLE_ERROR_NO_DEVICE_ENTITY | 'bdHandle' does not represent known device entity |

**CYBLE_API_RESULT_T CyBle_GapGetPeerBdAddr (uint8 *bdHandle*, CYBLE_GAP_BD_ADDR_T **peerBdAddr*)**

This function reads the peer Bluetooth device address which has already been fetched by the BLE Stack. 'peerBdAddr' stores the peer's Bluetooth device address identified with 'bdHandle'. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *peerBdAddr* | Empty buffer where the Bluetooth device address gets stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'peerBdAddr'. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Specified device handle does not map to any device handle entry in BLE stack. |

**CYBLE_API_RESULT_T CyBle_GapGetPeerBdHandle (uint8 **bdHandle*, CYBLE_GAP_BD_ADDR_T **peerBdAddr*)**

This function reads the device handle of the remote Bluetooth device using 'peerBdAddr', which has already been fetched by the BLE Stack. 'bdHandle' stores the peer device handle. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Pointer to a variable to store peer device handle |
| *peerBdAddr* | Pointer to Bluetooth device address of peer device of type CYBLE_GAP_BD_ADDR_T, to be provided to this function as an input |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'peerBdAddr' or 'bdHandle'. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Specified device handle does not map to any device handle entry in BLE stack. |

**CYBLE_API_RESULT_T CyBle_GapGetPeerDevSecurity (uint8 *bdHandle*, CYBLE_GAP_AUTH_INFO_T **security*)**

This function enables the application to get the device security of the peer device, which has already been fetched by the BLE Stack, identified using 'bdHandle' when the peer device is in the trusted list. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *security* | Pointer to a buffer into which security information will be written. security level of the peer device is provided in CYBLE_GAP_AUTH_INFO_T->security. It ignores LE Security mode. Security should be interpreted as MITM and no MITM as encryption is always supported if pairing is performed between two devices. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'security'. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Specified device handle does not map to any device handle entry in BLE stack. |

### CYBLE_API_RESULT_T CyBle_GapDisconnect (uint8 *bdHandle*)

This function disconnects the peer device. It is to be used by the device in GAP Central mode and may be used by a GAP Peripheral device to send a disconnect request. This is a non-blocking function. On disconnection, the following events are generated, in order.

- CYBLE_EVT_GATT_DISCONNECT_IND
- CYBLE_EVT_GAP_DEVICE_DISCONNECTED

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | No device to be disconnected. The specified device handle does not map to any device entry in the BLE Stack. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

### CYBLE_API_RESULT_T CyBle_GapDisconnectWithReason (uint8 *bdHandle*, uint8 *reason*)

This function allows to disconnect the peer device with reason code. It is to be used by the device in GAP Central mode and may be used by a GAP Peripheral device to send a disconnect request. This is a non-blocking function. On disconnection, the following events are generated, in order.

- CYBLE_EVT_GATT_DISCONNECT_IND
- CYBLE_EVT_GAP_DEVICE_DISCONNECTED

Note: If the reason code is not valid, then by default reason code sent is 0x13

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *reason* | Reason for the disconnect. Refer Volume 2, Part E, section 7.1.6 for the reason codes. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |

| Error codes | Description |
|---|---|
| CYBLE_ERROR_INVALID_PARAMETER | No device to be disconnected. The specified device handle does not map to any device entry in the BLE Stack. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

**CYBLE_API_RESULT_T CyBle_GapGetPeerDevSecurityKeyInfo (uint8 *bdHandle*, uint8 **keysFlag*, CYBLE_GAP_SMP_KEY_DIST_T **keyInfo*)**

This function enables the application to know the keys shared by a given peer device upon completion of the security sequence (already fetched by the BLE Stack). The keys are shared by the peer device on initiation of authentication which is performed using the CyBle_GapAuthReq() or CyBle_GappAuthReqReply() function.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *keysFlag* | Indicates the keys to be retrieved from peer device. The following bit fields indicate the presence or absence of the keys distributed.<br>**Negotiated Local/Peer Key distribution**<br>• Bit 0. Encryption information (LTK and MID Information)<br>• Bit 1. Identity information<br>• Bit 2. Signature Key<br>• Bit 3-7. Reserved |
| *keyInfo* | Pointer to variable of type CYBLE_GAP_SMP_KEY_DIST_T to copy the stored keys of the peer device identified by 'bdHandle' |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'keyInfo'. |
| CYBLE_ERROR_INVALID_OPERATION | An error occurred in BLE stack. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |

**CYBLE_API_RESULT_T CyBle_GapGenerateDeviceAddress (CYBLE_GAP_BD_ADDR_T **bdAddr*, CYBLE_GAP_ADDR_TYPE_T *addrType*, uint8 **irk*)**

This function generates either public or random address based on 'type' field of CYBLE_GAP_BD_ADDR_T structure. It uses BLE Controller's random number generator to generate the random part of the Bluetooth device address.

The parameter 'addrType' specifies further sub-classification within the public and random address types.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Bluetooth device address is generated and populated in the structure pointed to by this pointer. The structure is of type CYBLE_GAP_BD_ADDR_T. |

| | |
|---|---|
| *addrType* | Specifies the type of address. This can take one of the values from the enumerated data type CYBLE_GAP_ADDR_TYPE_T. |
| *irk* | Pointer to buffer containing 128-bit 'IRK' data. This parameter is only used when CYBLE_GAP_RANDOM_PRIV_RESOLVABLE_ADDR is the value set to 'addrType'. For other values of 'addrType', this parameter is not used. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_GapSetSecurityKeys (uint8 *keysFlag*, CYBLE_GAP_SMP_KEY_DIST_T **keyInfo*)**

This function sets the security keys that are to be exchanged with peer device during key exchange stage of authentication procedure and sets it in the BLE Stack. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *keysFlag* | This parameter indicates which keys get exchanged with peer device. The following is the bit field mapping for the keys.<br>• Bit 0: Local Encryption information<br>• Bit 1: Local Identity information<br>• Bit 2: Local Signature Key<br>• Bit 3: Reserved<br>• Bit 4: Remote Encryption information<br>• Bit 5: Remote Identity information<br>• Bit 6: Remote Signature Key<br>• Bit 7: Reserved |
| *keyInfo* | Pointer to a variable containing the keys to be set, of type 'CYBLE_GAP_SMP_KEY_DIST_T'. idAddrInfo param of 'CYBLE_GAP_SMP_KEY_DIST_T' will be ignored. 'CyBle_GapSetIdAddress' api needs to be used to set bd address. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'keyInfo' |

**CYBLE_API_RESULT_T CyBle_GapGenerateKeys (uint8 *keysFlag*, CYBLE_GAP_SMP_KEY_DIST_T **keyInfo*)**

This function generates and sets the security keys into BLE Stack that are to be exchanged with peer device during key exchange stage of authentication procedure. This is a blocking function. No event is generated on calling this function. This API function does not generate identity address (keyInfo->idAddrInfo)

**Parameters:**

| | |
|---|---|
| *keysFlag* | This parameter indicates which keys get exchanged with peer device. The following is the bit field mapping for the keys.<br>• Bit 0: Local Encryption information<br>• Bit 1: Local Identity information<br>• Bit 2: Local Signature Key<br>• Bit 3: Reserved<br>• Bit 4: Remote Encryption information<br>• Bit 5: Remote Identity information<br>• Bit 6: Remote Signature Key<br>• Bit 7: Reserved |
| *keyInfo* | Pointer to a variable containing the returned keys, of type 'CYBLE_GAP_SMP_KEY_DIST_T' |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'keyInfo' |

**CYBLE_API_RESULT_T CyBle_GapAuthReq (uint8 *bdHandle*, CYBLE_GAP_AUTH_INFO_T **authInfo*)**

This function starts authentication/pairing procedure with the peer device. It is a non-blocking function.

If the local device is a GAP Central, the pairing request is sent to the GAP Peripheral device. On receiving CYBLE_EVT_GAP_AUTH_REQ event, the GAP Peripheral is expected to respond by invoking the CyBle_GappAuthReqReply() function.

If the local device is GAP Peripheral, a Security Request is sent to GAP Central device. On receiving CYBLE_EVT_GAP_AUTH_REQ event, the GAP Central device is expected to respond by invoking 'CyBle_GapAuthReq ()' function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *authInfo* | Pointer to security information of the device of type CYBLE_GAP_AUTH_INFO_T. The 'authErr' parameter in CYBLE_GAP_AUTH_INFO_T should be ignored as it is not used in this function. NOTE: If the bonding flag in authInfo is set to CYBLE_GAP_BONDING_NONE then, SMP keys will not be distributed even if application has generated and set the keys explicitly. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying null pointer for 'advInfo' or if any of the element of this structure has an invalid value. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

| Error codes | Description |
|---|---|
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | On bonded device is full and application tries to initiate pairing with bonding enable. |

**CYBLE_API_RESULT_T CyBle_GapAuthPassKeyReply (uint8 *bdHandle*, uint32 *passkey*, uint8 *accept*)**

This function sends passkey for authentication. It is a non-blocking function.

It should be invoked in reply to the authentication request event CYBLE_EVT_GAP_PASSKEY_ENTRY_REQUEST received by the BLE Stack. This function is used to accept the passkey request and send the passkey or reject the passkey request.

- If the authentication operation succeeds, CYBLE_EVT_GAP_AUTH_COMPLETE is generated. If the authentication process times out, CYBLE_EVT_TIMEOUT event is generated.
- If the authentication fails, CYBLE_EVT_GAP_AUTH_FAILED event is generated.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *passkey* | 6-digit decimal number (authentication passkey) |
| *accept* | Accept or reject passkey entry request. Allowed values are,<br>• CYBLE_GAP_REJECT_PASSKEY_REQ<br>• CYBLE_GAP_ACCEPT_PASSKEY_REQ |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Invalid parameter. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |

**CYBLE_API_RESULT_T CyBle_GapRemoveDeviceFromWhiteList (CYBLE_GAP_BD_ADDR_T **bdAddr*)**

This function marks the device untrusted. It removes the bonding information of the device and removes it from the white list. More details on 'bonding' and 'trusted devices' is available in Bluetooth 4.1 core specification, Volume 3, Part C, section 9.4.4.

This is a blocking function. No event is generated on calling this function. This API function is kept as is for backward compatibility. This API function will be obsolete in future.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Pointer to peer device address, of type CYBLE_GAP_BD_ADDR_T. If device address is set to 0, then all devices shall be removed from trusted list and white list. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'bdAddr'. |
| CYBLE_ERROR_INVALID_OPERATION | Whitelist is already in use. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device does not exist in the whitelist. |

**CYBLE_API_RESULT_T CyBle_GapAddDeviceToWhiteList (CYBLE_GAP_BD_ADDR_T *bdAddr)**

This function adds the device to the whitelist. Maximum number of devices that can be added to the whitelist is eight including CYBLE_GAP_MAX_BONDED_DEVICE. Refer to Bluetooth 4.1 core specification, Volume 3, Part C, section 9.3.5 for more details on whitelist.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Peer device address, of type CYBLE_GAP_BD_ADDR_T. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'bdAddr' or 'bdAddr->type' has an invalid value |
| CYBLE_ERROR_INVALID_OPERATION | Whitelist is already in use |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | WhitelistMemory is full |
| CYBLE_ERROR_DEVICE_ALREADY_EXISTS | Matching device already exists in the whitelist |

**CYBLE_API_RESULT_T CyBle_GapGetBondedDevicesList (CYBLE_GAP_BONDED_DEV_ADDR_LIST_T *bondedDevList)**

This function returns the count and Bluetooth device address of the devices in the bonded device list. This is a blocking function. No event is generated on calling this function.

Application invoking this function should allocate sufficient memory for the structure CYBLE_GAP_BONDED_DEV_ADDR_LIST_T, where the complete list of bonded devices along with count can be written. Maximum devices bonded are specified by CYBLE_GAP_MAX_BONDED_DEVICE, which is a pre processing time parameter for the BLE Stack. Hence, the bonded device count will be less than or equal to CYBLE_GAP_MAX_BONDED_DEVICE.

Refer Bluetooth 4.1 core specification, Volume 3, Part C, section 9.4.4 for details on bonded devices.

**Parameters:**

| | |
|---|---|
| *bondedDevList* | Buffer to which list of bonded device list will be stored of type CYBLE_GAP_BONDED_DEV_ADDR_LIST_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

**CYBLE_API_RESULT_T CyBle_GapRemoveOldestDeviceFromBondedList (void )**

This function removes the oldest device from the bonded and white lists. This api should not be called while in connected state. If device is connected to the oldest device, and this API function is called, it will remove the device which is next oldest and not connected.

**Returns:**

CYBLE_API_RESULT_T : Return value indicates if the function succeeded (0x0000) or failed. Following are the possible error codes returned.

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_MAX | On failure operation. |

**CYBLE_API_RESULT_T CyBle_GapGetDevSecurityKeyInfo (uint8 **keyFlags*, CYBLE_GAP_SMP_KEY_DIST_T **keys*)**

This function gets the local device's Keys and key flags. The IRK received from this function should be used as the input IRK for the function 'CyBle_GapGenerateDeviceAddress' to generate Random Private Resolvable address. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *keyFlags* | Pointer to a byte where the key flags are stored. Based on the flag bits, the calling application can determine if the returned value is valid (1) or not (0). Key distribution flag<br>• Bit 0: Local Encryption information<br>• Bit 1: Local Identity information<br>• Bit 2: Local Signature Key<br>• Bit 3 - Bit 7: Reserved |
| *keys* | Pointer to a structure of type CYBLE_GAP_SMP_KEY_DIST_T where the keys get stored |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameters |

**CYBLE_API_RESULT_T CyBle_GapGetDevicesFromWhiteList (uint8 **count*, CYBLE_GAP_BD_ADDR_T **addr*)**

This function extracts the list of devices added to the white list. This is a blocking function. No events are generated on calling this function. There is no HCI command defined for this operation as the application is expected to keep track of the devices added to the white list. This function has been provided to facilitate testing of the Cypress BLE Hardware using CySmart tool.

**Parameters:**

| | |
|---|---|
| *count* | Pointer to a variable to hold the number of enabled addresses in the white list. This is an output parameter. |
| *addr* | Pointer to a variable of type 'CYBLE_GAP_BD_ADDR_T' which holds Address type and Address of the device. |

The function invoking this should allocate memory for the variables pointed to by the above pointers. 'addr' should point to an array of type CYBLE_GAP_BD_ADDR_T and size equal to the maximum number of white list devices supported by the BLE Stack (CYBLE_MAX_WHITELIST_ENTRIES).

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter(s) |

### CYBLE_API_RESULT_T CyBle_GapGetChannelMap (uint8 *bdHandle*, uint8 **channelMap*)

This function reads the channel map for data channels. This classification persists until it is overwritten by a subsequent call to this function or the controller is reset. If this command is used, updates should be sent within 10 seconds of the BLE Host knowing that the channel classification has changed. The interval between two successive commands sent will be at least one second. This command will only be used when the local device supports the Master role.

For details, refer to Bluetooth core specification 4.1, Volume 2, part E, section 7.8.19.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *channelMap* | This parameter contains five octet byte stream (Least Significant Byte having the bit fields 0 to 7, most significant byte having the bit fields 32 to 36). The nth such field (in the range 0 to 36) contains the value for the link layer channel index n. Allowed values and their interpretation are,<br>• Channel 'n' is bad = 0x00u<br>• Channel 'n' is unknown = 0x01u<br>The most significant bits (37 to 39) are reserved and will be set to 0. At least one channel will be marked as unknown. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'channelMap'. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

### CYBLE_API_RESULT_T CyBle_GapSetSecureConnectionsOnlyMode (uint8 *state*)

This API function sets the state of secure connections only mode for device. If device is in secure connections only mode, it will allow pairing to complete only with secure connections security. Other kind of pairing will lead to

pairing failure with reason "Authentication requirement not met" It is expected to call this API function on host stack on, though can be called at any point. Secure connections only is not persistent across power cycles. It is persistent across stack shutdown-init cycles.

**Parameters:**

| | |
|---|---|
| *state* | 0 - Disable (Device not in secure connections only mode)<br>1 - Enable (Device is in secure connections only mode) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_OPERATION | Secure connections feature was not selected in feature config and API function is called. |
| CYBLE_ERROR_INVALID_PARAMETER | parameter out of range |

### CYBLE_API_RESULT_T CyBle_GapGenerateLocalP256Keys (void )

This API function is used to generate P-256 Public-Private key pair to be used during LE Secure connection pairing procedure. Application may choose to generate P-256 public-private key pair before pairing process starts. If this API function is not called before pairing process starts, BLE Stack will use default public-private key pair. On the Completion of key generation, new keys will be set in the BLE Stack for SC pairing procedure and application receives CYBLE_EVT_GAP_SMP_LOC_P256_KEYS_GEN_AND_SET_COMPLETE event.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_OPERATION | Pairing is in progress. |

### CYBLE_API_RESULT_T CyBle_GapSetLocalP256Keys (CYBLE_GAP_SMP_LOCAL_P256_KEYS **localP256Keys*, uint8 *isValidateKeys*)

This API function is used to set P-256 Public-Private key pair to be used during LE Secure connection pairing procedure. Application may choose to set P-256 public-private key pair before pairing process starts. If this API function is not called before pairing process starts, BLE Stack will use default public-private key pair. This API function is not expected to be called when pairing procedure is in progress. Application can generate P-256 Public-Private key pair using API function CyBle_GapGenerateLocalP256Keys() and can set the generated key pair using this API function.

**Parameters:**

| | |
|---|---|
| *localP256Keys* | Pointer to structure CYBLE_GAP_SMP_LOCAL_P256_KEYS, that has fields for local P-256 public-private key pair. |
| *isValidateKeys* | If it is set to 1 public key is validated, if it is set to 0 public key is not validated. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_INVALID_PARAMETER | Parameter is NULL Or Public key is not valid |
| CYBLE_ERROR_INVALID_OPERATION | Pairing is in progress. |

**CYBLE_API_RESULT_T CyBle_GapAuthSendKeyPress (uint8 *bdHandle*, CYBLE_GAP_KEYPRESS_NOTIFY_TYPE *notificationType*)**

This API function is used to send LE Secure connections key press notification to peer device during secure connection pairing. This API function should be called by application to inform stack about passkey entry process started for each digit

- Started (0), entered (1), erased (2), cleared (3), completed (4). Once all the digits are entered, application needs to call 'CyBle_GapAuthPassKeyReply()' to inform stack for passkey enter completed. Error will be returned if key press entry bit was not set in 'pairingProperties' of CYBLE_GAP_AUTH_INFO_T during authentication procedure.

Typical application usage scenario:

1. Call with CYBLE_GAP_PASSKEY_ENTRY_STARTED on receiving event to enter passkey.
2. Call with CYBLE_GAP_PASSKEY_DIGIT_ENTERED, CYBLE_GAP_PASSKEY_DIGIT_ERASED or CYBLE_GAP_PASSKEY_CLEARED based on application events while user enters passkey.
3. Call with CYBLE_GAP_PASSKEY_ENTRY_COMPLETED after user application successfully received passkey.
4. This should be followed by call to CyBle_GapAuthPassKeyReply API function to provide user entered passkey to Stack.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *notificationType* | parameter of type 'CYBLE_GAP_KEYPRESS_NOTIFY_TYPE' |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | notificationType is invalid. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |
| CYBLE_ERROR_INVALID_OPERATION | Keypress was not negotiated or |

passkey entry procedure not ongoing or Secured Connection is not enabled or pairing procedure is not in progress.

**CYBLE_API_RESULT_T CyBle_GapGenerateOobData (const uint8 **pRand*)**

This API function is used to generate OOB data based on the input parameter (16 Byte random number) This API function is called to generate OOB data to be used by peer device. Peer device (or local device with peer's OOB data) will use 'CyBle_GapSetOobData()' to set the OOB data to be used for secure connections pairing.

Note: This API function should be used only in secured connection pairing. In case of legacy pairing only key is used for OOB authentication. But in SC pairing, key(pRand) is used to generate local OOB data(Confirm value).

In SC both key and generated OOB data are used in OOB authentication. Hence this API function is used only in SC pairing.

**Parameters:**

| | |
|---|---|
| *pRand* | 16 Bytes Random number to be used for generating OOB data. If NULL is passed, stack will generate 16 Bytes random number and then will generate OOB data. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Sufficient memory is not available to handle this request. |

**CYBLE_API_RESULT_T CyBle_GapSetDataLength (uint8 *bdHandle*, uint16 *connMaxTxOctets*, uint16 *connMaxTxTime*)**

This API function allows application to suggest maximum transmission packet size and maximum packet transmission time for current connection. Actual data length used by controller will be informed through 'CYBLE_EVT_GAP_DATA_LENGTH_CHANGE' event

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *connMaxTxOctets* | Preferred maximum number of payload octets that the local Controller should include in a single Link Layer Data Channel PDU. Range 0x001B-0x00FB (0x0000 - 0x001A and 0x00FC - 0xFFFF Reserved for future use) |
| *connMaxTxTime* | Preferred maximum number of microseconds that the local Controller should use to transmit a single Link Layer Data Channel PDU. Range 0x0148-0x0848 (0x0000 - 0x0147 and 0x0849 - 0xFFFF Reserved for future use) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified by bdHandle is not present |
| CYBLE_ERROR_INVALID_PARAMETER | Out of range value passed. |
| CYBLE_ERROR_INVALID_OPERATION | DLE feature not enabled |
| CYBLE_ERROR_LL_SAME_TRANSACTION_COLLISION | When there is already DLE procedure is pending |

**CYBLE_API_RESULT_T CyBle_GapSetRxDataLength (CYBLE_GAP_RX_DATA_LENGTH_T **RxDleParams*)**

This API function allows application to suggest the maximum number of payload octets that the local controller expects to receive and maximum time that local controller expects to take to receive a PDU on current connection.

Actual data length used by controller will be informed through 'CYBLE_EVT_GAP_DATA_LENGTH_CHANGE' event

**Parameters:**

| | |
|---|---|
| *RxDleParams* | Pointer to a structure of type 'CYBLE_GAP_RX_DATA_LENGTH_T'. It has three fields bdHandle field representing the peer device handle, connmaxRxOctets field representing preferred maximum number of payload octets that the local controller should expects to receive on current connection Range 0x001B-0x00FB (0x0000 - 0x001A and 0x00FC - 0xFFFF Reserved for future use) and connMaxRxTime field representing preferred maximum number of microseconds that the local Controller should use to receive a single Link Layer Data Channel PDU. Range 0x0148-0x0848 (0x0000 - 0x0147 and 0x0849 - 0xFFFF Reserved for future use) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified by bdHandle is not present |
| CYBLE_ERROR_INVALID_PARAMETER | If NULL passed |
| CYBLE_ERROR_INVALID_OPERATION | DLE feature not enabled |
| CYBLE_ERROR_LL_SAME_TRANSACTION_COLLISION | When there is already DLE procedure is pending |

**CYBLE_API_RESULT_T CyBle_GapSetSuggestedDataLength (uint16 *suggestedTxOctets*, uint16 *suggestedTxTime*)**

This API function allows the application to specify its preferred values for the Link Layer maximum Tx packet (Data Channel PDU) size (connInitialMaxTxOctets) and maximum Tx packet transmission time (connInitialMaxTxTime)to be used for new connections.

**Parameters:**

| | |
|---|---|
| *suggestedTxOctets* | The suggested value (connInitialMaxTxOctets) for the maximum transmitted number of payload octets (Link Layer Data Channel PDU) to be used for new connections. Range 0x001B-0x00FB (0x0000 - 0x001A and 0x00FC - 0xFFFF Reserved for future use) |
| *suggestedTxTime* | The suggested value (connInitialMaxTxTime)for the maximum packet (Link Layer Data Channel PDU) transmission time to be used for new connections. Application can use API function CyBle_GapConvertOctetToTime to get timeconnMaxTxTime corresponding to suggestedTxOctets. Range 0x0148-0x0848 (0x0000 - 0x0147 and 0x0849 - 0xFFFF Reserved for future use) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Out of range values. |
| CYBLE_ERROR_INVALID_OPERATION | DLE feature not enabled |

**CYBLE_API_RESULT_T CyBle_GapGetDataLength (CYBLE_GAP_DATA_LENGTH_T *readParam)**

This API function allows the application to read Link Layer maximum supported Tx/Rx packet (DataChannel PDU) octets / transmission time and maximum suggested Tx/Rx packet octets / transmission time.

**Parameters:**

| | |
|---|---|
| *readParam* | Pointer to structure of type 'CYBLE_GAP_DATA_LENGTH_T'. This is an output parameter which contain the maximum supported Tx and Rx octets & time and maximum suggested Tx octets & time. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Null pointer passed. |
| CYBLE_ERROR_INVALID_OPERATION | DLE feature not enabled |

**CYBLE_API_RESULT_T CyBle_GapConvertOctetToTime (CYBLE_GAP_PHY_TYPE_T *phy*, uint16 *octets*, uint16 **pTime*)**

This API function allows application to compute time from Octets. Time can be used to pass to BLE Stack while setting data length.

**Parameters:**

| | |
|---|---|
| *phy* | Physical layer to be considered while computing. Should be passed as CYBLE_GAP_PHY_1MBPS. Other values are Reserved. |
| *octets* | Payload octets. This is an input parameter. |
| *pTime* | Buffer where time in microseconds will be stored which is derived from octets and phy. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Null pointer passed. Invalid PHY Value passed. Invalid Octet Value is passed. (Valid Range 27 to 251) |

**CYBLE_API_RESULT_T CyBle_GapAddDeviceToResolvingList (const CYBLE_GAP_RESOLVING_DEVICE_INFO_T *rpaInfo)**

This API function is used to add a device to the resolving list in the controller for resolving Resolvable Private Address(RPA). This API function can be used to update local and/or peer IRKs for an existing Resolving List entry by passing the same peer address type and peer address in the argument.

**Parameters:**

| | |
|---|---|
| *rpaInfo* | Buffer which contains the information of peer address, peer address type, local and peer IRKs. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Any of the input parameter is NULL |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | When a Controller cannot add a device to the resolving list because the list is full. |
| CYBLE_ERROR_INVALID_OPERATION | Request is not permitted when address translation is enabled in the Controller and:<br>• Advertising is enabled<br>• Scanning is enabled<br>• Create connection command is outstanding. |

**CYBLE_API_RESULT_T CyBle_GapRemoveDeviceFromResolvingList (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr)**

This API function is used to remove one device from the list of address translations used to resolve Resolvable Private Addresses in the BLE Stack.

**Parameters:**

| | |
|---|---|
| *peerIdentityAddr* | Buffer which contains the information of peer bd address and address type |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Any of the input parameter is NULL |
| CYBLE_ERROR_INVALID_OPERATION | Request is not permitted when address translation is enabled in the Controller and:<br>• Advertising is enabled<br>• Scanning is enabled<br>• Create connection command is outstanding. |
| CYBLE_ERROR_NO_DEVICE_ENTITY | When a Controller cannot remove a device from the resolving list because it is not found. |

**CYBLE_API_RESULT_T CyBle_GapClearResolvingList (void )**

This API function is used to clear all devices from the list of address translations used to resolve Resolvable Private Addresses in the Controller.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_OPERATION | Request is not permitted when address translation is enabled in the Controller and:<br>• Advertising is enabled<br>• Scanning is enabled<br>• Create connection command is outstanding. |

**CYBLE_API_RESULT_T CyBle_GapReadPeerResolvableAddress (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr, uint8 *peerResolvableAddress)**

This API function is used to get the current peer Resolvable Private Address being used for the corresponding peer Public and Random (static) Identity Address. The peers resolvable address being used may change after the command is called.

**Parameters:**

| | |
|---|---|
| *peerIdentityAddr* | Buffer which contains the information of peer bd address and address type |
| *peerResolvableAddress* | Buffer to which peer resolvable private address will be stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Any of the input parameter is NULL |
| CYBLE_ERROR_NO_DEVICE_ENTITY | When a Controller cannot remove a device from the resolving list because it is not found. |

**CYBLE_API_RESULT_T CyBle_GapReadLocalResolvableAddress (const CYBLE_GAP_BD_ADDR_T *peerIdentityAddr, uint8 *localResolvableAddress)**

This API function is used to get the current local Resolvable Private Address being used for the corresponding peer Identity Address. The locals resolvable address being used may change after the command is called.

**Parameters:**

| | |
|---|---|
| *peerIdentityAddr* | Buffer which contains the information of peer bd address and address type |
| *localResolvableAddress* | Buffer to which local resolvable private address will be stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Any of the input parameter is NULL |
| CYBLE_ERROR_NO_DEVICE_ENTITY | When a Controller cannot remove a device from the resolving list because it is not found. |

**CYBLE_API_RESULT_T CyBle_GapSetResolvablePvtAddressTimeOut (uint16 *rpaTimeOut*)**

This API function is used to set the length of time the controller uses a Resolvable Private Address before a new resolvable private address is generated and starts being used. This timeout applies to all addresses generated by the BLE Stack.

**Parameters:**

| | |
|---|---|
| *rpaTimeOut* | RPA_Timeout measured in seconds. Range for N: 0x0001 0xA1B8 (1 sec approximately 11.5 hours) Default: N= 0x0384 (900 secs or 15 minutes) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Invalid timeout value |

**CYBLE_API_RESULT_T CyBle_GapReadResolvingList (CYBLE_GAP_RESOLVING_LIST_T **resolvingList*)**

This API function is used to read all the entries of address translation in the resolving list that is stored in BLE Stack.

**Parameters:**

| | |
|---|---|
| *resolvingList* | Buffer to store resolving list. Memory shall be allocated by the calling function. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Input parameter is NULL |

**CYBLE_API_RESULT_T CyBle_GapSetAddressResolutionEnable (uint8 *enableDisable*)**

This API function is used to enable resolution of Resolvable Private Addresses in the BLE Stack. This causes the BLE Stack to use the resolving list whenever the Controller receives a local or peer Resolvable Private Address.

**Parameters:**

| | |
|---|---|
| *enableDisable* | 0x00 - Address Resolution in controller disabled (default)<br>0x01 - Address Resolution in controller enabled<br>0x02 0xFF Reserved for Future Use |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | 'enableDisable' value is greater than 0x01 |

**CYBLE_API_RESULT_T CyBle_GapSetPrivacyMode (CYBLE_GAP_PRIVACY_MODE_INFO_T *privacyModeInfo)**

This API function is used to allow the Host to specify the privacy mode to be used for a given entry on the resolving list.

The effect of this setting is specified in [Vol 6] Part B, Section 4.7.

When an entry on the resolving list is removed, the mode associated with that entry shall also be removed.

**Parameters:**

| | |
|---|---|
| *privacyModeInfo* | Pointer to a structure of type CYBLE_GAP_PRIVACY_MODE_INFO_T |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | Any of the input parameter is NULL |
| CYBLE_ERROR_NO_DEVICE_ENTITY | When a Controller cannot find device entry in the resolving list |
| CYBLE_ERROR_INVALID_OPERATION | Request is not permitted when address translation is enabled in the Controller and:<br>• Advertising is enabled<br>• Scanning is enabled<br>• Create connection command is outstanding. |

**CYBLE_API_RESULT_T CyBle_GapGetBondedDevicesByRank (CYBLE_GAP_DEVICE_ADDR_LIST_T *bondedDevList)**

This function returns the count and Bluetooth device address along with bd handles of the devices in the bonded device list in the order of Rank*. This is a blocking function. No event is generated on calling this function.

Rank: Newest device bonded will be at 0 index.

Application invoking this function should allocate sufficient memory for the structure CYBLE_GAP_DEVICE_ADDR_LIST_T, where the complete list of bonded devices along with count can be written. Maximum devices bonded are specified by CYBLE_GAP_MAX_BONDED_DEVICE, which is a pre processing time parameter for the BLE Stack. Hence, the bonded device count will be less than or equal to CYBLE_GAP_MAX_BONDED_DEVICE.

Refer Bluetooth 4.1 core specification, Volume 3, Part C, section 9.4.4 for details on bonded devices.

**Parameters:**

| | |
|---|---|
| *bondedDevList* | Buffer to which list of bonded device list will be stored of type CYBLE_GAP_DEVICE_ADDR_LIST_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

### CYBLE_API_RESULT_T CyBle_GapSetLeEventMask (uint8 *hciLeEventMask)

The CyBle_GapSetLeEventMask API function is equivalent of LE_Set_Event_Mask HCI command and is used to control which LE events are generated by the HCI for the Host. Host will process these events and will send appropriate events to application. If the bit in the hciLeEventMask is set to a one, then the event associated with that bit will be enabled. The Host has to deal with each event that is generated by an LE Controller. The event mask allows the application to control which events will be generated for host.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *hciLeEventMask* | Pointer to the LE Mask. As of today stack expects 2 bytes length for this buffer (hciLeEventMask) Refer Core Spec, Vol2, Part E, 7.8.1 for further information. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

### CYBLE_API_RESULT_T CyBle_GapSetIdAddress (const CYBLE_GAP_BD_ADDR_T *bdAddr)

This function sets the Bluetooth identity address into BLE Stack. Calling to this API function will only change the identity address of the device. If public address or static random address is changed by user, this API function needs to be called to set the appropriate address as identity address.

This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Pointer to the CYBLE_GAP_BD_ADDR_T structure variable. It has two fields where,<br>• bdAddr.addr: Bluetooth Device address buffer that is populated with the device address data.<br>• bdAddr.type: Caller function should fill the "address type" to set appropriate address.<br>Caller function should use bdAddr.type = 0x00 to set the "Public Device Address" as identity address.<br>Caller function use bdAddr.type = 0x01 to set the "Static Random Device Address" as identity address. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter. |

### CYBLE_API_RESULT_T CyBle_GapGenerateAndSetIrk (uint8 *keysFlag*, uint8 *irk)

This function generates and sets local Identity resolving key into BLE Stack that is to be exchanged with peer device during key exchange stage of authentication procedure. This API function only updates IRK and does not change any other keys. This is a blocking function. No event is generated on calling this function. This API function does not generate identity address (keyInfo->idAddrInfo)

**Parameters:**

| | |
|---|---|
| *keysFlag* | (Input parameter) This parameter indicates which keys get exchanged with peer device. The following is the bit field mapping for the keys.<br>• Bit 0: Local Encryption information<br>• Bit 1: Local Identity information<br>• Bit 2: Local Signature Key<br>• Bit 3: Reserved<br>• Bit 4: Remote Encryption information<br>• Bit 5: Remote Identity information<br>• Bit 6: Remote Signature Key<br>• Bit 7: Reserved |
| *irk* | (output parameter) Pointer to 16 Bytes buffer where IRK is stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'keyInfo'. |

**CYBLE_API_RESULT_T CyBle_GapFixAuthPassKey (uint8 *isFixed*, uint32 *fixedPassKey*)**

Sets or clears fixed passkey to be used by SMP procedure. This is a blocking function. No event is generated on calling this function.

Note1: The fixed passkey will only work if we are the device displaying the passkey and peer has to enter the passkey. This will not work for numeric comparison(secure connections) method.

Note2: The fixed passkey is not persistent across power cycle.

Note3: This API function should not be called during ongoing SMP procedure. Recommendation is to call this API function on Stack Init completion.

**Parameters:**

| | |
|---|---|
| *isFixed* | isFixed should be true(non zero) and fixedPassKey should be valid passkey (<=999999) to set the fixed passkey.<br>isFixed should be false(0) to ask SMP to generate random passkey instead of using the fixed passkey. This is only required if previously the passkey was fixed using this API function. |
| *fixedPassKey* | Valid fixed passkey (<=999999) to be used by SMP. This is only used if isFixed is set to true else ignored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | If fixedPassKey is not a valid passkey. |

**CYBLE_API_RESULT_T CyBle_GappSetNumOfAdvPkts (CYBLE_GAPP_DISC_MODE_INFO_T **advInfo*, uint16 *NumOfAdvPkts*)**

Sets number of advertisement packets to be sent over the air and starts Advertisement.

Gap Peripheral receives CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event after specified number of adv packets transmitted successfully.

Note1: Advertisement Timeout will be ignored. Note2: Ongoing Advertisement should be stopped by the application before calling this API function.

**Parameters:**

| | |
|---|---|
| *advInfo* | Structure of type CYBLE_GAPP_DISC_MODE_INFO_T, which contains the advertisement parameters |
| *NumOfAdvPkts* | (Input parameter) Total number of packets to transmitted over the air. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | If Zero passed |

# GAP Central Functions

## Description

APIs unique to designs configured as a GAP Central role.

A letter 'c' is appended to the API name: CyBle_Gapc

## Functions

- CYBLE_API_RESULT_T CyBle_GapcStartScan (uint8 scanningIntervalType)
- void CyBle_GapcStopScan (void)
- CYBLE_API_RESULT_T CyBle_GapcConnectDevice (const CYBLE_GAP_BD_ADDR_T *address)
- CYBLE_API_RESULT_T CyBle_GapcCancelDeviceConnection (void)
- CYBLE_API_RESULT_T CyBle_GapcStartDiscovery (CYBLE_GAPC_DISC_INFO_T *scanInfo)
- void CyBle_GapcStopDiscovery (void)
- CYBLE_API_RESULT_T CyBle_GapcInitConnection (CYBLE_GAPC_CONN_PARAM_T *connParam)
- CYBLE_API_RESULT_T CyBle_GapcCancelConnection (void)
- CYBLE_API_RESULT_T CyBle_GapcResolveDevice (const uint8 *bdAddr, const uint8 *irk)
- CYBLE_API_RESULT_T CyBle_GapcConnectionParamUpdateRequest (uint8 bdHandle, CYBLE_GAP_CONN_UPDATE_PARAM_T *connParam)
- CYBLE_API_RESULT_T CyBle_GapcSetHostChannelClassification (uint8 *channelMap)
- CYBLE_API_RESULT_T CyBle_GapcSetRemoteAddr (uint8 bdHandle, CYBLE_GAP_BD_ADDR_T remoteAddr)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GapcStartScan (uint8 *scanningIntervalType*)**

This function is used for discovering GAP peripheral devices that are available for connection. It performs the scanning routine using the parameters entered in the component's customizer.

As soon as the discovery operation starts, CYBLE_EVT_GAPC_SCAN_START_STOP event is generated. The CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT event is generated when a GAP peripheral device is located. There are three discovery procedures can be selected in the customizer's GUI:

- Observation procedure: A device performing the observer role receives only advertisement data from devices irrespective of their discoverable mode settings. Advertisement data received is provided by the event, CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT. This procedure requires the scanType sub parameter to be passive scanning.
- Limited Discovery procedure: A device performing the limited discovery procedure receives advertisement data and scan# response data from devices in the limited discoverable mode only. Received data is provided by the event, CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT. This procedure requires the scanType sub-parameter to be active scanning.
- General Discovery procedure: A device performing the general discovery procedure receives the advertisement data and scan response data from devices in both limited discoverable mode and the general discoverable mode. Received data is provided by the event, CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT. This procedure requires the scanType sub-parameter to be active scanning.

Every Advertisement / Scan response packet received results in a new event, CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT. If 'scanTo' sub-parameter is a non-zero value, then upon commencement of discovery procedure and elapsed time = 'scanTo', CYBLE_EVT_TIMEOUT event is generated with the event parameter indicating CYBLE_GAP_SCAN_TO. Possible generated events are:

- CYBLE_EVT_GAPC_SCAN_START_STOP: If a device started or stopped scanning. Use CyBle_GetState() to determine the state. Sequential scanning could be started when CYBLE_STATE_DISCONNECTED state is returned.
- CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT
- CYBLE_EVT_TIMEOUT (CYBLE_GAP_SCAN_TO)

**Parameters:**

| | |
|---|---|
| *scanningIntervalType* | Fast or slow scanning interval with timings entered in Scan settings section of the customizer.<br>• CYBLE_SCANNING_FAST 0x00u<br>• CYBLE_SCANNING_SLOW 0x01u<br>• CYBLE_SCANNING_CUSTOM 0x02u |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_STACK_INTERNAL | An error occurred in the BLE stack. |
| CYBLE_ERROR_INVALID_PARAMETER | On passing an invalid parameter. |

**void CyBle_GapcStopScan (void )**

This function used to stop the discovery of devices. On stopping discovery operation, CYBLE_EVT_GAPC_SCAN_START_STOP event is generated. Application layer needs to keep track of the function call made before receiving this event to associate this event with either the start or stop discovery function.

Possible events generated are:

- CYBLE_EVT_GAPC_SCAN_START_STOP

**Returns:**

None

**CYBLE_API_RESULT_T CyBle_GapcConnectDevice (const CYBLE_GAP_BD_ADDR_T *address)**

This function is used to send a connection request to the remote device with the connection parameters set in the component customizer. This function needs to be called only once after the target device is discovered by CyBle_GapcStartScan() and further scanning has stopped. Scanning is successfully stopped on invoking CyBle_GapcStopScan() and then receiving the event CYBLE_EVT_GAPC_SCAN_START_STOP with sub-parameter 'success' = 0x01u.

On successful connection, the following events are generated at the GAP Central device (as well as the GAP Peripheral device), in the following order.

- CYBLE_EVT_GATT_CONNECT_IND
- CYBLE_EVT_GAP_DEVICE_CONNECTED - If the device connects to a GAP Central and Link Layer Privacy is disabled in component customizer.
- CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE - If the device connects to a GAP Central and Link Layer Privacy is enabled in component customizer.
- CYBLE_EVT_GAP_DEVICE_CONNECTED

A procedure is considered to have timed out if a connection response packet is not received within time set by cyBle_connectingTimeout global variable (30 seconds by default). CYBLE_EVT_TIMEOUT event with CYBLE_GENERIC_TO parameter will indicate about connection procedure timeout. Connection will automatically be canceled and state will be changed to CYBLE_STATE_DISCONNECTED.

**Parameters:**

| | |
|---|---|
| *address* | The device address of the remote device to connect to. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_STACK_INTERNAL | On error occurred in the BLE stack. |
| CYBLE_ERROR_INVALID_PARAMETER | On passing an invalid parameter. |
| CYBLE_ERROR_INVALID_STATE | On calling this function not in Disconnected state. |

**CYBLE_API_RESULT_T CyBle_GapcCancelDeviceConnection (void )**

This function cancels a previously initiated connection with the remote device. It is a blocking function. No event is generated on calling this function. If the devices are already connected then this function should not be used. If you intend to disconnect from an existing connection, the function CyBle_GapDisconnect() should be used.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_STACK_INTERNAL | An error occurred in the BLE stack. |
| CYBLE_ERROR_INVALID_STATE | On calling this function not in Connecting state. |

**CYBLE_API_RESULT_T CyBle_GapcStartDiscovery (CYBLE_GAPC_DISC_INFO_T *scanInfo)**

This function starts the discovery of devices which are advertising. This is a non-blocking function. As soon as the discovery operation starts, CYBLE_EVT_GAPC_SCAN_START_STOP event is generated.

Every Advertisement / Scan response packet received results in a new event, CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT. If 'scanInfo->scanTo' is a non-zero value, upon commencement of discovery procedure and elapsed time = 'scanInfo->scanTo', CYBLE_EVT_TIMEOUT event is generated with the event parameter indicating CYBLE_GAP_SCAN_TO.

If 'scanInfo->scanTo' is equal to zero, the scanning operation is performed until the CyBle_GapcStopDiscovery() function is invoked.

There are three discovery procedures that can be specified as a parameter to this function.

**Observation procedure**

A device performing the observer role receives only advertisement data from devices irrespective of their discoverable mode settings. Advertisement data received is provided by the event,

CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT

'scanInfo->scanType' should be set as passive scanning (0x00).

**Limited Discovery procedure**

A device performing the limited discovery procedure receives advertisement data and scan response data from devices in the limited discoverable mode only. Received data is provided by the event,

CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT

'scanInfo->scanType' should be set as active scanning (0x01).

**General Discovery procedure**

A device performing the general discovery procedure receives the advertisement data and scan response data from devices in both limited discoverable mode and the general discoverable mode. Received data is provided by the event,

CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT

'scanInfo->scanType' should be set as active scanning (0x01).

**Parameters:**

| | |
|---|---|
| *scanInfo* | Pointer to a variable of type CYBLE_GAPC_DISC_INFO_T |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'scanInfo' or if any element within ' scanInfo' has an invalid value. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

**void CyBle_GapcStopDiscovery (void )**

This function stops the discovery of devices. This is a non-blocking function. On stopping discovery operation, CYBLE_EVT_GAPC_SCAN_START_STOP event is generated. Application layer needs to keep track of the function call made before receiving this event to associate this event with either the start or stop discovery function.

**CYBLE_API_RESULT_T CyBle_GapcInitConnection (CYBLE_GAPC_CONN_PARAM_T *connParam)**

This function instructs BLE Stack to initiate connection request to the remote device with required connection parameters. Connection request from application is acknowledged by BLE Controller as

'CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE' or 'CYBLE_EVT_GAP_DEVICE_CONNECTED' depend on Link Layer Privacy is enabled or not in component customizer. That means, request is correct, permitted and all parameters as part of the request are correct. If the parameter validation or request is not permitted, then BLE controller throws 'CYBLE_EVT_HCI_STATUS' event with error code instead of CYBLE_EVT_GAP_DEVICE_CONNECTEDCYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE. For positive condition, controller can issue connect request to peer. Once connection is done, no more event is required but if fails to establish connection, 'CYBLE_EVT_GAP_DEVICE_DISCONNECTED' is passed to application.

This is a non-blocking function. This function needs to be called after successfully stopping scanning. Scanning is successfully stopped on invoking the CyBle_GapcStopDiscovery() function and receiving the event CYBLE_EVT_GAPC_SCAN_START_STOP with the event data of '0x01', indicating success.

For details related to connection modes and procedures, refer to Bluetooth 4.1 Core Specification, Volume 3, Part C, Section 9.3.

**Parameters:**

| | |
|---|---|
| *connParam* | Structure of type 'CYBLE_GAPC_CONN_PARAM_T' which contains the connection parameters.<br>**Note** Any parameter of structure type CYBLE_GAPC_CONN_PARAM_T, if not required by a specific Bluetooth Low Energy profile, may be ignored. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'connParam' or if any element within 'connParam' has an invalid value. |
| CYBLE_ERROR_INVALID_OPERATION | Device already connected. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

### CYBLE_API_RESULT_T CyBle_GapcCancelConnection (void )

Description: This function cancels a previously initiated connection with the peer device. This is a blocking function. No event is generated on calling this function.

If the devices are already connected, then this function should not be used. To disconnect from an existing connection, use the function CyBle_GapDisconnect().

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_OPERATION | Device already connected. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

**CYBLE_API_RESULT_T CyBle_GapcResolveDevice (const uint8 **bdAddr*, const uint8 **irk*)**

This function enables the application to start resolution procedure for a device that is connected using resolvable private address. This is a blocking function. Application should use this function when in GAP Central mode.

Refer to Bluetooth 4.1 Core specification, Volume 3, Part C, section 10.8.2.3 Resolvable Private Address Resolution Procedure to understand the usage of Private addresses.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Pointer to peer Bluetooth device address of length 6 bytes, not NULL terminated. |
| *irk* | Pointer to 128-bit IRK to be used for resolving the peer's private resolvable address. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'bdAddr' or 'irk'. |
| CYBLE_ERROR_INVALID_OPERATION | No device to be resolved. The specified device handle does not map to any device entry in the BLE Stack. |

**CYBLE_API_RESULT_T CyBle_GapcConnectionParamUpdateRequest (uint8 *bdHandle*, CYBLE_GAP_CONN_UPDATE_PARAM_T **connParam*)**

This function sends the connection parameter update command to local controller. This function can only be used from device connected in GAP Central role. Note: Connection parameter update procedure, defined as part of Bluetooth spec 4.1, is not supported. This function will allow GAP Central application to update connection parameter for local controller and local controller will follow the procedure as defined in Bluetooth Core specification 4.0.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *connParam* | Pointer to a structure of type CYBLE_GAP_CONN_UPDATE_PARAM_T containing connection parameter updates |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation 'connParam' is NULL |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |

**CYBLE_API_RESULT_T CyBle_GapcSetHostChannelClassification (uint8 **channelMap*)**

This function sets channel classification for data channels. This classification persists until it is overwritten by a subsequent call to this function or the controller is reset. If this command is used, updates should be sent within 10 seconds of the BLE Host knowing that the channel classification has changed. The interval between two

successive commands sent will be at least one second. This command will only be used when the local device supports the Master role.

For details, refer to Bluetooth core specification 4.1, Volume 2, part E, section 7.8.19.

This is a non blocking function. Application should look for 'CYBLE_EVT_HCI_STATUS' for any error condition.

**Parameters:**

| | |
|---|---|
| *channelMap* | This parameter contains five octet byte stream (Least Significant Byte having the bit fields 0 to 7, most significant byte having the bit fields 32 to 36). The nth such field (in the range 0 to 36) contains the value for the link layer channel index n. Allowed values and their interpretation are,<br>• Channel 'n' is disabled = 0x00u<br>• Channel 'n' is enabled = 0x01u |

The most significant bits (37 to 39) are reserved and will be set to 0. At least one channel will be marked as unknown. For example- expected pattern = XX XX XX XX 1F not expected = XX XX XX XX 10, XX XX XX XX 2f MSB 3 bits should be not set. (1f is most significant bytes in this case)

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'channelMap'. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

**CYBLE_API_RESULT_T CyBle_GapcSetRemoteAddr (uint8 *bdHandle*, CYBLE_GAP_BD_ADDR_T *remoteAddr*)**

This function allows application to set the new address of remote device identified by bdHandle. This API function should be used when:

1. If peer device is previously bonded with public address and changes its bd address to resolvable private address. Application should resolve the device by calling 'CyBle_GapcResolveDevice()' api and set the new address if successfully resolved.
2. If device is previously bonded with random, application should call this api to set the new address(public/random).

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *remoteAddr* | Peer device address, of type CYBLE_GAP_BD_ADDR_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On invalid bdHandle |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |

# GAP Peripheral Functions

## Description

APIs unique to designs configured as a GAP Peripheral role.

A letter 'p' is appended to the API name: CyBle_Gapp

## Functions

- CYBLE_API_RESULT_T CyBle_GappStartAdvertisement (uint8 advertisingIntervalType)
- void CyBle_GappStopAdvertisement (void)
- void CyBle_ChangeAdDeviceAddress (const CYBLE_GAP_BD_ADDR_T *bdAddr, uint8 dest)
- CYBLE_API_RESULT_T CyBle_GappEnterDiscoveryMode (CYBLE_GAPP_DISC_MODE_INFO_T *advInfo)
- void CyBle_GappExitDiscoveryMode (void)
- CYBLE_API_RESULT_T CyBle_GappAuthReqReply (uint8 bdHandle, CYBLE_GAP_AUTH_INFO_T *authInfo)
- CYBLE_API_RESULT_T CyBle_GapUpdateAdvData (CYBLE_GAPP_DISC_DATA_T *advDiscData, CYBLE_GAPP_SCAN_RSP_DATA_T *advScanRespData)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GappStartAdvertisement (uint8 *advertisingIntervalType*)**

This function is used to start the advertisement using the advertisement data set in the component customizer's GUI. After invoking this function, the device will be available for connection by the devices configured for GAP central role. It is only included if the device is configured for GAP Peripheral or GAP Peripheral + Central role.

On start of advertisement, GAP Peripheral receives the CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event. The following events are possible on invoking this function:

- CYBLE_EVT_GAP_DEVICE_CONNECTED - If the device connects to a GAP Central and Link Layer Privacy is disabled in component customizer.
- CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE - If the device connects to a GAP Central and Link Layer Privacy is enabled in component customizer.
- CYBLE_EVT_TIMEOUT: If no device in GAP Central mode connects to this device within the specified timeout limit. Stack automatically initiate stop advertising when Slow advertising was initiated, or starts Slow advertising after Fast advertising timeout occur.
- CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP: If device started or stopped advertising. Use CyBle_GetState() to determine the state. Sequential advertising could be started when CYBLE_STATE_DISCONNECTED state is returned.

**Parameters:**

| | |
|---|---|
| *advertisingIntervalType* | Fast or slow advertising interval with timings entered in Advertising settings section of the customizer.<br>• CYBLE_ADVERTISING_FAST 0x00u<br>• CYBLE_ADVERTISING_SLOW 0x01u<br>• CYBLE_ADVERTISING_CUSTOM 0x02u |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On passing an invalid parameter. |
| CYBLE_ERROR_INVALID_STATE | On calling this function not in Disconnected state. |

**void CyBle_GappStopAdvertisement (void )**

This function can be used to exit from discovery mode. After the execution of this function, there will no longer be any advertisements. On stopping advertising, GAP Peripheral receives CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event. It is expected that the application layer tracks the function call performed before occurrence of this event as this event can occur on making a call to Cy_BleGappStartAdvertisement(), CyBle_GappEnterDiscoveryMode(), or CyBle_GappStartAdvertisement() functions as well.

The following event occurs on invoking this function:

- CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP

**Returns:**

None

**void CyBle_ChangeAdDeviceAddress (const CYBLE_GAP_BD_ADDR_T **bdAddr*, uint8 *dest*)**

This function is used to set the Bluetooth device address into the advertisement or scan response data structure.

**Parameters:**

| | |
|---|---|
| *bdAddr* | Bluetooth Device address. The variable is of type CYBLE_GAP_BD_ADDR_T |
| *dest* | 0 - selects advertisement structure, not zero value selects scan response structure. |

**Returns:**

None

**CYBLE_API_RESULT_T CyBle_GappEnterDiscoveryMode (CYBLE_GAPP_DISC_MODE_INFO_T **advInfo*)**

This function sets the device into discoverable mode. In the discoverable mode, based on the parameters passed to this function, the BLE Device starts advertisement and can respond to scan requests. This is a non-blocking function. It is to be used by the device in 'GAP Peripheral' mode of operation to set parameters essential for starting advertisement procedure.

On start of advertisement, the GAP Peripheral receives CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event. The following events can occur on invoking this function.

- CYBLE_EVT_GAP_DEVICE_CONNECTED - If the device connects to a GAP Central and Link Layer Privacy is disabled in component customizer. CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE - If the device connects to a GAP Central and Link Layer Privacy is enabled in component customizer.
- CYBLE_EVT_TIMEOUT - If no device in 'GAP Central' mode connects to this device within the specified timeout limit. This event can occur if 'advInfo ->discMode' is equal to CYBLE_GAPP_LTD_DISC_MODE or CYBLE_GAPP_GEN_DISC_MODE. 'advInfo-> advTo' specifies the timeout duration. Set the 'advInfo-> advTo' to 0 when 'advInfo -> discMode' is set to CYBLE_GAPP_GEN_DISC_MODE so that the timeout event does not occur and the advertisement continues until the CyBle_GappExitDiscoveryMode() function is invoked.

**Parameters:**

| | |
|---|---|
| *advInfo* | Structure of type CYBLE_GAPP_DISC_MODE_INFO_T, which contains the advertisement parameters |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying null pointer for 'advInfo' or if any of the elements of this structure have invalid values. |

**void CyBle_GappExitDiscoveryMode (void )**

This function is used to exit from discoverable mode. This is a non-blocking function. After the execution of this function, the device stops advertising.

On stopping advertising, GAP Peripheral receives CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP event. It is expected that the application layer keeps track of the function call performed before occurrence of this event, as this event can occur on making a call to the CyBle_GappEnterDiscoveryMode () function as well.

**CYBLE_API_RESULT_T CyBle_GappAuthReqReply (uint8 *bdHandle*, CYBLE_GAP_AUTH_INFO_T **authInfo*)**

This function is used to pass security information for authentication in reply to an authentication request from the master device. It should be invoked on receiving CYBLE_EVT_GAP_AUTH_REQ event. Events shown in the following table may be received by the application based on the authentication result.

| Event Parameter | Description |
|---|---|
| CYBLE_EVT_TIMEOUT | With error code CYBLE_GAP_PAIRING_PROCESS_TO on invoking CyBle_GappAuthReqReply() or CyBle_GapAuthReq() if there is no response from the peer device |
| CYBLE_EVT_GAP_AUTH_COMPLETE | Pointer to structure of type 'CYBLE_GAP_AUTH_INFO_T' is returned as parameter to both the peer devices on successful authentication. |
| CYBLE_EVT_GAP_AUTH_FAILED | Received by both GAP Central and Peripheral devices (peers) on authentication failure. Data is of type CYBLE_GAP_AUTH_FAILED_REASON_T. |
| CYBLE_EVT_GAP_SMP_NEGOTIATED_AUTH_INFO | With negotiated pairing parameters on invoking CyBle_GappAuthReqReply() from function call context. |

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *authInfo* | Pointer to a variable containing security information of the device of type CYBLE_GAP_AUTH_INFO_T. |

NOTE: If the bonding flag in authInfo is set to CYBLE_GAP_BONDING_NONE then, SMP keys will not be distributed even if application has generated and set the keys explicitly.

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying null pointer for 'advInfo' or if any of the element of this structure has an invalid value. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_NO_DEVICE_ENTITY | Device identified using 'bdHandle' does not exist. |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | On bonded device is full and application tries to initiate pairing with bonding enable. |

**CYBLE_API_RESULT_T CyBle_GapUpdateAdvData (CYBLE_GAPP_DISC_DATA_T *advDiscData, CYBLE_GAPP_SCAN_RSP_DATA_T *advScanRespData)**

This function allows setting the ADV data and SCAN response data while advertising is ongoing. Application shall preserve Bluetooth Spec 4.1 mandated AD flags fields corresponding to the type of discovery mode the device is in and only change the rest of the data. This API function must be called when API function CyBle_GetBleSsState() returns CYBLE_BLESS_STATE_EVENT_CLOSE state. If API returns is called in any of the BLESS Low Power Modes, it will force exit BLESS from Low Power Mode state to update ADV Data.

**Parameters:**

| | |
|---|---|
| *advDiscData* | Pointer to a structure of CYBLE_GAPP_DISC_DATA_T. It has two fields advData field representing the data and advDataLen indicating the length of present data. Application can pass length as 0 if the ADV data doesn't need to be changed. |
| *advScanRespData* | Pointer to a structure of type CYBLE_GAPP_SCAN_RSP_DATA_T. It has two fields scanRspData field representing the data and scanRspDataLen indicating the length of present data. Application can pass length as 0 if the SCAN RESP data doesn't need to be changed. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On NULL pointer, Data length in input parameter exceeds 31 bytes. |
| CYBLE_ERROR_INVALID_OPERATION | ADV Event is not closed, BLESS is active or ADV is not enabled. |

# GAP Definitions and Data Structures

## Description

Contains the GAP specific definitions and data structures used in the GAP APIs.

## Data Structures

- struct CYBLE_GAPC_T
- struct CYBLE_GAPS_T
- struct CYBLE_GAP_BD_ADDR_T
- struct CYBLE_GAP_AUTH_INFO_T
- struct CYBLE_GAP_BONDED_DEV_ADDR_LIST_T
- struct CYBLE_GAP_SMP_KEY_DIST_T
- struct CYBLE_GAP_SMP_LOCAL_P256_KEYS
- struct CYBLE_GAPP_DISC_PARAM_T
- struct CYBLE_GAPP_DISC_DATA_T
- struct CYBLE_GAPP_SCAN_RSP_DATA_T
- struct CYBLE_GAPP_DISC_MODE_INFO_T
- struct CYBLE_GAPC_DISC_INFO_T
- struct CYBLE_GAPC_CONN_PARAM_T
- struct CYBLE_GAPC_ADV_REPORT_T
- struct CYBLE_GAP_PASSKEY_DISP_INFO_T
- struct CYBLE_GAP_CONN_UPDATE_PARAM_T
- struct CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T
- struct CYBLE_GAP_OOB_DATA_T
- struct CYBLE_GAP_DATA_LENGTH_T
- struct CYBLE_GAP_CONN_DATA_LENGTH_T
- struct CYBLE_GAP_RX_DATA_LENGTH_T
- struct CYBLE_GAP_RESOLVING_DEVICE_INFO_T
- struct CYBLE_GAP_RESOLVING_LIST_T
- struct CYBLE_GAPC_DIRECT_ADV_REPORT_T
- struct CYBLE_GAP_ENHANCE_CONN_COMPLETE_T
- struct CYBLE_GAP_DEVICE_LIST_T
- struct CYBLE_GAP_DEVICE_ADDR_LIST_T
- struct CYBLE_GAP_PRIVACY_MODE_INFO_T

## Enumerations

- enum CYBLE_GAP_ADV_ASSIGN_NUMBERS
- enum CYBLE_GAPP_ADV_T
- enum CYBLE_GAPC_ADV_EVENT_T
- enum CYBLE_GAP_SEC_LEVEL_T
- enum CYBLE_GAP_IOCAP_T
- enum CYBLE_GAP_AUTH_FAILED_REASON_T
- enum CYBLE_GAP_ADDR_TYPE_T
- enum CYBLE_GAP_KEYPRESS_NOTIFY_TYPE
- enum CYBLE_GAP_ADV_ADDR_TYPE_T
- enum CYBLE_GAP_PHY_TYPE_T

## Data Structure Documentation

### struct CYBLE_GAPC_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T deviceNameCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T appearanceCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T periphPrivacyCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T reconnAddrCharHandle

- CYBLE_GATT_DB_ATTR_HANDLE_T prefConnParamCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T centralAddrResolutionCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T rpaOnlyCharHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::deviceNameCharHandle**
Discovered handle of the GAP Service Device Name Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::appearanceCharHandle**
Discovered handle of the GAP Service Appearance Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::periphPrivacyCharHandle**
Discovered handle of the GAP Service Peripheral Privacy Flag Parameters Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::reconnAddrCharHandle**
Discovered handle of the GAP Service Reconnection Address Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::prefConnParamCharHandle**
Discovered handle of the GAP Service Peripheral Preferred Connection Parameters Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::centralAddrResolutionCharHandle**
Discovered handle of the GAP Service Central Address Resolution Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPC_T::rpaOnlyCharHandle**
Discovered handle of the GAP Service Resolvable Private Address Only Characteristic

**struct CYBLE_GAPS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T gapServiceCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T deviceNameCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T appearanceCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T prefConnParamCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T centralAddrResolutionCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T rpaOnlyCharHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::gapServiceCharHandle**
Handle of the GAP Service Device Name Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::deviceNameCharHandle**
Handle of the GAP Service Device Name Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::appearanceCharHandle**
Handle of the GAP Service Appearance Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::prefConnParamCharHandle**
Handle of the GAP Service Peripheral Preferred Connection Parameters Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::centralAddrResolutionCharHandle**
Handle of the GAPS Central Address Resolution characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GAPS_T::rpaOnlyCharHandle**
Handle of the GAPS Resolvable Private Address Only characteristic

**struct CYBLE_GAP_BD_ADDR_T**

**Data Fields**

- uint8 bdAddr [(0x06u)]
- uint8 type

**Field Documentation**

**uint8 CYBLE_GAP_BD_ADDR_T::bdAddr[(0x06u)]**
Bluetooth device address

**uint8 CYBLE_GAP_BD_ADDR_T::type**
public = 0, Random = 1

**struct CYBLE_GAP_AUTH_INFO_T**

**Data Fields**

- uint8 security
- uint8 bonding
- uint8 ekeySize
- CYBLE_GAP_AUTH_FAILED_REASON_T authErr
- uint8 pairingProperties

**Field Documentation**

**uint8 CYBLE_GAP_AUTH_INFO_T::security**
Security Mode setting will be as follows: (CYBLE_GAP_SEC_MODE_1 | CYBLE_GAP_SEC_LEVEL_1) (CYBLE_GAP_SEC_MODE_1 | CYBLE_GAP_SEC_LEVEL_2) (CYBLE_GAP_SEC_MODE_1 | CYBLE_GAP_SEC_LEVEL_3) (CYBLE_GAP_SEC_MODE_1 | CYBLE_GAP_SEC_LEVEL_4) (CYBLE_GAP_SEC_MODE_2 | CYBLE_GAP_SEC_LEVEL_2) (CYBLE_GAP_SEC_MODE_2 | CYBLE_GAP_SEC_LEVEL_3)

**uint8 CYBLE_GAP_AUTH_INFO_T::bonding**
Bonding type setting: CYBLE_GAP_BONDING_NONE CYBLE_GAP_BONDING

**uint8 CYBLE_GAP_AUTH_INFO_T::ekeySize**
Encryption Key Size (octets) Minimum = 7 maximum = 16 For slave initiated security request, this parameter needs to be ignored.

**CYBLE_GAP_AUTH_FAILED_REASON_T CYBLE_GAP_AUTH_INFO_T::authErr**
Parameter to say it authentication is accepted or rejected with reason. accepted = CYBLE_GAP_AUTH_ERROR_NONE or error code CYBLE_GAP_AUTH_FAILED_REASON_T.

**uint8 CYBLE_GAP_AUTH_INFO_T::pairingProperties**
Bit 0: MITM (Applicable only if Secure connections) Use SMP_SC_PAIR_PROP_MITM_MASK Bit 1: Key press (sets Key press bit in authentication requirements flags of pairing request/response. Applicable only for secure connections) Use SMP_SC_PAIR_PROP_KP_MASK Bit [2-7]: RFU

**struct CYBLE_GAP_BONDED_DEV_ADDR_LIST_T**

**Data Fields**

- uint8 count
- CYBLE_GAP_BD_ADDR_T bdAddrList [0x04u]

**Field Documentation**

**uint8 CYBLE_GAP_BONDED_DEV_ADDR_LIST_T::count**
Number of bonded devices

**CYBLE_GAP_BD_ADDR_T CYBLE_GAP_BONDED_DEV_ADDR_LIST_T::bdAddrList[0x04u]**
Pointer to list of Bluetooth device addresses of bonded devices, of type 'CYBLE_GAP_BD_ADDR_T'. 'CYBLE_GAP_MAX_BONDED_DEVICE' is a '#define' to be defined during build-time.

**struct CYBLE_GAP_SMP_KEY_DIST_T**

**Data Fields**

- uint8 ltkInfo [0x10u]

- uint8 midInfo [0x0Au]
- uint8 irkInfo [0x10u]
- uint8 idAddrInfo [0x07u]
- uint8 csrkInfo [0x10u]

**Field Documentation**

**uint8 CYBLE_GAP_SMP_KEY_DIST_T::ltkInfo[0x10u]**
Long Term Key

**uint8 CYBLE_GAP_SMP_KEY_DIST_T::midInfo[0x0Au]**
Encrypted Diversifier and Random Number

**uint8 CYBLE_GAP_SMP_KEY_DIST_T::irkInfo[0x10u]**
Identity Resolving Key

**uint8 CYBLE_GAP_SMP_KEY_DIST_T::idAddrInfo[0x07u]**
Public device/Static Random address type idAddrInfo[0] - Address Type idAddrInfo[1] to idAddrInfo[6] - Address

**uint8 CYBLE_GAP_SMP_KEY_DIST_T::csrkInfo[0x10u]**
Connection Signature Resolving Key

### struct CYBLE_GAP_SMP_LOCAL_P256_KEYS

**Data Fields**

- uint8 publicKey [0x40u]
- uint8 privateKey [0x20u]

**Field Documentation**

**uint8 CYBLE_GAP_SMP_LOCAL_P256_KEYS::publicKey[0x40u]**
P-256 public key

**uint8 CYBLE_GAP_SMP_LOCAL_P256_KEYS::privateKey[0x20u]**
P-256 private key

### struct CYBLE_GAPP_DISC_PARAM_T

**Data Fields**

- uint16 advIntvMin
- uint16 advIntvMax
- CYBLE_GAPP_ADV_T advType
- uint8 ownAddrType
- uint8 directAddrType
- uint8 directAddr [(0x06u)]
- uint8 advChannelMap
- uint8 advFilterPolicy

**Field Documentation**

**uint16 CYBLE_GAPP_DISC_PARAM_T::advIntvMin**
Minimum advertising interval for undirected and low duty cycle directed advertising.

- Time Range: 20 ms to 10.24 sec

**uint16 CYBLE_GAPP_DISC_PARAM_T::advIntvMax**
Maximum advertising interval for undirected and low duty cycle directed advertising.

- Time Range: 20 ms to 10.24 sec

**CYBLE_GAPP_ADV_T CYBLE_GAPP_DISC_PARAM_T::advType**
Type of advertisement

- Connectable undirected advertising (0x00)
- Connectable high duty cycle directed advertising (0x01)
- Scannable undirected advertising (0x02)
- Non connectable undirected advertising (0x03)
- Connectable low duty cycle directed advertising (0x04)

**uint8 CYBLE_GAPP_DISC_PARAM_T::ownAddrType**

Own BD Address Type

- CYBLE_GAP_ADDR_TYPE_PUBLIC
- CYBLE_GAP_ADDR_TYPE_RANDOM
- CYBLE_GAP_ADDR_TYPE_PUBLIC_RPA
- CYBLE_GAP_ADDR_TYPE_RANDOM_RPA

**uint8 CYBLE_GAPP_DISC_PARAM_T::directAddrType**

Address type of the Bluetooth device address being used for directed advertising, not applicable otherwise

- CYBLE_PUBLIC_DEV_ADDR (Public device address)

CYBLE_RANDOM_DEV_ADDR (Random device address)

**uint8 CYBLE_GAPP_DISC_PARAM_T::directAddr[(0x06u)]**

This parameter specifies Bluetooth device address of the device to be connected while using directed advertising. In case of none direct advertising, parameter will be 0

**uint8 CYBLE_GAPP_DISC_PARAM_T::advChannelMap**

Advertising channels that shall be used when transmitting advertising packets. Channel map selection:

- Enable channel 37 = bitmask. xxxxxxx1b
- Enable channel 38 = bitmask. xxxxxx1xb
- Enable channel 39 = bitmask. xxxxx1xxb

**uint8 CYBLE_GAPP_DISC_PARAM_T::advFilterPolicy**

Advertising Filter Policy

- CYBLE_GAPP_SCAN_ANY_CONN_ANY (Allow Scan Request from Any, Allow Connect Request from Any (Default))
- CYBLE_GAPP_SCAN_WHITELIST_CONN_ANY (Allow Scan Request from White List Only, Allow Connect Request)
- CYBLE_GAPP_SCAN_ANY_CONN_WHITELIST (Allow Scan Request from Any, Allow Connect Request from White List Only)
- CYBLE_GAPP_SCAN_CONN_WHITELIST_ONLY (Allow Scan Request from White List Only, Allow Connect Request from White List Only)

**struct CYBLE_GAPP_DISC_DATA_T**

**Data Fields**

- uint8 advData [31u]
- uint8 advDataLen

**Field Documentation**

**uint8 CYBLE_GAPP_DISC_DATA_T::advData[31u]**

GAP Advertisement Parameters which includes Flags, Service UUIDs and short name

**uint8 CYBLE_GAPP_DISC_DATA_T::advDataLen**

length of the advertising data. This should be made zero if there is no data

**struct CYBLE_GAPP_SCAN_RSP_DATA_T**

**Data Fields**

- uint8 scanRspData [31u]
- uint8 scanRspDataLen

**Field Documentation**

**uint8 CYBLE_GAPP_SCAN_RSP_DATA_T::scanRspData[31u]**

Static user data transmitted in scan response. This should be made NULL if there is no data. Maximum length of the data is equal to 31 bytes

**uint8 CYBLE_GAPP_SCAN_RSP_DATA_T::scanRspDataLen**

Length of the scan response data. This should be made zero if there is no data

**struct CYBLE_GAPP_DISC_MODE_INFO_T**

**Data Fields**

- uint8 discMode
- CYBLE_GAPP_DISC_PARAM_T * advParam
- CYBLE_GAPP_DISC_DATA_T * advData
- CYBLE_GAPP_SCAN_RSP_DATA_T * scanRspData
- uint16 advTo

**Field Documentation**

**uint8 CYBLE_GAPP_DISC_MODE_INFO_T::discMode**

Broadcaster and discoverable mode

- CYBLE_GAPP_NONE_DISC_BROADCAST_MODE (Applicable for Broadcaster or non-discoverable mode)
- CYBLE_GAPP_LTD_DISC_MODE (Limited discovery mode)
- CYBLE_GAPP_GEN_DISC_MODE (General discovery mode)

**CYBLE_GAPP_DISC_PARAM_T* CYBLE_GAPP_DISC_MODE_INFO_T::advParam**

Advertisement parameters

**CYBLE_GAPP_DISC_DATA_T* CYBLE_GAPP_DISC_MODE_INFO_T::advData**

Advertisement data

**CYBLE_GAPP_SCAN_RSP_DATA_T* CYBLE_GAPP_DISC_MODE_INFO_T::scanRspData**

Scan Response data

**uint16 CYBLE_GAPP_DISC_MODE_INFO_T::advTo**

Advertisement timeout is in seconds. If timeout is set to 0, then there will not be any timeout. Parameter 'advTo' can be used for all GAP timeouts related to peripheral operation. For General discoverable mode, this timer will be ignored. Application is expected to exit from discoverable mode explicitly by calling CyBle_GappExitDiscoveryMode() function. For Limited discoverable mode, 'advTo' should not exceed 180 Sec.

**struct CYBLE_GAPC_DISC_INFO_T**

**Data Fields**

- uint8 discProcedure
- uint8 scanType
- uint16 scanIntv
- uint16 scanWindow
- uint8 ownAddrType
- uint8 scanFilterPolicy
- uint16 scanTo
- uint8 filterDuplicates

**Field Documentation**

**uint8 CYBLE_GAPC_DISC_INFO_T::discProcedure**

Observation and discovery procedure.

- CYBLE_GAPC_OBSER_PROCEDURE (Observation procedure)
- CYBLE_GAPC_LTD_DISC_PROCEDURE (Limited discovery procedure)
- CYBLE_GAPC_GEN_DISC_PROCEDURE (General discovery procedure)

**uint8 CYBLE_GAPC_DISC_INFO_T::scanType**

Type of scan to perform

- CYBLE_GAPC_PASSIVE_SCANNING (Passive Scanning)
- CYBLE_GAPC_ACTIVE_SCANNING (Active scanning)

**uint16 CYBLE_GAPC_DISC_INFO_T::scanIntv**

The time interval from when last LE scan is started until next subsequent LE scan.

- Time Range: 2.5 ms to 10.24 sec.

**uint16 CYBLE_GAPC_DISC_INFO_T::scanWindow**

The time duration of scanning to be performed

- Time Range: 2.5 ms to 10.24 sec

**uint8 CYBLE_GAPC_DISC_INFO_T::ownAddrType**

Own BD Address Type

- CYBLE_GAP_ADDR_TYPE_PUBLIC
- CYBLE_GAP_ADDR_TYPE_RANDOM
- CYBLE_GAP_ADDR_TYPE_PUBLIC_RPA
- CYBLE_GAP_ADDR_TYPE_RANDOM_RPA

**uint8 CYBLE_GAPC_DISC_INFO_T::scanFilterPolicy**

Filter policies to be applied during scanning procedure

- CYBLE_GAPC_ADV_ACCEPT_ALL_PKT
- CYBLE_GAPC_ADV_ACCEPT_WHITELIST_PKT
- CYBLE_GAPC_ADV_ACCEPT_DIRECTED_RPA_PKT
- CYBLE_GAPC_ADV_ACCEPT_WHITELIST_DIRECTED_RPA_PKT

**uint16 CYBLE_GAPC_DISC_INFO_T::scanTo**

Scan timeout. Timeout is in seconds and none zero. If timeout is set as 0, then there will not be any timeout scanTo can be used for all GAP timeouts related to Central operation.

**uint8 CYBLE_GAPC_DISC_INFO_T::filterDuplicates**

Filter Duplicate Advertisement. The Filter Duplicates parameter controls whether the Link Layer shall filter duplicate advertising reports to the Host, or if the Link Layer should generate advertising reports for each packet received.

- CYBLE_GAPC_FILTER_DUP_DISABLE (Duplicate filtering disabled)
- CYBLE_GAPC_FILTER_DUP_ENABLE (Duplicate filtering enabled)

By default, duplicate filtering is enabled

**struct CYBLE_GAPC_CONN_PARAM_T**

**Data Fields**

- uint16 scanIntv
- uint16 scanWindow
- uint8 initiatorFilterPolicy

- uint8 peerBdAddr [(0x06u)]
- uint8 peerAddrType
- uint8 ownAddrType
- uint16 connIntvMin
- uint16 connIntvMax
- uint16 connLatency
- uint16 supervisionTO
- uint16 minCeLength
- uint16 maxCeLength

**Field Documentation**

**uint16 CYBLE_GAPC_CONN_PARAM_T::scanIntv**

The time interval from when last LE scan is started until next subsequent LE scan.

- Time Range: 2.5 ms to 10.24 sec.

**uint16 CYBLE_GAPC_CONN_PARAM_T::scanWindow**

The time duration of scanning to be performed

- Time Range: 2.5 ms to 10.24 sec

**uint8 CYBLE_GAPC_CONN_PARAM_T::initiatorFilterPolicy**

Filter policies to be applied during connection procedure

- CYBLE_GAPC_CONN_ALL (White list is not used to determine which advertiser to connect. Peer address is used)
- CYBLE_GAPC_CONN_WHITELIST (White list is used to determine which advertiser to connect to. Peer address shall be ignored)

**uint8 CYBLE_GAPC_CONN_PARAM_T::peerBdAddr[(0x06u)]**

Peer's bd address with whom connection to be established

**uint8 CYBLE_GAPC_CONN_PARAM_T::peerAddrType**

Peer's bd address type

- CYBLE_GAP_ADDR_TYPE_PUBLIC
- CYBLE_GAP_ADDR_TYPE_RANDOM
- CYBLE_GAP_ADDR_TYPE_PUBLIC_RPA
- CYBLE_GAP_ADDR_TYPE_RANDOM_RPA

**uint8 CYBLE_GAPC_CONN_PARAM_T::ownAddrType**

Own bd address type

- CYBLE_GAP_ADDR_TYPE_PUBLIC
- CYBLE_GAP_ADDR_TYPE_RANDOM
- CYBLE_GAP_ADDR_TYPE_PUBLIC_RPA
- CYBLE_GAP_ADDR_TYPE_RANDOM_RPA

**uint16 CYBLE_GAPC_CONN_PARAM_T::connIntvMin**

Minimum value for the connection event interval. This shall be less than or equal to conn_Interval_Max. Minimum connection interval will be connIntvMin * 1.25 ms Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAPC_CONN_PARAM_T::connIntvMax**

Maximum value for the connection event interval. This shall be greater than or equal to conn_Interval_Min. Maximum connection interval will be connIntvMax * 1.25 ms Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAPC_CONN_PARAM_T::connLatency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F4

**uint16 CYBLE_GAPC_CONN_PARAM_T::supervisionTO**
Supervision timeout for the LE Link. Supervision timeout will be supervisionTO * 10 ms Time Range: 100 msec to 32 secs

**uint16 CYBLE_GAPC_CONN_PARAM_T::minCeLength**
Minimum length of connection needed for this LE connection. Range: 0x0000 - 0xFFFF

**uint16 CYBLE_GAPC_CONN_PARAM_T::maxCeLength**
Maximum length of connection needed for this LE connection. Range: 0x0000 - 0xFFFF

### struct CYBLE_GAPC_ADV_REPORT_T

**Data Fields**

- CYBLE_GAPC_ADV_EVENT_T eventType
- uint8 peerAddrType
- uint8 * peerBdAddr
- uint8 dataLen
- uint8 * data
- int8 rssi

**Field Documentation**

**CYBLE_GAPC_ADV_EVENT_T CYBLE_GAPC_ADV_REPORT_T::eventType**
Advertisement event type
- Connectable undirected advertising = 0x00
- Connectable directed advertising = 0x01
- Scannable undirected advertising = 0x02
- Non connectable undirected advertising = 0x03
- Scan Response = 0x04

**uint8 CYBLE_GAPC_ADV_REPORT_T::peerAddrType**
bd address type of the device advertising.
- CYBLE_GAP_ADDR_TYPE_PUBLIC
- CYBLE_GAP_ADDR_TYPE_RANDOM
- CYBLE_GAP_ADDR_TYPE_PUBLIC_RPA
- CYBLE_GAP_ADDR_TYPE_RANDOM_RPA

**uint8* CYBLE_GAPC_ADV_REPORT_T::peerBdAddr**
Public Device Address or Random Device Address for each device which responded to scanning.

**uint8 CYBLE_GAPC_ADV_REPORT_T::dataLen**
length of the data for each device that responded to scanning

**uint8* CYBLE_GAPC_ADV_REPORT_T::data**
Pointer to advertising or scan response data

**int8 CYBLE_GAPC_ADV_REPORT_T::rssi**
Rssi of the responding device. Range: -85 <= N <= 0 Units: dBm

### struct CYBLE_GAP_PASSKEY_DISP_INFO_T

**Data Fields**

- uint8 bdHandle
- uint32 passkey

**Field Documentation**

**uint8 CYBLE_GAP_PASSKEY_DISP_INFO_T::bdHandle**
bd handle of the remote device

**uint32 CYBLE_GAP_PASSKEY_DISP_INFO_T::passkey**
size = 6, not null terminated

**struct CYBLE_GAP_CONN_UPDATE_PARAM_T**

**Data Fields**

- uint16 connIntvMin
- uint16 connIntvMax
- uint16 connLatency
- uint16 supervisionTO

**Field Documentation**

**uint16 CYBLE_GAP_CONN_UPDATE_PARAM_T::connIntvMin**
Minimum value for the connection event interval. This shall be less than or equal to conn_Interval_Max. Minimum connection interval will be connIntvMin * 1.25 ms Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAP_CONN_UPDATE_PARAM_T::connIntvMax**
Maximum value for the connection event interval. This shall be greater than or equal to conn_Interval_Min. Maximum connection interval will be connIntvMax * 1.25 ms Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAP_CONN_UPDATE_PARAM_T::connLatency**
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

**uint16 CYBLE_GAP_CONN_UPDATE_PARAM_T::supervisionTO**
Supervision timeout for the LE Link. Supervision timeout will be supervisionTO * 10 ms Time Range: 100 msec to 32 secs

**struct CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T**

**Data Fields**

- uint8 status
- uint16 connIntv
- uint16 connLatency
- uint16 supervisionTO

**Field Documentation**

**uint8 CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T::status**
status corresponding to this event will be HCI error code as defined in BLE spec 4.1 or User can refer CYBLE_HCI_ERROR_T for HCI error codes

**uint16 CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T::connIntv**
Connection interval used on this connection. Range: 0x0006 to 0x0C80 Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T::connLatency**
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

**uint16 CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T::supervisionTO**
Supervision timeout for the LE Link. Supervision timeout will be supervisionTO * 10 ms Time Range: 100 msec to 32 secs

**struct CYBLE_GAP_OOB_DATA_T**

**Data Fields**

- uint8 status

- uint8 * key
- uint8 * oobData
- uint8 oobDataLen

**Field Documentation**

**uint8 CYBLE_GAP_OOB_DATA_T::status**
Status corresponding to this event will be HCI error code as defined in BLE spec 4.2

**uint8* CYBLE_GAP_OOB_DATA_T::key**
Rand for OOB. This is also stored in stack

**uint8* CYBLE_GAP_OOB_DATA_T::oobData**
OOB Data using 'key' and local Public Key

**uint8 CYBLE_GAP_OOB_DATA_T::oobDataLen**
Length of OOB data which is 16 Bytes for Secure connections

**struct CYBLE_GAP_DATA_LENGTH_T**

**Data Fields**

- uint16 suggestedTxOctets
- uint16 suggestedTxTime
- uint16 maxTxOctets
- uint16 maxTxTime
- uint16 maxRxOctets
- uint16 maxRxTime

**Field Documentation**

**uint16 CYBLE_GAP_DATA_LENGTH_T::suggestedTxOctets**
Controller's maximum transmitted number of payload octets to be used for new connections

**uint16 CYBLE_GAP_DATA_LENGTH_T::suggestedTxTime**
Controller's maximum packet transmission time to be used for new connections

**uint16 CYBLE_GAP_DATA_LENGTH_T::maxTxOctets**
Maximum number of payload octets that the local Controller supports for transmission of a single Link Layer Data Channel PDU.

**uint16 CYBLE_GAP_DATA_LENGTH_T::maxTxTime**
Maximum time, in microseconds, that the local Controller supports for transmission of a single Link Layer Data Channel PDU.

**uint16 CYBLE_GAP_DATA_LENGTH_T::maxRxOctets**
Maximum number of payload octets that the local Controller supports for reception of a single Link Layer Data Channel PDU.

**uint16 CYBLE_GAP_DATA_LENGTH_T::maxRxTime**
Maximum time, in microseconds, that the local Controller supports for reception of a single Link Layer Data Channel PDU.

**struct CYBLE_GAP_CONN_DATA_LENGTH_T**

**Data Fields**

- uint16 connMaxTxOctets
- uint16 connMaxTxTime
- uint16 connMaxRxOctets
- uint16 connMaxRxTime

**Field Documentation**

**uint16 CYBLE_GAP_CONN_DATA_LENGTH_T::connMaxTxOctets**

The maximum number of payload octets in a Link Layer Data Channel PDU that the local Controller will send on current connection.

**uint16 CYBLE_GAP_CONN_DATA_LENGTH_T::connMaxTxTime**

The maximum time that the local Controller will take to send a Link Layer Data Channel PDU on current connection

**uint16 CYBLE_GAP_CONN_DATA_LENGTH_T::connMaxRxOctets**

The maximum number of payload octets in a Link Layer Data Channel PDU that the local controller expects to receive on current connection

**uint16 CYBLE_GAP_CONN_DATA_LENGTH_T::connMaxRxTime**

The maximum time that the local Controller expects to take to receive a Link Layer Data Channel PDU on this connection

**struct CYBLE_GAP_RX_DATA_LENGTH_T**

**Data Fields**

- uint8 bdHandle
- uint16 connMaxRxOctets
- uint16 connMaxRxTime

**Field Documentation**

**uint8 CYBLE_GAP_RX_DATA_LENGTH_T::bdHandle**

Peer bdHandle

**uint16 CYBLE_GAP_RX_DATA_LENGTH_T::connMaxRxOctets**

The maximum number of payload octets in a Link Layer Data Channel PDU that the local controller expects to receive on current connection

**uint16 CYBLE_GAP_RX_DATA_LENGTH_T::connMaxRxTime**

The maximum time that the local Controller expects to take to receive a Link Layer Data Channel PDU on this connection

**struct CYBLE_GAP_RESOLVING_DEVICE_INFO_T**

**Data Fields**

- uint8 peerIrk [16u]
- uint8 localIrk [16u]
- uint8 bdAddr [(0x06u)]
- uint8 type

**Field Documentation**

**uint8 CYBLE_GAP_RESOLVING_DEVICE_INFO_T::peerIrk[16u]**

Peer IRK

**uint8 CYBLE_GAP_RESOLVING_DEVICE_INFO_T::localIrk[16u]**

Local IRK

**uint8 CYBLE_GAP_RESOLVING_DEVICE_INFO_T::bdAddr[(0x06u)]**

Peer Identity device address

**uint8 CYBLE_GAP_RESOLVING_DEVICE_INFO_T::type**

Peer Identity addr type

**struct CYBLE_GAP_RESOLVING_LIST_T**

**Data Fields**

- CYBLE_GAP_RESOLVING_DEVICE_INFO_T resolvingList [0x08u]
- uint8 noOfDevice

**Field Documentation**

**CYBLE_GAP_RESOLVING_DEVICE_INFO_T CYBLE_GAP_RESOLVING_LIST_T::resolvingList[0x08u]**
Pointer to Resolving list stored in controller

**uint8 CYBLE_GAP_RESOLVING_LIST_T::noOfDevice**
Number of entries in resolving list

**struct CYBLE_GAPC_DIRECT_ADV_REPORT_T**

**Data Fields**

- uint8 * localBdAddr
- uint8 * peerBdAddr
- CYBLE_GAP_ADV_ADDR_TYPE_T peerBdAddrType
- int8 rssi

**Field Documentation**

**uint8* CYBLE_GAPC_DIRECT_ADV_REPORT_T::localBdAddr**
Buffer containing Random Device Address of Scanner (local device) This is the address the directed advertisements are being directed to.

**uint8* CYBLE_GAPC_DIRECT_ADV_REPORT_T::peerBdAddr**
Buffer containing Device Address of advertiser sending the directed advertisement

**CYBLE_GAP_ADV_ADDR_TYPE_T CYBLE_GAPC_DIRECT_ADV_REPORT_T::peerBdAddrType**
Device Address type of advertiser sending the directed advertisement

**int8 CYBLE_GAPC_DIRECT_ADV_REPORT_T::rssi**
Rssi of the responding device. Range: -127 <= N <= +20 Units: dBm N = 127 -> RSSI not available

**struct CYBLE_GAP_ENHANCE_CONN_COMPLETE_T**

**Data Fields**

- uint16 connIntv
- uint16 connLatency
- uint16 supervisionTo
- uint8 * peerBdAddr
- CYBLE_GAP_ADV_ADDR_TYPE_T peerBdAddrType
- uint8 * localResolvablePvtAddr
- uint8 * peerResolvablePvtAddr
- uint8 role
- uint8 masterClockAccuracy
- uint8 status

**Field Documentation**

**uint16 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::connIntv**
Connection interval used on this connection. Range: 0x0006 to 0x0C80 Time Range: 7.5 ms to 4 sec

**uint16 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::connLatency**
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

**uint16 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::supervisionTo**

Supervision timeout for the LE Link. Supervision timeout will be supervisionTO * 10 ms Time Range: 100 msec to 32 secs

**uint8* CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::peerBdAddr**

Peer Device Address

**CYBLE_GAP_ADV_ADDR_TYPE_T CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::peerBdAddrType**

Peer Device Address type

**uint8* CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::localResolvablePvtAddr**

Local Resolvable Private Address Resolvable Private Address being used by the local device for this connection. This is only valid when the Own_Address_Type in connection/advertisement parameters is set to 0x02 or 0x03. For other Own_Address_Type values, This will be all zeros.

**uint8* CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::peerResolvablePvtAddr**

Peer Resolvable Private Address Resolvable Private Address being used by the peer device for this connection. This is only valid for the Peer_Address_Type 0x02 or 0x03. For other Peer_Address_Type values, This will be all zeros.

**uint8 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::role**

Connection is master/slave Master = 0x00 Slave = 0x01

**uint8 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::masterClockAccuracy**

Master clock accuracy 0x00 -> 500 ppm 0x01 -> 250 ppm 0x02 -> 150 ppm 0x03 -> 100 ppm 0x04 -> 75 ppm 0x05 -> 50 ppm 0x06 -> 30 ppm 0x07 -> 20 ppm

**uint8 CYBLE_GAP_ENHANCE_CONN_COMPLETE_T::status**

Status corresponding to this event will be HCI error code. Values of 0 indicates connection successfully completed. Refer BLE spec 4.2,Vol2, Part D for Error codes or User can refer CYBLE_HCI_ERROR_T for HCI error codes.

### struct CYBLE_GAP_DEVICE_LIST_T

**Data Fields**

- CYBLE_GAP_BD_ADDR_T bdAddr
- uint8 bdHandle

**Field Documentation**

**CYBLE_GAP_BD_ADDR_T CYBLE_GAP_DEVICE_LIST_T::bdAddr**

Bluetooth device address

**uint8 CYBLE_GAP_DEVICE_LIST_T::bdHandle**

Corresponding bdHandle

### struct CYBLE_GAP_DEVICE_ADDR_LIST_T

**Data Fields**

- CYBLE_GAP_DEVICE_LIST_T bdHandleAddrList [0x04u]
- uint8 count

**Field Documentation**

**CYBLE_GAP_DEVICE_LIST_T CYBLE_GAP_DEVICE_ADDR_LIST_T::bdHandleAddrList[0x04u]**

Pointer to list of Bluetooth device addresses and bdHandle of bonded devices

**uint8 CYBLE_GAP_DEVICE_ADDR_LIST_T::count**

Number of bonded devices

**struct CYBLE_GAP_PRIVACY_MODE_INFO_T**

**Data Fields**

- uint8 peerBdAddr [(0x06u)]
- uint8 peerBdAddrType
- uint8 privacyMode

**Field Documentation**

**uint8 CYBLE_GAP_PRIVACY_MODE_INFO_T::peerBdAddr[(0x06u)]**
Bluetooth device address

**uint8 CYBLE_GAP_PRIVACY_MODE_INFO_T::peerBdAddrType**
public = 0, Random = 1

**uint8 CYBLE_GAP_PRIVACY_MODE_INFO_T::privacyMode**
Privacy Mode

## Enumeration Type Documentation

**enum CYBLE_GAP_ADV_ASSIGN_NUMBERS**

Stack mode defines Advertisement SIG assigned numbers

**Enumerator**

***CYBLE_GAP_ADV_FLAGS*** Flags
***CYBLE_GAP_ADV_INCOMPL_16UUID*** Incomplete List of 16-bit Service Class UUIDs
***CYBLE_GAP_ADV_COMPL_16UUID*** Complete List of 16-bit Service Class UUIDs
***CYBLE_GAP_ADV_INCOMPL_32_UUID*** Incomplete List of 32-bit Service Class UUIDs
***CYBLE_GAP_ADV_COMPL_32_UUID*** Complete List of 32-bit Service Class UUIDs
***CYBLE_GAP_ADV_INCOMPL_128_UUID*** Incomplete List of 128-bit Service Class UUIDs
***CYBLE_GAP_ADV_COMPL_128_UUID*** Complete List of 128-bit Service Class UUIDs
***CYBLE_GAP_ADV_SHORT_NAME*** Shortened Local Name
***CYBLE_GAP_ADV_COMPL_NAME*** Complete Local Name
***CYBLE_GAP_ADV_TX_PWR_LVL*** Tx Power Level
***CYBLE_GAP_ADV_CLASS_OF_DEVICE*** Class of Device
***CYBLE_GAP_ADV_SMPL_PAIR_HASH_C*** Simple Pairing Hash C
***CYBLE_GAP_ADV_SMPL_PAIR_RANDOM_R*** Simple Pairing Randomizer R
***CYBLE_GAP_ADV_DEVICE_ID*** Device ID
***CYBLE_GAP_ADV_SCRT_MNGR_TK_VAL*** Security Manager TK Value
***CYBLE_GAP_ADV_SCRT_MNGR_OOB_FLAGS*** Security Manager Out of Band Flags
***CYBLE_GAP_ADV_SLAVE_CONN_INTRV_RANGE*** Slave Connection Interval Range
***CYBLE_GAP_ADV_SOLICIT_16UUID*** List of 16-bit Service Solicitation UUIDs
***CYBLE_GAP_ADV_SOLICIT_128UUID*** List of 128-bit Service Solicitation UUIDs
***CYBLE_GAP_ADV_SRVC_DATA_16UUID*** Service Data - 16-bit UUID
***CYBLE_GAP_ADV_PUBLIC_TARGET_ADDR*** Public Target Address
***CYBLE_GAP_ADV_RANDOM_TARGET_ADDR*** Random Target Address
***CYBLE_GAP_ADV_APPEARANCE*** Appearance
***CYBLE_GAP_ADV_ADVERT_INTERVAL*** Advertising Interval
***CYBLE_GAP_ADV_LE_BT_DEVICE_ADDR*** LE Bluetooth Device Address
***CYBLE_GAP_ADV_LE_ROLE*** LE Role

***CYBLE_GAP_ADV_SMPL_PAIR_HASH_C256*** Simple Pairing Hash C-256

***CYBLE_GAP_ADV_SMPL_PAIR_RANDOM_R256*** Simple Pairing Randomizer R-256

***CYBLE_GAP_ADV_SOLICIT_32UUID*** List of 32-bit Service Solicitation UUIDs

***CYBLE_GAP_ADV_SRVC_DATA_32UUID*** Service Data - 32-bit UUID

***CYBLE_GAP_ADV_SRVC_DATA_128UUID*** Service Data - 128-bit UUID

***CYBLE_GAP_ADV_3D_INFO_DATA*** 3D Information Data

**enum CYBLE_GAPP_ADV_T**

Advertisement type

**Enumerator**

***CYBLE_GAPP_CONNECTABLE_UNDIRECTED_ADV*** Connectable undirected advertising

***CYBLE_GAPP_CONNECTABLE_HIGH_DC_DIRECTED_ADV*** Connectable high duty cycle directed advertising

***CYBLE_GAPP_SCANNABLE_UNDIRECTED_ADV*** Scannable undirected advertising

***CYBLE_GAPP_NON_CONNECTABLE_UNDIRECTED_ADV*** Non connectable undirected advertising

***CYBLE_GAPP_CONNECTABLE_LOW_DC_DIRECTED_ADV*** Connectable low duty cycle directed advertising

**enum CYBLE_GAPC_ADV_EVENT_T**

Advertisement event type

**Enumerator**

***CYBLE_GAPC_CONN_UNDIRECTED_ADV*** Connectable undirected advertising

***CYBLE_GAPC_CONN_DIRECTED_ADV*** Connectable directed advertising

***CYBLE_GAPC_SCAN_UNDIRECTED_ADV*** Scannable undirected advertising

***CYBLE_GAPC_NON_CONN_UNDIRECTED_ADV*** Non connectable undirected advertising

***CYBLE_GAPC_SCAN_RSP*** Scan Response

**enum CYBLE_GAP_SEC_LEVEL_T**

Security Levels

**Enumerator**

***CYBLE_GAP_SEC_LEVEL_1*** Level 1 Mode 1 - No Security (No Authentication & No Encryption) Mode 2 - N/A

***CYBLE_GAP_SEC_LEVEL_2*** Level 2 Mode 1 - Unauthenticated pairing with encryption (No MITM) Mode 2 - Unauthenticated pairing with data signing (No MITM)

***CYBLE_GAP_SEC_LEVEL_3*** Level 3 Mode 1 - Authenticated pairing with encryption (With MITM) Mode 2 - Authenticated pairing with data signing (With MITM)

***CYBLE_GAP_SEC_LEVEL_4*** Level 4 Secured Connection

***CYBLE_GAP_SEC_LEVEL_MASK*** LE Security Level Mask

**enum CYBLE_GAP_IOCAP_T**

IO capability

**Enumerator**

***CYBLE_GAP_IOCAP_DISPLAY_ONLY*** Platform supports only a mechanism to display or convey only 6 digit number to user.

***CYBLE_GAP_IOCAP_DISPLAY_YESNO*** The device has a mechanism whereby the user can indicate 'yes' or 'no'.

***CYBLE_GAP_IOCAP_KEYBOARD_ONLY*** Platform supports a numeric keyboard that can input the numbers '0' through '9' and a confirmation key(s) for 'yes' and 'no'.

***CYBLE_GAP_IOCAP_NOINPUT_NOOUTPUT*** Platform does not have the ability to display or communicate a 6 digit decimal number.

***CYBLE_GAP_IOCAP_KEYBOARD_DISPLAY*** Platform supports a mechanism through which 6 digit numeric value can be displayed and numeric keyboard that can input the numbers '0' through '9'.

**enum CYBLE_GAP_AUTH_FAILED_REASON_T**

Authentication Failed Error Codes

**Enumerator**

***CYBLE_GAP_AUTH_ERROR_NONE*** No Error

***CYBLE_GAP_AUTH_ERROR_PASSKEY_ENTRY_FAILED*** User input of passkey failed, for example, the user cancelled the operation

***CYBLE_GAP_AUTH_ERROR_OOB_DATA_NOT_AVAILABLE*** Out Of Band data is not available, applicable if NFC is supported

***CYBLE_GAP_AUTH_ERROR_AUTHENTICATION_REQ_NOT_MET*** Pairing procedure cannot be performed as authentication requirements cannot be met due to IO capabilities of one or both devices.

***CYBLE_GAP_AUTH_ERROR_CONFIRM_VALUE_NOT_MATCH*** Confirm value does not match the calculated compare value

***CYBLE_GAP_AUTH_ERROR_PAIRING_NOT_SUPPORTED*** Pairing is not supported by the device

***CYBLE_GAP_AUTH_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE*** Insufficient key size for the security requirements of this device or LTK is lost

***CYBLE_GAP_AUTH_ERROR_COMMAND_NOT_SUPPORTED*** command received is not supported

***CYBLE_GAP_AUTH_ERROR_UNSPECIFIED_REASON*** Pairing failed due to an unspecified reason

***CYBLE_GAP_AUTH_ERROR_REPEATED_ATTEMPTS*** Pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request.

***CYBLE_GAP_AUTH_ERROR_INVALID_PARAMETERS*** Invalid Parameters in Request - Invalid Command length and Parameter value outside range

***CYBLE_GAP_AUTH_ERROR_DHKEY_CHECK_FAILED*** Indicates to the remote device that the DHKey Check value received doesn't match the one calculated by the local device

***CYBLE_GAP_AUTH_ERROR_NUMERIC_COMPARISON_FAILED*** Indicates that the confirm values in the numeric comparison protocol do not match

***CYBLE_GAP_AUTH_ERROR_BR_EDR_PAIRING_IN_PROGRESS*** Indicates that the pairing over the LE transport failed due to a Pairing Request sent over the BR/EDR transport is in process.

***CYBLE_GAP_AUTH_ERROR_CROSS_TRANSPORT_KEY_GEN_DER_NOT_ALLOWED*** Indicates that the BR/EDR Link Key generated on the BR/EDR transport cannot be used to derive and distribute keys for LE transport

***CYBLE_GAP_AUTH_ERROR_CODE_SPEC_MAX_VALUE*** Indicates that over the air, spec will not allow error code value to be greater than 0x0E

***CYBLE_GAP_AUTH_ERROR_AUTHENTICATION_TIMEOUT*** Authentication process timeout, if pairing timeout happens for first time, application can choose to re-initiate the pairing procedure. If timeout occurs again, app may choose to disconnect peer device.

***CYBLE_GAP_AUTH_ERROR_LINK_DISCONNECTED*** Link disconnected

**enum CYBLE_GAP_ADDR_TYPE_T**

GAP address type

**Enumerator**

***CYBLE_GAP_RANDOM_PRIV_NON_RESOLVABLE_ADDR*** Random private non-resolvable address

***CYBLE_GAP_RANDOM_PRIV_RESOLVABLE_ADDR*** Random private resolvable address
***CYBLE_GAP_PUBLIC_ADDR*** Public address
***CYBLE_GAP_RANDOM_STATIC_ADDR*** Random static address

**enum CYBLE_GAP_KEYPRESS_NOTIFY_TYPE**

Passkey entry notification types. These are used for CyBle_GapAuthSendKeyPress() function as well as with CYBLE_EVT_GAP_KEYPRESS_NOTIFICATION event parameter.

**Enumerator**

***CYBLE_GAP_PASSKEY_ENTRY_STARTED*** Passkey entry started
***CYBLE_GAP_PASSKEY_DIGIT_ENTERED*** One digit entered
***CYBLE_GAP_PASSKEY_DIGIT_ERASED*** One digit erased
***CYBLE_GAP_PASSKEY_CLEARED*** All digits cleared
***CYBLE_GAP_PASSKEY_ENTRY_COMPLETED*** Passkey entry completed

**enum CYBLE_GAP_ADV_ADDR_TYPE_T**

GAP Direct advertiser address type

**Enumerator**

***CYBLE_GAP_PUBLIC_ADDR_TYPE*** Public device address type
***CYBLE_GAP_RANDOM_RESOLVABLE_ADDR_TYPE*** Random private resolvable address type
***CYBLE_GAP_PUBLIC_IDENTITY_ADDR_TYPE*** Public Identity address type
***CYBLE_GAP_RANDOM_IDENTITY_ADDR_TYPE*** Random static Identity Address

**enum CYBLE_GAP_PHY_TYPE_T**

GAP physical layer

**Enumerator**

***CYBLE_GAP_PHY_1MBPS*** 1 - Mbps Physical Layer.
***CYBLE_GAP_PHY_INVALID*** Reserved Values.

# GATT Functions

## Description

The GATT APIs allow access to the Generic Attribute Profile (GATT) layer of the BLE stack. Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The GATT API names begin with CyBle_Gatt. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- GATT Client and Server Functions
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- GATT Client Functions
  *APIs unique to designs configured as a GATT Client role.*
- GATT Server Functions
  *APIs unique to designs configured as a GATT Server role.*
- GATT Definitions and Data Structures

*Contains the GATT specific definitions and data structures used in the GATT APIs.*

# GATT Client and Server Functions

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Gatt

## Functions

- CYBLE_API_RESULT_T CyBle_GattGetMtuSize (uint16 *mtu)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GattGetMtuSize (uint16 *_mtu_)**

This function provides the correct GATT MTU used by BLE stack. If function is called after GATT MTU configuration procedure, it will provide the final negotiated GATT MTU else default MTU (23 Bytes).

**Parameters:**

| | |
|---|---|
| *mtu* | buffer where Size of GATT MTU will be stored. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If invalid parameter passed |

# GATT Client Functions

## Description

APIs unique to designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Gattc

## Functions

- CYBLE_API_RESULT_T CyBle_GattcStartDiscovery (CYBLE_CONN_HANDLE_T connHandle)
- CYBLE_API_RESULT_T CyBle_GattcStartPartialDiscovery (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATT_DB_ATTR_HANDLE_T startHandle, CYBLE_GATT_DB_ATTR_HANDLE_T endHandle)
- void CyBle_GattcStopCmd (void)
- CYBLE_API_RESULT_T CyBle_GattcExchangeMtuReq (CYBLE_CONN_HANDLE_T connHandle, uint16 mtu)
- CYBLE_API_RESULT_T CyBle_GattcDiscoverAllPrimaryServices (CYBLE_CONN_HANDLE_T connHandle)
- CYBLE_API_RESULT_T CyBle_GattcDiscoverPrimaryServiceByUuid (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATT_VALUE_T value)
- CYBLE_API_RESULT_T CyBle_GattcFindIncludedServices (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATT_ATTR_HANDLE_RANGE_T *range)

- CYBLE_API_RESULT_T CyBle_GattcDiscoverAllCharacteristics (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATT_ATTR_HANDLE_RANGE_T range)
- CYBLE_API_RESULT_T CyBle_GattcDiscoverCharacteristicByUuid (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_BY_TYPE_REQ_T *readByTypeReqParam)
- CYBLE_API_RESULT_T CyBle_GattcDiscoverAllCharacteristicDescriptors (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_FIND_INFO_REQ_T *findInfoReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_REQ_T readReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadUsingCharacteristicUuid (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_BY_TYPE_REQ_T *readByTypeReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadLongCharacteristicValues (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_BLOB_REQ_T *readBlobReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadMultipleCharacteristicValues (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_MULT_REQ_T *readMultiReqParam)
- CYBLE_API_RESULT_T CyBle_GattcWriteWithoutResponse (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_WRITE_CMD_REQ_T *writeCmdReqParam)
- CYBLE_API_RESULT_T CyBle_GattcSignedWriteWithoutRsp (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_SIGNED_WRITE_CMD_REQ_T *signedWriteWithoutRspParam)
- CYBLE_API_RESULT_T CyBle_GattcWriteCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_WRITE_REQ_T *writeReqParam)
- CYBLE_API_RESULT_T CyBle_GattcWriteLongCharacteristicValues (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_PREP_WRITE_REQ_T *writePrepReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReliableWrites (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_PREP_WRITE_REQ_T *writePrepReqParam, uint8 numOfRequests)
- CYBLE_API_RESULT_T CyBle_GattcConfirmation (CYBLE_CONN_HANDLE_T connHandle)
- CYBLE_API_RESULT_T CyBle_GattcReadCharacteristicDescriptors (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_REQ_T readReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadLongCharacteristicDescriptors (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_BLOB_REQ_T *readBlobReqParam)
- CYBLE_API_RESULT_T CyBle_GattcWriteCharacteristicDescriptors (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_WRITE_REQ_T *writeReqParam)
- CYBLE_API_RESULT_T CyBle_GattcWriteLongCharacteristicDescriptors (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_PREP_WRITE_REQ_T *writePrepReqParam)
- CYBLE_API_RESULT_T CyBle_GattcReadByTypeReq (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTC_READ_BY_TYPE_REQ_T *readByTypeReqParam)
- CYBLE_API_RESULT_T CyBle_GattcSendExecuteWriteReq (CYBLE_CONN_HANDLE_T connHandle, uint8 flag)
- CYBLE_API_RESULT_T CyBle_GattcDiscoverPrimaryServices (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATT_ATTR_HANDLE_RANGE_T *range)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GattcStartDiscovery (CYBLE_CONN_HANDLE_T *connHandle*)**

Starts the automatic server discovery process.

Discovery procedure is based on the user configuration. It discovers only services, characteristics, descriptors which were declared in the GATT database. Discovery procedure has the following flow:

- discovering primary services by BLE Stack function CyBle_GattcDiscoverAllPrimaryServices();
- discovering included services by BLE Stack function CyBle_GattcFindIncludedServices();
- discovering characteristics for available services by BLE Stack function CyBle_GattcDiscoverAllCharacteristics();

- discovering characteristic descriptors by BLE Stack function CyBle_GattcDiscoverAllCharacteristicDescriptors();

During the discovery procedure the discovery-specific stack events are handled by the component and thus aren't passed to the application callback: CYBLE_EVT_GATTC_READ_BY_GROUP_TYPE_RSP, CYBLE_EVT_GATTC_READ_BY_TYPE_RSP, CYBLE_EVT_GATTC_FIND_INFO_RSP, CYBLE_EVT_GATTC_ERROR_RSP.

After the discovery procedure all information about available services is stored in CYBLE_DISC_SRVC_INFO_T structures, and discovered attributes handles are stored in service-specific client structures, such as CYBLE_BASC_T for Battery Service or CYBLE_HRSC_T for Heart Rate Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The handle which consists of the device ID and ATT connection ID. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry. |
| CYBLE_ERROR_INVALID_OPERATION | The operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_INVALID_STATE | If the function is called in any state except connected or discovered |

**Events**

The following events may be generated after calling this function:

- CYBLE_EVT_GATTC_DISCOVERY_COMPLETE - event is generated when the remote device was successfully discovered.
- CYBLE_EVT_GATTC_ERROR_RSP - is generated if the device discovery has failed.
- CYBLE_EVT_GATTC_SRVC_DUPLICATION - is generated if duplicate service record was found during the server device discovery.
- CYBLE_EVT_GATTC_CHAR_DUPLICATION - is generated if duplicate service's characteristic descriptor record was found during the server device discovery.
- CYBLE_EVT_GATTC_DESCR_DUPLICATION - is generated if duplicate service's characteristic descriptor record was found during the server device discovery.

**CYBLE_API_RESULT_T CyBle_GattcStartPartialDiscovery (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATT_DB_ATTR_HANDLE_T *startHandle*, CYBLE_GATT_DB_ATTR_HANDLE_T *endHandle*)**

Starts the automatic server discovery process as per the range provided on a GATT Server to which it is connected. This function could be used for partial server discovery after indication received to the Service Changed Characteristic Value.

**Parameters:**

| | |
|---|---|
| *connHandle* | The handle which consists of the device ID and ATT connection ID. |
| *startHandle* | Start of affected attribute handle range. |
| *endHandle* | End of affected attribute handle range. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry. |
| CYBLE_ERROR_INVALID_OPERATION | The operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_INVALID_STATE | If the function is called in any state except connected or discovered |

**Events**

Two events may be generated after calling this function: CYBLE_EVT_GATTC_DISCOVERY_COMPLETE or CYBLE_EVT_GATTC_ERROR_RSP. The CYBLE_EVT_GATTC_DISCOVERY_COMPLETE event is generated when the remote device was successfully discovered. The CYBLE_EVT_GATTC_ERROR_RSP is generated if the device discovery is failed.

### void CyBle_GattcStopCmd (void )

This function is used by the GATT Client to stop any of the following ongoing GATT procedures:

1. CyBle_GattcDiscoverAllPrimaryServices()
2. CyBle_GattcDiscoverPrimaryServiceByUuid()
3. CyBle_GattcFindIncludedServices()
4. CyBle_GattcDiscoverAllCharacteristics()
5. CyBle_GattcDiscoverCharacteristicByUuid()
6. CyBle_GattcDiscoverAllCharacteristicDescriptors()
7. CyBle_GattcReadLongCharacteristicValues()
8. CyBle_GattcWriteLongCharacteristicValues()
9. CyBle_GattcReliableWrites()
10. CyBle_GattcReadLongCharacteristicDescriptors()
11. CyBle_GattcWriteLongCharacteristicDescriptors()

If none of the above procedures is ongoing, then this command will be ignored. This function has no effect on ATT procedures other than those listed above.

If the user intends to start a new GATT procedure including those listed above and there is an ongoing GATT procedure (any one from the above list), the user needs to call this function to stop the ongoing GATT procedure and then invoke the desired GATT procedure. This is a blocking function. No event is generated on calling this function.

**Returns:**

None

### CYBLE_API_RESULT_T CyBle_GattcExchangeMtuReq (CYBLE_CONN_HANDLE_T *connHandle*, uint16 *mtu*)

This function is used by the GATT Client to send Maximum Transmitted Unit (GATT MTU) supported by the GATT Client. This is a non-blocking function.

Default GATT MTU size as per Bluetooth 4.1 core specification is 23 bytes. If the GATT Client supports a size greater than the default, it has to invoke this function with the desired GATT MTU size. This function should only be initiated once during a connection.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.3.1 for more details on GATT MTU exchange operation.

This function call results in CYBLE_EVT_GATTS_XCNHG_MTU_REQ event at the GATT Server's end in response to which the GATT Server is expected to send its GATT MTU size.

The CYBLE_EVT_GATTC_XCHNG_MTU_RSP event is generated at the GATT Client's end on receiving GATT MTU response from the GATT Server.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *mtu* | Size of GATT MTU. Max GATT MTU supported by BLE stack is 512 Bytes. |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack or, 'mtu' has a value which is greater than that set on calling CyBle_StackInit function |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcDiscoverAllPrimaryServices (CYBLE_CONN_HANDLE_T *connHandle*)

This function is used by the GATT Client to discover all the primary services on a GATT Server to which it is connected. This is a non-blocking function.

Internally, this function initiates multiple Read By Group Type Requests to the peer device in response to which it receives Read By Group Type Responses. Each Read By Group Type Response results in CYBLE_EVT_GATTC_READ_BY_GROUP_TYPE_RSP event, which is propagated to the application layer for handling.

Primary service discovery is complete when Error Response (CYBLE_EVT_GATTC_ERROR_RSP) is received and the Error Code is set to Attribute Not Found or when the End Group Handle in the Read by Group Type Response is 0xFFFF. Completion of this operation is notified to the upper layer(s) using CYBLE_EVT_GATTC_ERROR_RSP with error code updated appropriately.

It is permitted to end the above stated sequence of operations early if the desired primary service is found prior to discovering all the primary services on the GATT Server. This can be achieved by calling the CyBle_GattcStopCmd() function.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.4.1 for more details on this sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcDiscoverPrimaryServiceByUuid (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATT_VALUE_T *value*)**

This function is used by the GATT Client to discover a specific primary service on a GATT Server, to which it is connected, when only the Service UUID is known. This is a non-blocking function.

Internally, this function initiates multiple Find By Type Value Requests with the Attribute Type parameter set to the UUID for Primary Service and the Attribute Value set to the 16-bit Bluetooth UUID or 128-bit UUID for the specific primary service. Each Find By Type Value Response received from the peer device is passed to the application as CYBLE_EVT_GATTC_FIND_BY_TYPE_VALUE_RSP event.

The sequence of operations is complete when the Error Response is received and the Error Code is set to Attribute Not Found or when the End Group Handle in the Find By Type Value Response is 0xFFFF. Completion of this function is notified to upper layer using CYBLE_EVT_GATTC_ERROR_RSP event with the error code updated appropriately.

It is permitted to end the function early by calling the CyBle_GattcStopCmd() function if a desired primary service is found prior to discovery of all the primary services of the specified service UUID supported on the GATT Server.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.4.2 for more details on this sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *value* | Parameter is of type CYBLE_GATT_VALUE_T, where, |

1. 'value.val' should point to uint8 array containing the UUID to look for. UUID can be 16 or 128 bit.
2. 'value.len' should be set to 2 if the 16 bit UUID is to be found. The length should be set to 16 if 128 bit UUID is to be found.
3. 'value.actualLen' is an unused parameter and should be ignored as it is unused.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |

**CYBLE_API_RESULT_T CyBle_GattcFindIncludedServices (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATT_ATTR_HANDLE_RANGE_T ***range*)**

This function is used by the GATT Client to find Included Service declarations within a GATT Service to which it is connected. This is a non-blocking function.

Internally, multiple Read By Type Requests are sent to the peer device in response to which Read By Type Responses are received (CYBLE_EVT_GATTC_READ_BY_TYPE_RSP) and passed to the application layer.

When Read By Type Response data does not contain the service UUID, indicating the service UUID is a 128-bit UUID, the application layer can choose to get the service UUID by performing the following steps:

1. Stop ongoing GATT operation by invoking CyBle_GattcStopCmd()

2. Send Read Request by invoking the function CyBle_GattcReadCharacteristicValue() with the read request handle set to the attribute handle of the included service. Handle associated events.
3. Re-initiate CyBle_GattcFindIncludedServices function, setting the start handle to the attribute handle which is placed next to the one used in the above step.

It is permitted to end the function early if a desired included service is found prior to discovering all the included services of the specified service supported on the server by calling the CyBle_GattcStopCmd() function. If the CyBle_GattcStopCmd() function is not invoked, completion of this function is notified to the upper layer using CYBLE_EVT_GATTC_ERROR_RSP.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.5.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *range* | Pointer to the handle range of type CYBLE_GATT_ATTR_HANDLE_RANGE_T for which relationship discovery has to be performed |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcDiscoverAllCharacteristics (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATT_ATTR_HANDLE_RANGE_T *range*)

This function is used by the GATT Client to find all characteristic declarations within a service definition on a GATT Server connect to it when only the service handle range is known. This is a non-blocking function.

Internally, multiple Read By Type Requests are sent to the GATT Server in response to which Read By Type Responses are received. Each response results in the event CYBLE_EVT_GATTC_READ_BY_TYPE_RSP, which is passed to the application layer for handling.

It is permitted to end the function early by calling the CyBle_GattcStopCmd() function if a desired characteristic is found prior to discovering all the characteristics of the specified service supported on the GATT Server. Completion of this function is notified to upper layer using CYBLE_EVT_GATTC_ERROR_RSP event.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.6.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *range* | Parameter is of type CYBLE_GATT_ATTR_HANDLE_RANGE_T where: |

1. 'range.startHandle' can be set to the start handle of the desired primary service.
2. 'range.endHandle' can be set to the end handle of the desired primary service.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcDiscoverCharacteristicByUuid (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_BY_TYPE_REQ_T **readByTypeReqParam*)**

This function is used by the GATT Client to discover service characteristics on a GATT Server when only the service handle ranges are known and the characteristic UUID is known. This is a non-blocking function.

Internally, multiple Read By Type Requests are sent to the peer device in response to which Read By Type Responses are received. Each of these responses results in the event CYBLE_EVT_GATTC_READ_BY_TYPE_RSP, which is passed to the application layer for further processing.

It is permitted to end the function early by calling the CyBle_GattcStopCmd() function if a desired characteristic is found prior to discovering all the characteristics for the specified service supported on the GATT Server. Completion of this function is notified to upper layer using CYBLE_EVT_GATTC_ERROR_RSP event.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.6.2 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *readByTypeReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_BY_TYPE_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcDiscoverAllCharacteristicDescriptors (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_FIND_INFO_REQ_T **findInfoReqParam*)**

This function is used by the GATT Client to find all the characteristic descriptors. This is a non-blocking function.

Internally, multiple Find Information Requests are sent to the peer device in response to which Find Information Responses are received by the GATT Client. Each of these responses generate CYBLE_EVT_GATTC_FIND_INFO_RSP event at the GATT Client end which is propagated to the application layer for further processing.

It is permitted to end the function early by calling the CyBle_GattcStopCmd() function if desired Characteristic Descriptor is found prior to discovering all the characteristic descriptors of the specified characteristic. Completion of this function is notified to upper layer using CYBLE_EVT_GATTC_ERROR_RSP event.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.7.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *findInfoReqParam* | Pointer to a variable of type CYBLE_GATTC_FIND_INFO_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcReadCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_REQ_T *readReqParam*)

This function reads a Characteristic Value from a GATT Server when the GATT Client knows the Characteristic Value Handle. This is a non-blocking function.

Internally, Read Request is sent to the peer device in response to which Read Response is received. This response results in CYBLE_EVT_GATTC_READ_RSP event which is propagated to the application for handling the event data. An Error Response (CYBLE_EVT_GATTC_ERROR_RSP event at the GATT Client's end) is sent by the GATT Server in response to the Read Request on insufficient authentication or insufficient authorization or insufficient encryption key size is caused by the GATT Client, or if a read operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.8.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *readReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReadUsingCharacteristicUuid (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_BY_TYPE_REQ_T **readByTypeReqParam*)**

This function reads a Characteristic Value from the GATT Server when the GATT Client only knows the characteristic UUID and does not know the handle of the characteristic. This is a non-blocking function.

Internally, Read By Type Request is sent to the peer device in response to which Read By Type Response is received by the GATT Client. This results in CYBLE_EVT_GATTC_READ_BY_TYPE_RSP event, which is propagated to the application layer for further handling.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.8.2 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *readByTypeReqParam* | Parameter is of type CYBLE_GATTC_READ_BY_TYPE_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReadLongCharacteristicValues (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_BLOB_REQ_T **readBlobReqParam*)**

This function reads a Characteristic Value from the GATT Server when the GATT Client knows the Characteristic Value Handle and the length of the Characteristic Value is longer than can be sent in a single Read Response Attribute Protocol message. This is a non-blocking function.

Internally multiple Read Blob Requests are sent to the peer device in response to which Read Blob Responses are received. For each Read Blob Request, a Read Blob Response event is received (CYBLE_EVT_GATTC_READ_BLOB_RSP) with a portion of the Characteristic Value contained in the Part Attribute Value parameter. These events are propagated to the application layer for further processing. Each read blob response will return up to (GATT MTU-1) bytes of data. If the size of characteristic value field is an integral multiple of (GATT MTU-1) then the operation terminates with an error response event, where the error code is CYBLE_GATT_ERR_INVALID_OFFSET. If the size of the characteristic value field is not an integral multiple of (GATT MTU-1), the last read blob response will return data bytes which are less than (GATT MTU-1). The application needs to monitor these two conditions before proceeding with the initiation of any other GATT operation.

An Error Response event (CYBLE_EVT_GATTC_ERROR_RSP) is sent by the GATT Server in response to the Read Blob Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

If the Characteristic Value is not longer than (GATT MTU - 1), an Error Response with the Error Code set to Attribute Not Long is received by the GATT Client on the first Read Blob Request.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.8.3 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *readBlobReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_BLOB_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcReadMultipleCharacteristicValues (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_MULT_REQ_T **readMultiReqParam*)

This function reads multiple Characteristic Values from a GATT Server when the GATT Client knows the Characteristic Value Handles. This is a non-blocking function.

Internally, Read Multiple Request is sent to the peer device in response to which Read Multiple Response is received. This results in C YBLE_EVT_GATTC_READ_MULTI_RSP event, which is propagated to the application layer.

An Error Response event is sent by the server (CYBLE_EVT_GATTC_ERROR_RSP) in response to the Read Multiple Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on any of the Characteristic Values. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.8.4 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *readMultiReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_MULT_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcWriteWithoutResponse (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_WRITE_CMD_REQ_T ****writeCmdReqParam*****)**

This function writes a Characteristic Value to a GATT Server when the GATT Client knows the Characteristic Value Handle and the client does not need an acknowledgment that the write was successfully performed. This is a blocking function. No event is generated on calling this function.

Internally, Write Command is sent to the GATT Server and nothing is received in response from the GATT Server.

Refer Bluetooth 4.1 core specification, Volume 3, Part G, section 4.9.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writeCmdReqParam* | Pointer to a variable of type CYBLE_GATTC_WRITE_CMD_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcSignedWriteWithoutRsp (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_SIGNED_WRITE_CMD_REQ_T ****signedWriteWithoutRspParam*****)**

This function writes a Characteristic Value to a server when the client knows the Characteristic Value Handle and the ATT Bearer is not encrypted. This procedure shall only be used if the Characteristic Properties authenticated bit is enabled and the client and server device share a bond as defined in Bluetooth Spec4.1 [Vol. 3] Part C, Generic Access Profile.

This function only writes the first (GATT_MTU - 15) octets of an Attribute Value. This function cannot be used to write a long Attribute.

Internally, Signed Write Command is used. Refer Bluetooth Spec 4.1 Security Manager [Vol. 3] Part H, Section 2.4.5.

If the authenticated Characteristic Value that is written is the wrong size, has an invalid value as defined by the profile, or the signed value does not authenticate the client, then the write shall not succeed and no error shall be generated by the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *signedWriteWithoutRspParam* | Pointer to a variable of type CYBLE_GATTC_SIGNED_WRITE_CMD_REQ_T |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | BLE stack out of resource |

**CYBLE_API_RESULT_T CyBle_GattcWriteCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_WRITE_REQ_T **writeReqParam*)**

This function writes a Characteristic Value to a GATT Server when the GATT Client knows the Characteristic Value Handle. This is a non-blocking function.

Internally, Write Request is sent to the GATT Server in response to which Write Response is received. This results in the event CYBLE_EVT_GATTC_WRITE_RSP, which indicates that the write operation succeeded.

An Error Response event (CYBLE_EVT_GATTC_ERROR_RSP) is sent by the server in response to the Write Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.9.3 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writeReqParam* | Pointer to a variable of type CYBLE_GATTC_WRITE_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcWriteLongCharacteristicValues (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_PREP_WRITE_REQ_T **writePrepReqParam*)**

This function writes a Characteristic Value to a GATT Server when the GATT Client knows the Characteristic Value Handle but the length of the Characteristic Value is longer than GATT MTU size and cannot be sent in a single Write Request Attribute Protocol message. This is a non-blocking function.

Internally, multiple Prepare Write Requests are sent to the GATT Server in response to which Prepare Write Responses are received. No events are generated by the BLE Stack during these operations.

Prepare Write Requests are repeated until the complete Characteristic Value has been transferred to the GATT Server, after which an Execute Write Request is sent to the GATT Server to write the initially transferred value at the GATT Server's end. This generates CYBLE_EVT_GATTS_EXEC_WRITE_REQ at the GATT Server's end.

Once the GATT Server responds, CYBLE_EVT_GATTC_EXEC_WRITE_RSP event is generated at the GATT Client's end. The value associated with this event has to be checked by the application layer to confirm that the long write operation succeeded.

An Error Response event CYBLE_EVT_GATTC_ERROR_RSP is received by the GATT Client in response to the Prepare Write Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.9.4 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writePrepReqParam* | Pointer to a variable of type CYBLE_GATTC_PREP_WRITE_REQ_T, where 'writePrepReqParam->value.val' points to the actual data to be written. 'writePrepReqParam' and all associated variables need to be retained in memory by the calling application until the GATT Write Long Characteristic Value operation is completed successfully. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReliableWrites (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_PREP_WRITE_REQ_T **writePrepReqParam*, uint8 *numOfRequests*)**

This function writes a Characteristic Value to a GATT Server when the GATT Client knows the Characteristic Value Handle, and assurance is required that the correct Characteristic Value is going to be written by transferring the Characteristic Value to be written in both directions before the write is performed. This is a non-blocking function.

Internally, multiple Prepare Write Requests are sent to the GATT Server in response to which Prepare Write Responses are received. No events are generated by the BLE Stack during these operations.

Prepare Write Requests are repeated until the complete Characteristic Value has been transferred to the GATT Server, after which an Execute Write Request is sent to the GATT Server to write the initially transferred value at the GATT Server's end. This generates CYBLE_EVT_GATTS_EXEC_WRITE_REQ at the GATT Server's end.

Once the GATT Server responds, a CYBLE_EVT_GATTC_EXEC_WRITE_RSP event is generated at the GATT Client's end. The value associated with this event has to be checked by the application layer to confirm that the long write operation succeeded. An Error Response event CYBLE_EVT_GATTC_ERROR_RSP is received by the GATT Client in response to the Prepare Write Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.9.5 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writePrepReqParam* | Pointer to a variable of type CYBLE_GATTC_PREP_WRITE_REQ_T. Since more than one writes are performed as part of this function, the first array element of the array of type CYBLE_GATTC_PREP_WRITE_REQ_T, which contains the values to be written, has to be specified. 'writePrepReqParam' and all associated variables need to be retained in memory by the calling application until the GATT Reliable Write operation is completed successfully. |
| *numOfRequests* | Number of requests. That is, the count of array of structures of type CYBLE_GATTC_PREP_WRITE_REQ_T. Each array element represents a value and the attribute to which the value has to be written. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcConfirmation (CYBLE_CONN_HANDLE_T *connHandle*)

This function sends confirmation to the GATT Server on receiving Handle Value Indication event CYBLE_EVT_GATTC_HANDLE_VALUE_IND at the GATT Client's end. This is a non-blocking function.

This function call results in CYBLE_EVT_GATTS_HANDLE_VALUE_CNF event at the GATT Server's end.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.11.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReadCharacteristicDescriptors (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_REQ_T *readReqParam*)**

This function reads a characteristic descriptor from a GATT Server when the GATT Client knows the Attribute handle from the characteristic descriptor declaration. This is a non-blocking function.

Internally, Read Request is sent to the peer device in response to which Read Response is received. This response results in CYBLE_EVT_GATTC_READ_RSP event, which is propagated to the application for handling the event data.

An Error Response (CYBLE_EVT_GATTC_ERROR_RSP event at the GATT Client's end) is sent by the GATT Server in response to the Read Request on insufficient authentication or insufficient authorization or insufficient encryption key size is caused by the GATT Client, or if a read operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.12.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *readReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_REQ_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReadLongCharacteristicDescriptors (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_BLOB_REQ_T ***readBlobReqParam*)**

This function reads a characteristic descriptor from a GATT Server when the GATT Client knows the Attribute handle from the characteristic descriptor declaration and the length of the characteristic descriptor declaration is longer than what can be sent in a single Read Response Attribute Protocol message. This is a non-blocking function.

Internally multiple Read Blob Requests are sent to the peer device in response to which Read Blob Responses are received. For each Read Blob Request, a Read Blob Response event is received (CYBLE_EVT_GATTC_READ_BLOB_RSP) with a portion of the Characteristic Value contained in the Part Attribute Value parameter. These events are propagated to the application layer for further processing. Each read blob response will return up to (GATT MTU-1) bytes of data. If the size of characteristic descriptor field is an integral multiple of (GATT MTU-1) then the operation terminates with an error response event, where the error code is CYBLE_GATT_ERR_INVALID_OFFSET. If the size of the characteristic descriptor field is not an integral multiple of (GATT MTU-1), the last read blob response will return data bytes which are less than (GATT MTU-1). The application needs to monitor these two conditions before proceeding with the initiation of any other GATT operation.

An Error Response event (CYBLE_EVT_GATTC_ERROR_RSP) is sent by the GATT Server in response to the Read Blob Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol. If the Characteristic Value is not longer than (GATT MTU - 1) an Error

Response with the Error Code set to Attribute Not Long is received by the GATT Client on the first Read Blob Request.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.12.2 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *readBlobReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_BLOB_REQ_T |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

### CYBLE_API_RESULT_T CyBle_GattcWriteCharacteristicDescriptors (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_WRITE_REQ_T **writeReqParam*)

This function writes a characteristic descriptor value to a GATT Server when the GATT Client knows the characteristic descriptor handle. This is a non-blocking function.

Internally, Write Request is sent to the GATT Server in response to which Write Response is received. This results in the event CYBLE_EVT_GATTC_WRITE_RSP, which indicates that the write operation succeeded.

An Error Response event (CYBLE_EVT_GATTC_ERROR_RSP) is sent by the server in response to the Write Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.12.3 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writeReqParam* | Pointer to a variable of type CYBLE_GATTC_WRITE_REQ_T |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcWriteLongCharacteristicDescriptors (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_PREP_WRITE_REQ_T **writePrepReqParam*)**

This function writes a characteristic descriptor value to a GATT Server when the GATT Client knows the characteristic descriptor handle but the length of the characteristic descriptor value is longer than what can be sent in a single Write Request Attribute Protocol message. This is a non-blocking function.

Internally, multiple Prepare Write Requests are sent to the GATT Server in response to which Prepare Write Responses are received. No events are generated by the BLE Stack during these operations.

Prepare Write Requests are repeated until the complete Characteristic Descriptor Value has been transferred to the GATT Server, after which an Execute Write Request is sent to the GATT Server to write the initially transferred value at the GATT Server's end. This generates CYBLE_EVT_GATTS_EXEC_WRITE_REQ at the GATT Server's end.

Once the GATT Server responds, CYBLE_EVT_GATTC_EXEC_WRITE_RSP' event is generated at the GATT Client's end. The value associated with this event has to be checked by the application layer to confirm that the long write operation succeeded.

An Error Response event CYBLE_EVT_GATTC_ERROR_RSP is received by the GATT Client in response to the Prepare Write Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the Characteristic Value. The Error Code parameter is set as specified in the Attribute Protocol.

Refer Bluetooth 4.1 core specification, Volume 3, Part G, section 4.12.4 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *writePrepReqParam* | Pointer to a variable of type CYBLE_GATTC_PREP_WRITE_REQ_T, where 'writePrepReqParam->value.val' points to the actual data to be written. 'writePrepReqParam' and all associated variables need to be retained in memory by the calling application until the GATT Write Long Characteristic Descriptor operation is completed successfully. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcReadByTypeReq (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTC_READ_BY_TYPE_REQ_T **readByTypeReqParam*)**

This function allows the user to send Read by type request to peer server

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.5.1 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *readByTypeReqParam* | Pointer to a variable of type CYBLE_GATTC_READ_BY_TYPE_REQ_T, Where, the following needs to be set:<br>• readByTypeReqParam->range.startHandle<br>• readByTypeReqParam->range.endHandle<br>• readByTypeReqParam->uuidFormat (CYBLE_GATT_16_BIT_UUID_FORMAT or CYBLE_GATT_128_BIT_UUID_FORMAT)<br>• readByTypeReqParam->uuid.uuid16 or readByTypeReqParam->uuid.uuid128 based on the uuidFormat |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcSendExecuteWriteReq (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *flag*)**

This function allows the user to send execute write request to remote server. This function should be called if client has previously initiated long/reliable write operation and remote has send error response. Based on error response application may choose to execute all pending requests or cancel the request.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *flag* | Indicates whether Queued Write is to be executed (0x01) or canceled (0x00) |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattcDiscoverPrimaryServices (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATT_ATTR_HANDLE_RANGE_T **range*)**

This function is used by the GATT Client to discover the primary services as per the range provided on a GATT Server to which it is connected. This is a non-blocking function.

Internally, this function initiates multiple Read By Group Type Requests to the peer device in response to which it receives Read By Group Type Responses. Each Read By Group Type Response results in CYBLE_EVT_GATTC_READ_BY_GROUP_TYPE_RSP event, which is propagated to the application layer for handling.

Primary service discovery is complete when Error Response (CYBLE_EVT_GATTC_ERROR_RSP) is received and the Error Code is set to Attribute Not Found or when the End Group Handle in the Read by Group Type Response is 0xFFFF. Completion of this operation is notified to the upper layer(s) using CYBLE_EVT_GATTC_ERROR_RSP with error code updated appropriately.

It is permitted to end the above stated sequence of operations early if the desired primary service is found prior to discovering all the primary services on the GATT Server. This can be achieved by calling the CyBle_GattcStopCmd() function.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.4.1 for more details on this sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity of type CYBLE_CONN_HANDLE_T. |
| *range* | Parameter is of type CYBLE_GATT_ATTR_HANDLE_RANGE_T where,<br>1. 'range.startHandle' can be set to the start handle of the desired primary service.<br>2. 'range.endHandle' can be set to the end handle of the desired primary service. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

# GATT Server Functions

## Description

APIs unique to designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Gatts

## Functions

- CYBLE_API_RESULT_T CyBle_GattsReInitGattDb (void)

- CYBLE_API_RESULT_T CyBle_GattsDbRegister (const CYBLE_GATTS_DB_T *gattDbPtr, uint16 gattDbTotalEntries, uint16 gattDbMaxValue)
- CYBLE_GATT_ERR_CODE_T CyBle_GattsWriteAttributeValue (CYBLE_GATT_HANDLE_VALUE_PAIR_T *handleValuePair, uint16 offset, CYBLE_CONN_HANDLE_T *connHandle, uint8 flags)
- CYBLE_GATT_ERR_CODE_T CyBle_GattsReadAttributeValue (CYBLE_GATT_HANDLE_VALUE_PAIR_T *handleValuePair, CYBLE_CONN_HANDLE_T *connHandle, uint8 flags)
- CYBLE_GATT_ERR_CODE_T CyBle_GattsEnableAttribute (CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle)
- CYBLE_GATT_ERR_CODE_T CyBle_GattsDisableAttribute (CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle)
- CYBLE_GATT_ERR_CODE_T CyBle_GattsDbAuthorize (uint8 yesNo)
- CYBLE_API_RESULT_T CyBle_GattsNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTS_HANDLE_VALUE_NTF_T *ntfParam)
- CYBLE_API_RESULT_T CyBle_GattsIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTS_HANDLE_VALUE_IND_T *indParam)
- CYBLE_API_RESULT_T CyBle_GattsErrorRsp (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GATTS_ERR_PARAM_T *errRspParam)
- CYBLE_API_RESULT_T CyBle_GattsExchangeMtuRsp (CYBLE_CONN_HANDLE_T connHandle, uint16 mtu)
- void CyBle_GattsPrepWriteReqSupport (uint8 prepWriteSupport)
- CYBLE_API_RESULT_T CyBle_GattsWriteRsp (CYBLE_CONN_HANDLE_T connHandle)

## Function Documentation

### CYBLE_API_RESULT_T CyBle_GattsReInitGattDb (void )

Reinitializes the GATT database.

**Returns:**

CYBLE_API_RESULT_T: A function result states if it succeeded or failed with error codes:

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | GATT database was reinitialized successfully. |
| CYBLE_ERROR_INVALID_STATE | If the function is called in any state except CYBLE_STATE_DISCONNECTED. |
| CYBLE_ERROR_INVALID_PARAMETER | If the Database has zero entries or is a NULL pointer. |

### CYBLE_API_RESULT_T CyBle_GattsDbRegister (const CYBLE_GATTS_DB_T *gattDbPtr, uint16 gattDbTotalEntries, uint16 gattDbMaxValue)

This function registers the GATT database for the GATT Server. The GATT database stores all the attributes used by the GATT server, along with their permissions. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *gattDbPtr* | Pointer to the GATT database of type CYBLE_GATTS_DB_T. |
| *gattDbTotalEntries* | Total number of entries in the GATT database. |
| *gattDbMaxValue* | Maximum characteristic value length |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If the Database has zero entries or is a NULL pointer |

**CYBLE_GATT_ERR_CODE_T CyBle_GattsWriteAttributeValue (CYBLE_GATT_HANDLE_VALUE_PAIR_T **handleValuePair*, uint16 *offset*, CYBLE_CONN_HANDLE_T **connHandle*, uint8 *flags*)**

This function is used to write to the value field of the specified attribute in the GATT database of a GATT Server. This is a blocking function. No event is generated on calling this function.

If a peer device connected to the GATT Server initiates a write operation, this function is executed on the GATT Server. During such a call, the function checks for the attribute permissions (flags) before executing the write operation.

**Parameters:**

| | |
|---|---|
| *handleValuePair* | Pointer to handle value pair of type CYBLE_GATT_HANDLE_VALUE_PAIR_T.<br>• 'handleValuePair.attrHandle' is an input for which value has to be written.<br>• 'handleValuePair.value.len' is an input parameter for the length to be written.<br>• 'handleValuePair.value.val' is an input parameter for data buffer.<br>• 'handleValuePair.actualLen' has to be ignored as it is unused in this function. |
| *offset* | Offset at which the data (length in number of bytes) is written. |
| *connHandle* | Pointer to the attribute instance handle, of type CYBLE_CONN_HANDLE_T. |
| *flags* | Attribute permissions. Allowed values are,<br>• CYBLE_GATT_DB_LOCALLY_INITIATED<br>• CYBLE_GATT_DB_PEER_INITIATED |

**Returns:**

Return value is GATT Error code specified in 'CYBLE_GATT_ERR_CODE_T'

**CYBLE_GATT_ERR_CODE_T CyBle_GattsReadAttributeValue (CYBLE_GATT_HANDLE_VALUE_PAIR_T **handleValuePair*, CYBLE_CONN_HANDLE_T **connHandle*, uint8 *flags*)**

This function is used to read the value field of the specified attribute from the GATT database in a GATT Server. This is a blocking function. No event is generated on calling this function.

Peer initiated call to this function results in the function checking for attribute permissions before performing this operation.

**Parameters:**

| | |
|---|---|
| *handleValuePair* | Pointer to handle value pair of type CYBLE_GATT_HANDLE_VALUE_PAIR_T.<br>• 'handleValuePair.attrHandle' is an input for which value has to be read.<br>• 'handleValuePair.value.len' is an input parameter, the characteristic value is read based on length.<br>• 'handleValuePair.value.val' is an output parameter for data buffer. |

| | |
|---|---|
| | • 'handleValuePair.actualLen' has to be ignored as it is unused in this function. |
| *connHandle* | Pointer to the attribute instance handle, of type CYBLE_CONN_HANDLE_T. connHandle can be NULL if flags field is set to CYBLE_GATT_DB_LOCALLY_INITIATED. |
| *flags* | Attribute permissions. Allowed values are,<br>• CYBLE_GATT_DB_LOCALLY_INITIATED<br>• CYBLE_GATT_DB_PEER_INITIATED |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_GATT_ERR_NONE | On successful operation |
| CYBLE_GATT_ERR_INVALID_HANDLE | 'handleValuePair.attrHandle' is not valid |
| CYBLE_GATT_ERR_READ_NOT_PERMITTED | Read operation is not permitted on this attribute |
| CYBLE_GATT_ERR_UNLIKELY_ERROR | Invalid arguments passed |

### CYBLE_GATT_ERR_CODE_T CyBle_GattsEnableAttribute (CYBLE_GATT_DB_ATTR_HANDLE_T *attrHandle*)

This function enables the attribute entry for service or characteristic logical group in the GATT database registered in BLE Stack. This is a blocking function. No event is generated on calling this function.

This function returns an error if the attribute does not belong to any service or characteristic logical group. If the attribute entry is already enabled, then this function returns status CYBLE_GATT_ERR_NONE.

**Parameters:**

| | |
|---|---|
| *attrHandle* | Attribute handle of the registered GATT Database to enable particular attribute entry, of type CYBLE_GATT_DB_ATTR_HANDLE_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_GATT_ERR_NONE | On successful operation |
| CYBLE_GATT_ERR_INVALID_HANDLE | 'attrHandle' is not valid |

### CYBLE_GATT_ERR_CODE_T CyBle_GattsDisableAttribute (CYBLE_GATT_DB_ATTR_HANDLE_T *attrHandle*)

This function disables the attribute entry for service or characteristic logical group in the GATT database registered in the BLE Stack. This is a blocking function. No event is generated on calling this function.

This function returns error if the attribute does not belong to a service or a characteristic logical group. If attribute entry is already disabled then it returns CYBLE_GATT_ERR_NONE as status. All the attribute entries are enabled in GATT database during stack initialization.

**Parameters:**

| | |
|---|---|
| *attrHandle* | Attribute handle of the registered GATT Database to disable particular attribute entry, of type 'CYBLE_GATT_DB_ATTR_HANDLE_T' |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_GATT_ERR_NONE | On successful operation |
| CYBLE_GATT_ERR_INVALID_HANDLE | 'attrHandle' is not valid |

### CYBLE_GATT_ERR_CODE_T CyBle_GattsDbAuthorize (uint8 *yesNo*)

This Function sets or clears authorization permission for the GATT database

**Parameters:**

| | |
|---|---|
| *yesNo* | Setting this to '0' turns off authorization on the entire GATT database and all attributes marked as authorize will return authorization error. Setting this to any non-zero value will authorize the entire GATT database and all attributes marked as authorize can be read / written based on other allowed permissions. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_GATT_ERR_NONE | On successful operation |

### CYBLE_API_RESULT_T CyBle_GattsNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTS_HANDLE_VALUE_NTF_T **ntfParam*)

This function sends a notification to the peer device when the GATT Server is configured to notify a Characteristic Value to the GATT Client without expecting any Attribute Protocol layer acknowledgment that the notification was successfully received. This is a non-blocking function.

On enabling notification successfully for a specific attribute, if the GATT server has an updated value to be notified to the GATT Client, it sends out a 'Handle Value Notification' which results in CYBLE_EVT_GATTC_HANDLE_VALUE_NTF event at the GATT Client's end.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.10 for more details on notifications.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *ntfParam* | Pointer to structure of type CYBLE_GATTS_HANDLE_VALUE_NTF_T which is same as CYBLE_GATT_HANDLE_VALUE_PAIR_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted as BLE Stack is busy processing previous requests. The Error code is returned if the stack queue is full or for other reasons, the stack cannot process the operation. If stack busy event 'CYBLE_EVT_STACK_BUSY_STATUS' is triggered with status busy, calling this API function will trigger this error code. For details refer 'CYBLE_EVT_STACK_BUSY_STATUS' event |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattsIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTS_HANDLE_VALUE_IND_T **indParam*)**

This function sends an indication to the peer device when the GATT Server is configured to indicate a Characteristic Value to the GATT Client and expects an Attribute Protocol layer acknowledgment that the indication was successfully received. This is a non-blocking function.

On enabling indication successfully, if the GATT server has an updated value to be indicated to the GATT Client, it sends out a 'Handle Value Indication' which results in CYBLE_EVT_GATTC_HANDLE_VALUE_IND event at the GATT Client's end.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.11 for more details on Indications.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *indParam* | Pointer to structure of type CYBLE_GATTS_HANDLE_VALUE_IND_T which is same as CYBLE_GATT_HANDLE_VALUE_PAIR_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattsErrorRsp (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GATTS_ERR_PARAM_T **errRspParam*)**

This function sends an error response to the peer device. The Error Response is used to state that a given request cannot be performed, and to provide the reason as defined in 'CYBLE_GATT_ERR_CODE_T'. This is a non-blocking function.

Note that the 'Write Command' initiated by GATT Client does not generate an 'Error Response' from the GATT Server's end. The GATT Client gets CYBLE_EVT_GATTC_ERROR_RSP event on receiving error response.

Refer Bluetooth 4.1 core specification, Volume 3, Part F, section 3.4.1.1 for more details on Error Response operation.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *errRspParam* | Pointer to structure of type CYBLE_GATTS_ERR_PARAM_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

**CYBLE_API_RESULT_T CyBle_GattsExchangeMtuRsp (CYBLE_CONN_HANDLE_T *connHandle*, uint16 *mtu*)**

This function sends the GATT Server's GATT MTU size to the GATT Client. This function has to be invoked in response to an Exchange GATT MTU Request received from the GATT Client. The GATT Server's GATT MTU size should be greater than or equal to the default GATT MTU size (23 bytes). This is a non-blocking function.

The peer GATT Client receives CYBLE_EVT_GATTC_XCHNG_MTU_RSP event on executing this function on the GATT Server.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.3.1 for more details on exchange of GATT MTU.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *mtu* | Size of GATT MTU, of type uint16 |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If 'l2capPsm' is 0 |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | Cannot register more than one PSM |
| CYBLE_ERROR_L2CAP_PSM_WRONG_ENCODING | PSM value must be an odd number and the Most Significant Byte must have Least Significant Bit value set to '0'. If PSM does not follow this guideline, this return code is generated. |
| CYBLE_ERROR_L2CAP_PSM_ALREADY_REGISTERED | PSM already Registered |

**void CyBle_GattsPrepWriteReqSupport (uint8 *prepWriteSupport*)**

This API function needs to be called after getting CYBLE_EVT_GATTS_PREP_WRITE_REQ event from the BLE Stack to support prepare write request operation. This API function should be called only once during one Long/reliable write session. This needs to be called from the same event call back context. This is a non-blocking function.

On receiving CYBLE_EVT_GATTS_PREP_WRITE_REQ, returning from the event handler without calling this function will result in prepare write response being sent to the peer device rejecting the prepare write operation. CYBLE_GATT_ERR_REQUEST_NOT_SUPPORTED error code will be sent to client.

**Parameters:**

| | |
|---|---|
| *prepWriteSupport* | If prepare write operation is supported by the application then the application layer should set this variable to CYBLE_GATTS_PREP_WRITE_SUPPORT. Any other value will result in the device rejecting the prepare write operation. Allowed values for this parameter<br>• CYBLE_GATTS_PREP_WRITE_SUPPORT<br>• CYBLE_GATTS_PREP_WRITE_NOT_SUPPORT |

**Returns:**

None

**CYBLE_API_RESULT_T CyBle_GattsWriteRsp (CYBLE_CONN_HANDLE_T *connHandle*)**

This function sends a Write Response from a GATT Server to the GATT Client. This is a non-blocking function. This function has to be invoked in response to a valid Write Request event from the GATT Client (CYBLE_EVT_GATTS_WRITE_REQ) to acknowledge that the attribute has been successfully written.

The Write Response has to be sent after the attribute value is written or saved by the GATT Server. Write Response results in CYBLE_EVT_GATTC_WRITE_RSP event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |

# GATT Definitions and Data Structures

## Description

Contains the GATT specific definitions and data structures used in the GATT APIs.

## Data Structures

- struct CYBLE_DISC_SRVC_INFO_T
- struct CYBLE_DISC_SRVC128_INFO_T
- struct CYBLE_DISC_INCL_INFO_T
- struct CYBLE_DISC_CHAR_INFO_T
- struct CYBLE_SRVR_CHAR_INFO_T
- struct CYBLE_DISC_DESCR_INFO_T
- struct CYBLE_GATTS_T
- struct CYBLE_GATTC_T
- struct CY_BLE_FLASH_STORAGE
- struct CYBLE_GATT_VALUE_T
- struct CYBLE_GATT_HANDLE_VALUE_PAIR_T
- struct CYBLE_GATT_ATTR_HANDLE_RANGE_T
- struct CYBLE_GATT_XCHG_MTU_PARAM_T
- struct CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T
- struct CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T
- struct CYBLE_GATTC_ERR_RSP_PARAM_T
- struct CYBLE_GATTC_READ_BY_TYPE_REQ_T
- struct CYBLE_GATTC_READ_BLOB_REQ_T
- struct CYBLE_GATTC_HANDLE_LIST_T
- struct CYBLE_GATTC_READ_RSP_PARAM_T
- struct CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T
- struct CYBLE_GATTC_GRP_ATTR_DATA_LIST_T
- struct CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T
- struct CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T
- struct CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T
- struct CYBLE_GATTC_FIND_INFO_RSP_PARAM_T
- struct CYBLE_GATTC_FIND_BY_TYPE_VALUE_REQ_T
- struct CYBLE_GATTC_EXEC_WRITE_RSP_T
- struct CYBLE_GATTS_ATT_GEN_VAL_LEN_T
- struct CYBLE_GATTS_ATT_PACK_VAL_LEN_T
- union CYBLE_GATTS_ATT_VALUE_T
- struct CYBLE_GATTS_DB_T
- struct CYBLE_GATTS_ERR_PARAM_T
- struct CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T
- struct CYBLE_GATTS_EXEC_WRITE_REQ_T
- struct CYBLE_GATTS_WRITE_REQ_PARAM_T
- struct CYBLE_GATTS_CHAR_VAL_READ_REQ_T

## Typedefs

- typedef uint16 CYBLE_GATT_DB_ATTR_HANDLE_T
- typedef CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_GATTC_FIND_INFO_REQ_T
- typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_SIGNED_WRITE_CMD_REQ_T
- typedef CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTC_READ_REQ_T
- typedef CYBLE_GATTC_HANDLE_LIST_T CYBLE_GATTC_READ_MULT_REQ_T
- typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_WRITE_CMD_REQ_T
- typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_WRITE_REQ_T
- typedef CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T CYBLE_GATTC_PREP_WRITE_REQ_T
- typedef CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T CYBLE_GATTC_HANDLE_VALUE_IND_PARAM_T

- typedef CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T CYBLE_GATTC_READ_BY_TYPE_RSP_PARAM_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_EXT_PRPRTY_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_USER_DESCRIPTION_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_CLIENT_CHAR_CONFIG_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_SERVER_CHAR_CONFIG_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_PRESENT_FMT_T
- typedef CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_AGGREGATE_FMT_T
- typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTS_HANDLE_VALUE_NTF_T
- typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTS_HANDLE_VALUE_IND_T
- typedef CYBLE_GATT_VALUE_T CYBLE_GATTS_READ_RSP_PARAM_T
- typedef CYBLE_GATTS_WRITE_REQ_PARAM_T CYBLE_GATTS_WRITE_CMD_REQ_PARAM_T
- typedef CYBLE_GATTS_WRITE_REQ_PARAM_T CYBLE_GATTS_SIGNED_WRITE_CMD_REQ_PARAM_T
- typedef CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T CYBLE_GATTS_PREP_WRITE_RSP_PARAM_T

## Enumerations

- enum CYBLE_GATT_PDU_T
- enum CYBLE_GATT_ERR_CODE_T

## Data Structure Documentation

### struct CYBLE_DISC_SRVC_INFO_T

**Data Fields**

- CYBLE_GATT_ATTR_HANDLE_RANGE_T range
- uint16 uuid

**Field Documentation**

**CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_DISC_SRVC_INFO_T::range**
Handle range of the request

**uint16 CYBLE_DISC_SRVC_INFO_T::uuid**
16-bit UUID

### struct CYBLE_DISC_SRVC128_INFO_T

**Data Fields**

- CYBLE_GATT_ATTR_HANDLE_RANGE_T range
- CYBLE_UUID_T uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_DISC_SRVC128_INFO_T::range**
Handle range of the request

**CYBLE_UUID_T CYBLE_DISC_SRVC128_INFO_T::uuid**
128-bit UUID

**uint8 CYBLE_DISC_SRVC128_INFO_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**struct CYBLE_DISC_INCL_INFO_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T inclDefHandle
- CYBLE_GATT_ATTR_HANDLE_RANGE_T inclHandleRange
- CYBLE_UUID_T uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISC_INCL_INFO_T::inclDefHandle**
Included definition handle

**CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_DISC_INCL_INFO_T::inclHandleRange**
Included declaration handle range

**CYBLE_UUID_T CYBLE_DISC_INCL_INFO_T::uuid**
Included UUID

**uint8 CYBLE_DISC_INCL_INFO_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**struct CYBLE_DISC_CHAR_INFO_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charDeclHandle
- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_UUID_T uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISC_CHAR_INFO_T::charDeclHandle**
Handle for characteristic declaration

**uint8 CYBLE_DISC_CHAR_INFO_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISC_CHAR_INFO_T::valueHandle**
Handle to server database attribute value entry

**CYBLE_UUID_T CYBLE_DISC_CHAR_INFO_T::uuid**
Characteristic UUID

**uint8 CYBLE_DISC_CHAR_INFO_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**struct CYBLE_SRVR_CHAR_INFO_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle

**Field Documentation**

**uint8 CYBLE_SRVR_CHAR_INFO_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SRVR_CHAR_INFO_T::valueHandle**
Handle of server database attribute value entry

**struct CYBLE_DISC_DESCR_INFO_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle
- CYBLE_UUID_T uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_DISC_DESCR_INFO_T::connHandle**
Handle to server database attribute entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISC_DESCR_INFO_T::descrHandle**
Descriptor handle

**CYBLE_UUID_T CYBLE_DISC_DESCR_INFO_T::uuid**
Descriptor UUID

**uint8 CYBLE_DISC_DESCR_INFO_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**struct CYBLE_GATTS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T serviceChangedHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_T::serviceHandle**
Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_T::serviceChangedHandle**
Handle of the Service Changed characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_T::cccdHandle**
Client Characteristic Configuration descriptor handle

**struct CYBLE_GATTC_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T serviceChanged
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_GATTC_T::serviceChanged**
Handle of the Service Changed characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTC_T::cccdHandle**
Client Characteristic Configuration descriptor handle

**struct CY_BLE_FLASH_STORAGE**

**Data Fields**

- uint8 stackFlashptr [((0x09u+(0x9Cu *0x04u)))]
- uint8 attValuesCCCDFlashMemory [0x04u+1u][(1u)]
- uint8 cccdCount
- uint8 boundedDevCount

**Field Documentation**

**uint8 CY_BLE_FLASH_STORAGE::stackFlashptr[((0x09u+(0x9Cu *0x04u)))]**
Stack internal bonding data

**uint8 CY_BLE_FLASH_STORAGE::attValuesCCCDFlashMemory[0x04u+1u][(1u)]**
CCCD values

**uint8 CY_BLE_FLASH_STORAGE::cccdCount**
Number of CCCD

**uint8 CY_BLE_FLASH_STORAGE::boundedDevCount**
Number of bonded devices

**struct CYBLE_GATT_VALUE_T**

**Data Fields**

- uint8 * val
- uint16 len
- uint16 actualLen

**Field Documentation**

**uint8* CYBLE_GATT_VALUE_T::val**
Pointer to the value to be packed

**uint16 CYBLE_GATT_VALUE_T::len**
Length of Value to be packed

**uint16 CYBLE_GATT_VALUE_T::actualLen**
Out Parameter Indicating Actual Length Packed and sent over the air. Actual length can be less than or equal to the 'len' parameter value. This provides information to application that what is the actual length of data that is transmitted over the air. Each GATT procedures defines different length of data that can be transmitted over the air. If application sends more than that, all data may not transmitted over air.

**struct CYBLE_GATT_HANDLE_VALUE_PAIR_T**

**Data Fields**

- CYBLE_GATT_VALUE_T value
- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle

**Field Documentation**

**CYBLE_GATT_VALUE_T CYBLE_GATT_HANDLE_VALUE_PAIR_T::value**
Attribute Value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATT_HANDLE_VALUE_PAIR_T::attrHandle**
Attribute Handle of GATT DB

**struct CYBLE_GATT_ATTR_HANDLE_RANGE_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T startHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATT_ATTR_HANDLE_RANGE_T::startHandle**
Start Handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATT_ATTR_HANDLE_RANGE_T::endHandle**
End Handle

**struct CYBLE_GATT_XCHG_MTU_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint16 mtu

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATT_XCHG_MTU_PARAM_T::connHandle**

Connection handle

**uint16 CYBLE_GATT_XCHG_MTU_PARAM_T::mtu**

Client/Server Rx/Tx GATT MTU Size

**struct CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T**

**Data Fields**

- CYBLE_GATT_HANDLE_VALUE_PAIR_T handleValuePair
- uint16 offset

**Field Documentation**

**CYBLE_GATT_HANDLE_VALUE_PAIR_T**
**CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T::handleValuePair**

Attribute Handle & Value to be Written

**uint16 CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T::offset**

Offset at which Write is to be performed

**struct CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T**

**Data Fields**

- uint8 * queueBuffer
- uint16 totalAttrValueLength
- uint16 prepareWriteQueueSize

**Field Documentation**

**uint8* CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T::queueBuffer**

buffer to which prepare write queue request will be stored buffer can be calculated as - total buffer = totalAttrValueLength

- prepareWriteQueueSize * sizeof (CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T)

**uint16 CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T::totalAttrValueLength**

length of attribute value. This value can be max attribute value length or summation of values lengths which supports long write. Value should be multiple of 32 bit unsigned integer

**uint16 CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T::prepareWriteQueueSize**

Size of prepareWriteQueue buffer. Application may choose to decide the size base on (totalAttrValueLength or Max attribute length or summation of values lengths which supports long write) /(negotiated or default MTU size - 5) In case of reliable write, queue depth should at least be equal to number of handles which has reliable write support

**struct CYBLE_GATTC_ERR_RSP_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_PDU_T opCode
- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle
- CYBLE_GATT_ERR_CODE_T errorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_ERR_RSP_PARAM_T::connHandle**
Connection handle

**CYBLE_GATT_PDU_T CYBLE_GATTC_ERR_RSP_PARAM_T::opCode**
Opcode which has resulted in Error

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTC_ERR_RSP_PARAM_T::attrHandle**
Attribute Handle in which error is generated

**CYBLE_GATT_ERR_CODE_T CYBLE_GATTC_ERR_RSP_PARAM_T::errorCode**
Error Code describing cause of error

**struct CYBLE_GATTC_READ_BY_TYPE_REQ_T**

**Data Fields**

- CYBLE_GATT_ATTR_HANDLE_RANGE_T range
- CYBLE_UUID_T uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_GATTC_READ_BY_TYPE_REQ_T::range**
Handle Range

**CYBLE_UUID_T CYBLE_GATTC_READ_BY_TYPE_REQ_T::uuid**
GATT UUID type

**uint8 CYBLE_GATTC_READ_BY_TYPE_REQ_T::uuidFormat**
Format indicating, 16 bit or 128 bit UUIDs For 16bits UUID format - CYBLE_GATT_16_BIT_UUID_FORMAT (0x01) For 128bits UUID format - CYBLE_GATT_128_BIT_UUID_FORMAT (0x02)

**struct CYBLE_GATTC_READ_BLOB_REQ_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle
- uint16 offset

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTC_READ_BLOB_REQ_T::attrHandle**
Handle on which Read Blob is requested

**uint16 CYBLE_GATTC_READ_BLOB_REQ_T::offset**
Value Offset from which the Read is Requested

**struct CYBLE_GATTC_HANDLE_LIST_T**

**Data Fields**

- uint16 * handleList
- uint16 listCount
- uint16 actualCount

**Field Documentation**

**uint16* CYBLE_GATTC_HANDLE_LIST_T::handleList**
Handle list where the UUID with value Indicated is found

**uint16 CYBLE_GATTC_HANDLE_LIST_T::listCount**
Number of Handles in the list

**uint16 CYBLE_GATTC_HANDLE_LIST_T::actualCount**
Actual Number of Handles Packed. This is a output parameter

**struct CYBLE_GATTC_READ_RSP_PARAM_T**

**Data Fields**
- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_VALUE_T value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_READ_RSP_PARAM_T::connHandle**
Connection handle

**CYBLE_GATT_VALUE_T CYBLE_GATTC_READ_RSP_PARAM_T::value**
Attribute Value

**struct CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T**

**Data Fields**
- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_HANDLE_VALUE_PAIR_T handleValPair

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T::connHandle**
Connection handle

**CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T::handleValPair**
handle value pair, actual length files needs to be ignored

**struct CYBLE_GATTC_GRP_ATTR_DATA_LIST_T**

**Data Fields**
- uint8 * attrValue
- uint16 length
- uint16 attrLen

**Field Documentation**

**uint8* CYBLE_GATTC_GRP_ATTR_DATA_LIST_T::attrValue**
attribute handle value pair

**uint16 CYBLE_GATTC_GRP_ATTR_DATA_LIST_T::length**
Length of each Attribute Data Element including the Handle Range

**uint16 CYBLE_GATTC_GRP_ATTR_DATA_LIST_T::attrLen**
Total Length of Attribute Data

**struct CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T**

**Data Fields**
- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATTC_GRP_ATTR_DATA_LIST_T attrData

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T::connHandle**
Connection handle

**CYBLE_GATTC_GRP_ATTR_DATA_LIST_T CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T::attrData**
Group attribute data list

**struct CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_ATTR_HANDLE_RANGE_T * range
- uint8 count

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T::connHandle**
Connection handle

**CYBLE_GATT_ATTR_HANDLE_RANGE_T* CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T::range**
Handle Range List

**uint8 CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T::count**
Size of List

**struct CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T**

**Data Fields**

- uint8 * list
- uint16 byteCount

**Field Documentation**

**uint8* CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T::list**
Handle - UUID Pair list This is a packed byte stream, hence it needs to be unpacked and decoded.

**uint16 CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T::byteCount**
Number of elements in the list in bytes

**struct CYBLE_GATTC_FIND_INFO_RSP_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T handleValueList
- uint8 uuidFormat

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_FIND_INFO_RSP_PARAM_T::connHandle**
Connection handle

**CYBLE_GATTC_HANDLE_UUID_LIST_PARAM_T**
**CYBLE_GATTC_FIND_INFO_RSP_PARAM_T::handleValueList**
Handle Value List

**uint8 CYBLE_GATTC_FIND_INFO_RSP_PARAM_T::uuidFormat**
Format indicating, 16 bit (0x01) or 128 bit (0x02) UUIDs

**struct CYBLE_GATTC_FIND_BY_TYPE_VALUE_REQ_T**

**Data Fields**

- CYBLE_GATT_VALUE_T value
- CYBLE_GATT_ATTR_HANDLE_RANGE_T range
- CYBLE_UUID16 uuid

**Field Documentation**

**CYBLE_GATT_VALUE_T CYBLE_GATTC_FIND_BY_TYPE_VALUE_REQ_T::value**
Attribute Value to Find

**CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_GATTC_FIND_BY_TYPE_VALUE_REQ_T::range**
Handle Range - Start and End Handle

**CYBLE_UUID16 CYBLE_GATTC_FIND_BY_TYPE_VALUE_REQ_T::uuid**
16-bit UUID to Find

### struct CYBLE_GATTC_EXEC_WRITE_RSP_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint8 result

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTC_EXEC_WRITE_RSP_T::connHandle**
Connection handle

**uint8 CYBLE_GATTC_EXEC_WRITE_RSP_T::result**
Result of the execute write request

### struct CYBLE_GATTS_ATT_GEN_VAL_LEN_T

**Data Fields**

- uint16 actualLength
- void * attGenericVal

**Field Documentation**

**uint16 CYBLE_GATTS_ATT_GEN_VAL_LEN_T::actualLength**
Length in number of bytes for attGenericVal

**void* CYBLE_GATTS_ATT_GEN_VAL_LEN_T::attGenericVal**
Buffer to the store generic characteristic value based on length or complete UUID value if the attribute is of type 128-bit UUID and 32-bit UUID type.

### struct CYBLE_GATTS_ATT_PACK_VAL_LEN_T

**Data Fields**

- uint16 maxAttrLength
- CYBLE_GATTS_ATT_GEN_VAL_LEN_T * attGenericValLen

**Field Documentation**

**uint16 CYBLE_GATTS_ATT_PACK_VAL_LEN_T::maxAttrLength**
Length in number of bytes for attGenericVal

**CYBLE_GATTS_ATT_GEN_VAL_LEN_T* CYBLE_GATTS_ATT_PACK_VAL_LEN_T::attGenericValLen**
Buffer to the store generic characteristic value based on length or complete UUID value if the attribute is of type 128-bit UUID and 32-bit UUID type.

### union CYBLE_GATTS_ATT_VALUE_T

**Data Fields**

- CYBLE_GATTS_ATT_PACK_VAL_LEN_T attFormatValue
- uint16 attValueUuid

**Field Documentation**

**CYBLE_GATTS_ATT_PACK_VAL_LEN_T CYBLE_GATTS_ATT_VALUE_T::attFormatValue**
Buffer containing 32-bit or 128-bit UUID values for Service and Characteristic declaration. Attribute format structure: if entry is for characteristic value format, then it has the "attribute format value" of pointer type to represent generic structure to cater wide formats of available list of characteristic formats.

**uint16 CYBLE_GATTS_ATT_VALUE_T::attValueUuid**

Attribute UUID value

**struct CYBLE_GATTS_DB_T**

**Data Fields**

- uint16 attHandle
- uint16 attType
- uint32 permission
- uint16 attEndHandle
- CYBLE_GATTS_ATT_VALUE_T attValue

**Field Documentation**

**uint16 CYBLE_GATTS_DB_T::attHandle**

Start Handle: Act as an index for querying BLE GATT database

**uint16 CYBLE_GATTS_DB_T::attType**

UUID: 16 bit UUID type for an attribute entry, for 32 bit and 128 bit UUIDs the last 16 bits should be stored in this entry GATT DB access layer shall retrieve complete 128 bit UUID from CYBLE_GATTS_ATT_GENERIC_VAL_T structure.

**uint32 CYBLE_GATTS_DB_T::permission**

The permission bits are clubbed in to a 32-bit field. These 32-bits can be grouped in to 4 bytes. The lowest significant byte is byte 0 (B0) and the most significant byte is byte 3 (B3). The bytes where the permissions have been grouped is as given below. Attribute permissions for read (B0) Attribute permissions for write (B1) Characteristic properties (B2) Implementation specific permission (B3)

**uint16 CYBLE_GATTS_DB_T::attEndHandle**

Attribute end handle, indicating logical boundary of given attribute.

**CYBLE_GATTS_ATT_VALUE_T CYBLE_GATTS_DB_T::attValue**

Attribute value format, it can be one of following: uint16 16bit - UUID for 16bit service & characteristic declaration CYBLE_GATTS_ATT_GENERIC_VAL_T attFormatValue - Buffer containing 32 bit or 128 bit UUID values for service & characteristic declaration CYBLE_GATTS_ATT_GENERIC_VAL_T attFormatValue - Buffer containing generic char definition value, or generic descriptor values

**struct CYBLE_GATTS_ERR_PARAM_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle
- uint8 opcode
- CYBLE_GATT_ERR_CODE_T errorCode

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_ERR_PARAM_T::attrHandle**

Handle in which error is generated

**uint8 CYBLE_GATTS_ERR_PARAM_T::opcode**

Opcode which has resulted in Error Information on ATT/GATT opcodes is available in the Bluetooth specification.

**CYBLE_GATT_ERR_CODE_T CYBLE_GATTS_ERR_PARAM_T::errorCode**

Error Code describing cause of error

**struct CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T * baseAddr

- uint8 currentPrepWriteReqCount
- uint8 gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T::connHandle**

Connection handle

**CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T* CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T::baseAddr**

Base address of the queue where data is queued, Queue is of type CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T. Each baseAddr[currentPrepWriteReqCount-1].handleValuePair.value.val provides the current data and baseAddr[0].handleValuePair.value.val provides the base address of the data buffer where full value will be stored. Application can calculate the total length based on each each array element. i.e total length up current request = baseAddr[0].handleValuePair.value.len+ ....+baseAddr[currentPrepWriteReqCount-1].handleValuePair.value.len

**uint8 CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T::currentPrepWriteReqCount**

Current count of prepare request from remote. This parameter can be used to access the data from 'baseAddr[]'. Array index will range from 0 to currentPrepWriteReqCount - 1

**uint8 CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T::gattErrorCode**

Application provide GATT error code for the procedure. This is an o/p parameter

**struct CYBLE_GATTS_EXEC_WRITE_REQ_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T * baseAddr
- uint8 prepWriteReqCount
- uint8 execWriteFlag
- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle
- uint8 gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTS_EXEC_WRITE_REQ_T::connHandle**

Connection handle

**CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T* CYBLE_GATTS_EXEC_WRITE_REQ_T::baseAddr**

Base address of the queue where data is queued. Queue is of type CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T. baseAddr[0].handleValuePair.value.val provides the base address of the total data stored in prepare write queue internally by stack. Application can calculate the total length based on each each array element. i.e total length = baseAddr[0].handleValuePair.value.len+ ....+baseAddr[prepWriteReqCount-1].handleValuePair.value.len

**uint8 CYBLE_GATTS_EXEC_WRITE_REQ_T::prepWriteReqCount**

Total count of prepare request from remote. This parameter can be used to access the data from 'baseAddr[]'. array index will range from 0 to prepWriteReqCount - 1

**uint8 CYBLE_GATTS_EXEC_WRITE_REQ_T::execWriteFlag**

Execute write flag received from remote

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_EXEC_WRITE_REQ_T::attrHandle**

Attribute Handle at which error occurred. This is an o/p param

**uint8 CYBLE_GATTS_EXEC_WRITE_REQ_T::gattErrorCode**

Application provide GATT error code for the procedure. This is an o/p param

**struct CYBLE_GATTS_WRITE_REQ_PARAM_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_HANDLE_VALUE_PAIR_T handleValPair

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTS_WRITE_REQ_PARAM_T::connHandle**

Connection handle

**CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTS_WRITE_REQ_PARAM_T::handleValPair**

handle value pair

**struct CYBLE_GATTS_CHAR_VAL_READ_REQ_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GATTS_CHAR_VAL_READ_REQ_T::connHandle**

Connection handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTS_CHAR_VAL_READ_REQ_T::attrHandle**

Attribute Handle

**CYBLE_GATT_ERR_CODE_T CYBLE_GATTS_CHAR_VAL_READ_REQ_T::gattErrorCode**

Output Param: Profile/Service specific error code, profile or application need to change this to service specific error based on service/profile requirements.

## Typedef Documentation

**typedef uint16 CYBLE_GATT_DB_ATTR_HANDLE_T**

GATT BD Attribute Handle Type

**typedef CYBLE_GATT_ATTR_HANDLE_RANGE_T CYBLE_GATTC_FIND_INFO_REQ_T**

GATT find info request to be sent to Server

**typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_SIGNED_WRITE_CMD_REQ_T**

Signed Write command request to be sent to Server

**typedef CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GATTC_READ_REQ_T**

Read request to be sent to Server

**typedef CYBLE_GATTC_HANDLE_LIST_T CYBLE_GATTC_READ_MULT_REQ_T**

Read multiple request to be sent to Server

**typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_WRITE_CMD_REQ_T**

Write command request to be sent to Server

**typedef CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTC_WRITE_REQ_T**

Write request to be sent to Server

**typedef** CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T CYBLE_GATTC_PREP_WRITE_REQ_T

Prepare write request to be sent to Server

**typedef** CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T CYBLE_GATTC_HANDLE_VALUE_IND_PARAM_T

GATT handle value indication parameter received from server type

**typedef** CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T CYBLE_GATTC_READ_BY_TYPE_RSP_PARAM_T

GATT read by type response received from server

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_EXT_PRPRTY_T

Characteristic Extended Property

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_USER_DESCRIPTION_T

Characteristic User Description

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_CLIENT_CHAR_CONFIG_T

Client Characteristic Configuration

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_SERVER_CHAR_CONFIG_T

Server Characteristic Configuration

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_PRESENT_FMT_T

Characteristic Presentation Format

**typedef** CYBLE_GATTS_ATT_VALUE_T CYBLE_CHAR_AGGREGATE_FMT_T

Characteristic Aggregate Format

**typedef** CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTS_HANDLE_VALUE_NTF_T

Handle value notification data to be sent to Client

**typedef** CYBLE_GATT_HANDLE_VALUE_PAIR_T CYBLE_GATTS_HANDLE_VALUE_IND_T

GATT handle value indication parameter type

**typedef** CYBLE_GATT_VALUE_T CYBLE_GATTS_READ_RSP_PARAM_T

Read response parameter to be sent to Client

**typedef** CYBLE_GATTS_WRITE_REQ_PARAM_T CYBLE_GATTS_WRITE_CMD_REQ_PARAM_T

Write command request parameter received from Client

**typedef** CYBLE_GATTS_WRITE_REQ_PARAM_T CYBLE_GATTS_SIGNED_WRITE_CMD_REQ_PARAM_T

Signed Write command request parameter received from Client

**typedef** CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T CYBLE_GATTS_PREP_WRITE_RSP_PARAM_T

Prepare write response parameter to be sent to Client

## Enumeration Type Documentation

**enum CYBLE_GATT_PDU_T**

Opcode which has resulted in error

**Enumerator**

***CYBLE_GATT_ERROR_RSP*** Error Response PDU

***CYBLE_GATT_XCNHG_MTU_REQ*** Exchange GATT MTU Request PDU

***CYBLE_GATT_XCHNG_MTU_RSP*** Exchange GATT MTU Response PDU

***CYBLE_GATT_FIND_INFO_REQ*** Find Information Request PDU

***CYBLE_GATT_FIND_INFO_RSP*** Find Information Response PDU

***CYBLE_GATT_FIND_BY_TYPE_VALUE_REQ*** Find By Type Value Request PDU

***CYBLE_GATT_FIND_BY_TYPE_VALUE_RSP*** Find By Type Value Response PDU

***CYBLE_GATT_READ_BY_TYPE_REQ*** Read By Type Request PDU

***CYBLE_GATT_READ_BY_TYPE_RSP*** Read By Type Response PDU

***CYBLE_GATT_READ_REQ*** Read Request PDU

***CYBLE_GATT_READ_RSP*** Read Response PDU

***CYBLE_GATT_READ_BLOB_REQ*** Read Blob Request PDU

***CYBLE_GATT_READ_BLOB_RSP*** Read Blob Response PDU

***CYBLE_GATT_READ_MULTIPLE_REQ*** Read Multiple Request PDU

***CYBLE_GATT_READ_MULTIPLE_RSP*** Read Multiple Response PDU

***CYBLE_GATT_READ_BY_GROUP_REQ*** Read Group Type Request PDU

***CYBLE_GATT_READ_BY_GROUP_RSP*** Read Group Type Response PDU

***CYBLE_GATT_WRITE_REQ*** Write Request PDU

***CYBLE_GATT_WRITE_RSP*** Write Response PDU

***CYBLE_GATT_WRITE_CMD*** Write Command PDU

***CYBLE_GATT_PREPARE_WRITE_REQ*** Prepare Write Request PDU

***CYBLE_GATT_PREPARE_WRITE_RSP*** Prepare Write Response PDU

***CYBLE_GATT_EXECUTE_WRITE_REQ*** Execute Write Request PDU

***CYBLE_GATT_EXECUTE_WRITE_RSP*** Execute Write Response PDU

***CYBLE_GATT_HANDLE_VALUE_NTF*** Handle Value Notification PDU

***CYBLE_GATT_HANDLE_VALUE_IND*** Handle Value Indication PDU

***CYBLE_GATT_HANDLE_VALUE_CNF*** Handle Value Confirmation PDU

***CYBLE_GATT_SIGNED_WRITE_CMD*** Signed Write Command PDU

***CYBLE_GATT_UNKNOWN_PDU_IND*** Unknown or Unhandled PDU

**enum CYBLE_GATT_ERR_CODE_T**

GATT profile error codes

**Enumerator**

***CYBLE_GATT_ERR_NONE*** No Error

***CYBLE_GATT_ERR_INVALID_HANDLE*** Invalid Handle error code is used in the case when the ATT handle in the ATT request PDU is invalid.

***CYBLE_GATT_ERR_READ_NOT_PERMITTED*** Read Not Permitted error code is used in the case when the permission to read the value of an ATT handle is not permitted on the ATT server.

***CYBLE_GATT_ERR_WRITE_NOT_PERMITTED*** Write Not Permitted error code is used in the case when the permission to write the value of an ATT handle is not permitted on the ATT server.

***CYBLE_GATT_ERR_INVALID_PDU*** Invalid PDU error code is used in the case when the format of the PDU sent from the ATT Client is incorrect.

***CYBLE_GATT_ERR_INSUFFICIENT_AUTHENTICATION*** Insufficient Authentication error code is used in the case when an access to a handle is attempted on a un-authenticated link but the attribute requires that the link be authenticated before any client can access it.

***CYBLE_GATT_ERR_REQUEST_NOT_SUPPORTED*** Request not supported error code is used in the case when the server does not support the processing of an ATT request sent from the client.

***CYBLE_GATT_ERR_INVALID_OFFSET*** Invalid Offset error code is used in the case when the offset sent by the client in the Read blob/Prepare Write Request is invalid with respect to the length of the value in the server.

***CYBLE_GATT_ERR_INSUFFICIENT_AUTHORIZATION*** Insufficient Authorization error code is used in the case when the ATT server does not Authorize the client and hence prohibiting the client from reading the handle value.

***CYBLE_GATT_ERR_PREPARE_WRITE_QUEUE_FULL*** Write queue full error code is used when there is no more space left in the prepare write queue on the server to entertain any more prepare writes from a client.

***CYBLE_GATT_ERR_ATTRIBUTE_NOT_FOUND*** Attribute not found error is used when the ATT server cannot find any handles that belong to the Attribute type in the given range of handles that the client specified in its request. This error code can be sent to the client in response to the following request PDUs - Find Information, Find by Type Value, Read by Type, Read by Group Type requests.

***CYBLE_GATT_ERR_ATTRIBUTE_NOT_LONG*** Attribute Not Long error code is used when the client tries to read or write a Attribute handle's value which cannot be read or written through Read Blob or multiple prepare write requests.

***CYBLE_GATT_ERR_INSUFFICIENT_ENC_KEY_SIZE*** Insufficient encryption key size error code is used when the client tries to access an Attribute Handle's Value for which the link need to be encrypted with a key of certain minimum key size and the current link is encrypted with a key of lesser size than the minimum required.

***CYBLE_GATT_ERR_INVALID_ATTRIBUTE_LEN*** Invalid Attribute length error code is used when the Attribute value's length is not correct to process the request containing the value.

***CYBLE_GATT_ERR_UNLIKELY_ERROR*** Unlikely error is used when the processing of the Attribute request has encountered an error that is not covered by any other error code.

***CYBLE_GATT_ERR_INSUFFICIENT_ENCRYPTION*** Insufficient encryption error code is used when the client tries to read or write an Attribute handle which requires the link to be encrypted and the link is currently not encrypted.

***CYBLE_GATT_ERR_UNSUPPORTED_GROUP_TYPE*** Unsupported Group Type error code is used when the Attribute type requested in the Read by Group Type request is not a valid grouping attribute on the server.

***CYBLE_GATT_ERR_INSUFFICIENT_RESOURCE*** Insufficient Resources error code is used when the ATT server does not have enough resources such as memory etc. to process the request from the client.

***CYBLE_GATT_ERR_TRIGGER_CODITION_VALUE_NOT_SUPPORTED*** Other Error Groups for ATT - GATT Reserved: GATT-ATT Error codes 0x12 to 0x7F are reserved for Application Specific Error Code Range: 0x80 to 0x9F Reserved: 0xA0 to 0xDF Common Profile & Service Error Code : 0xE0 to 0xFF Trigger condition value not supported.

***CYBLE_GATT_ERR_HEART_RATE_CONTROL_POINT_NOT_SUPPORTED*** Heart Rate Control Point Not Supported error code is used when a unsupported code is written into Heart Rate service Control Point characteristic.

***CYBLE_GATT_ERR_USER_DATA_ACCESS_NOT_PERMITTED*** The user data access is not permitted (i.e. the user has not given consent in order to access these data).

***CYBLE_GATT_ERR_CPS_INAPPROPRIATE_CONNECTION_PARAMETERS*** The notifications of the Cycling Power Vector characteristic cannot be sent due to inappropriate connection parameters.

***CYBLE_GATT_ERR_HTS_OUT_OF_RANGE*** The value is considered invalid and outside of the range allowed by the characteristic.

***CYBLE_GATTS_ERR_PROCEDURE_ALREADY_IN_PROGRESS*** Procedure Already in Progress error code is used when a profile or service request cannot be serviced because an operation that has been previously triggered is still in progress.

***CYBLE_GATT_ERR_OP_CODE_NOT_SUPPORTED*** The Op Code Not Supported error code is used when a unsupported Op Code is written into Control Point characteristic.

***CYBLE_GATT_ERR_MISSING_CRC*** The Missing CRC error code is used when the CRC is missed in the incoming characteristic value.

***CYBLE_GATTS_ERR_CCCD_IMPROPERLY_CONFIGURED*** Client Characteristic Configuration Descriptor Improperly Configured error code is used when a Client Characteristic Configuration descriptor is not configured according to the requirements of the profile or service.

***CYBLE_GATTS_ERR_OPERATION_FAILED*** The Operation Failed error code is used when the device is unable to complete a procedure for any reason.

***CYBLE_GATT_ERR_INVALID_CRC*** The Invalid CRC error code is used when the CRC is invalid in the incoming characteristic value.

***CYBLE_GATTS_ERR_HPS_INVALID_REQUEST*** A HTTP Control Point request cannot be serviced because content of the URI, the HTTP Headers or the HTTP Entity Body characteristics is not set correctly.

***CYBLE_GATTS_ERR_NETWORK_NOT_AVAILABLE*** Network connection not available.

***CYBLE_GATT_ERR_ANS_COMMAND_NOT_SUPPORTED*** Command Not Supported used by the Alert Notification Server when the Client sends incorrect value of the Command ID or Category ID of to the Alert Notification Control Point Characteristic.

***CYBLE_GATT_ERR_ANCS_UNKNOWN_COMMAND*** Unknown command error code used by the Apple Notification Center Server when the Client sends unknown command value of the Apple Notification Center Service Control Point Characteristic.

***CYBLE_GATT_ERR_ANCS_INVALID_COMMAND*** Invalid command error code used by the Apple Notification Center Server when the Client sends invalid command value of the Apple Notification Center Service Control Point Characteristic.

***CYBLE_GATT_ERR_ANCS_INVALID_PARAMETER*** Invalid parameter error code used by the Apple Notification Center Server when the Client sends invalid parameter value of the Apple Notification Center Service Control Point Characteristic.

***CYBLE_GATT_ERR_ANCS_ACTION_FAILED*** Action failed error code used by the Apple Notification Center Server when some Apple Notification Center Service Control Point Characteristic command processing goes wrong

***CYBLE_GATT_ERR_CCCD_IMPROPERLY_CONFIGURED*** Client Characteristic Configuration Descriptor Improperly Configured error code is used when a Client Characteristic Configuration descriptor is not configured according to the requirements of the profile or service.

***CYBLE_GATT_ERR_PROCEDURE_ALREADY_IN_PROGRESS*** The Procedure Already in Progress error code is used when a profile or service request cannot be serviced because an operation that has been previously triggered is still in progress.

***CYBLE_GATT_ERR_OUT_OF_RANGE*** Out of Range error code is used when an attribute value is out of range as defined by a profile or service specification.

# L2CAP Functions

## Description

The L2CAP APIs allow access to the Logical link control and adaptation protocol (L2CAP) layer of the BLE stack.

The L2CAP API names begin with CyBle_L2cap.

## Modules

- L2CAP Definitions and Data Structures
  *Contains the L2CAP specific definitions and data structures used in the L2CAP APIs.*

## Functions

- CYBLE_API_RESULT_T CyBle_L2capCbfcRegisterPsm (uint16 l2capPsm, uint16 creditLwm)
- CYBLE_API_RESULT_T CyBle_L2capCbfcUnregisterPsm (uint16 l2capPsm)
- CYBLE_API_RESULT_T CyBle_L2capCbfcConnectReq (uint8 bdHandle, uint16 remotePsm, uint16 localPsm, CYBLE_L2CAP_CBFC_CONNECT_PARAM_T *param)
- CYBLE_API_RESULT_T CyBle_L2capCbfcConnectRsp (uint16 localCid, uint16 response, CYBLE_L2CAP_CBFC_CONNECT_PARAM_T *param)
- CYBLE_API_RESULT_T CyBle_L2capCbfcSendFlowControlCredit (uint16 localCid, uint16 credit)
- CYBLE_API_RESULT_T CyBle_L2capChannelDataWrite (uint8 bdHandle, uint16 localCid, uint8 *buffer, uint16 bufferLen)
- CYBLE_API_RESULT_T CyBle_L2capDisconnectReq (uint16 localCid)
- CYBLE_API_RESULT_T CyBle_L2capLeConnectionParamUpdateRequest (uint8 bdHandle, CYBLE_GAP_CONN_UPDATE_PARAM_T *connParam)
- CYBLE_API_RESULT_T CyBle_L2capLeConnectionParamUpdateResponse (uint8 bdHandle, uint16 result)

## Function Documentation

### CYBLE_API_RESULT_T CyBle_L2capCbfcRegisterPsm (uint16 *l2capPsm*, uint16 *creditLwm*)

This function registers a new upper layer protocol or PSM to L2CAP, along with the set of callbacks for the L2CAP Credit Based Flow Control mode. This is a blocking function. No event is generated on calling this function.

Refer Bluetooth 4.1 core specification, Volume 3, Part A, section 3.4 for more details about credit based flow control mode of operation.

**Parameters:**

| | |
|---|---|
| *l2capPsm* | PSM value of the higher-level protocol |
| *creditLwm* | Upper Layer defined Receive Credit Low Mark |

**Returns:**

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

| Error codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If 'l2capPsm' is 0 |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | Cannot register more than one PSM |
| CYBLE_ERROR_L2CAP_PSM_NOT_IN_RANGE | If the PSM is not in range of 0x0001 - 0x00FF. |
| CYBLE_ERROR_L2CAP_PSM_ALREADY_REGISTERED | PSM already Registered |

### CYBLE_API_RESULT_T CyBle_L2capCbfcUnregisterPsm (uint16 *l2capPsm*)

This function de-registers an upper layer protocol or LE_PSM from L2CAP for the L2CAP Credit Based Flow Control mode. This is a blocking function. No event is generated on calling this function.

**Parameters:**

| | |
|---|---|
| *l2capPsm* | PSM value of the higher-level protocol |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_L2CAP_PSM_WRONG_ENCODING | L2CAP PSM value specified is incorrect or does not exist |

### CYBLE_API_RESULT_T CyBle_L2capCbfcConnectReq (uint8 *bdHandle*, uint16 *remotePsm*, uint16 *localPsm*, CYBLE_L2CAP_CBFC_CONNECT_PARAM_T **param*)

This L2CAP function initiates L2CAP channel establishment procedure in Credit Based Flow Control (CBFC) mode. Connection establishment is initiated to the specified remote Bluetooth device, for the specified PSM representing an upper layer protocol above L2CAP. This is a non-blocking function.

At the receiver's end, CYBLE_EVT_L2CAP_CBFC_CONN_IND event is generated. In response to this call, CYBLE_EVT_L2CAP_CBFC_CONN_CNF event is generated at the sender's end.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.22 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *remotePsm* | Remote PSM, representing the upper layer protocol above L2CAP. |
| *localPsm* | Local PSM, representing the upper layer protocol above L2CAP. |
| *param* | This parameter must be a pointer to the CYBLE_L2CAP_CBFC_CONNECT_PARAM_T variable containing the connection parameters for the L2CAP channel. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If "param" is NULL |
| CYBLE_ERROR_INSUFFICIENT_RESOURCES | Insufficient resources |
| CYBLE_L2CAP_PSM_NOT_REGISTERED | PSM not Registered |

### CYBLE_API_RESULT_T CyBle_L2capCbfcConnectRsp (uint16 *localCid*, uint16 *response*, CYBLE_L2CAP_CBFC_CONNECT_PARAM_T **param*)

This L2CAP function enables an upper layer protocol to respond to L2CAP connection request for LE Credit Based Flow Control mode of the specified PSM from the specified remote Bluetooth device. This is a non-blocking function. It is mandatory that the upper layer PSM always responds back by calling this function upon receiving CBFC Connection Request (CYBLE_EVT_L2CAP_CBFC_CONN_IND) event.

The channel is established (opened) only when the PSM concerned responds back with an event indicating success (CYBLE_EVT_L2CAP_CBFC_CONN_CNF, at the peer device's end). Otherwise, the channel

establishment request from the peer will be rejected by L2CAP with appropriate result and status as received from the upper layer PSM.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.23 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *localCid* | This parameter specifies the local L2CAP channel end-point for this new L2CAP channel. On receipt of L2CAP Connect Request command from the peer, local L2CAP will temporarily create a channel. This parameter identifies the new channel. If the upper layer PSM chooses to reject this connection, this temporary channel will be closed. |
| *response* | This parameter specifies the response of the upper layer for the new L2CAP channel establishment request from the peer. It must be set to a value as specified in L2CAP Connect Result Codes. Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.23 for more details. |
| *param* | This parameter must be a pointer to the CYBLE_L2CAP_CBFC_CONNECT_PARAM_T variable containing the connection parameters for the L2CAP channel. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If "param" is NULL |
| CYBLE_ERROR_L2CAP_CONNECTION_ENTITY_NOT_FOUND | Connection entity is not found |

### CYBLE_API_RESULT_T CyBle_L2capCbfcSendFlowControlCredit (uint16 *localCid*, uint16 *credit*)

This L2CAP function enables an upper layer protocol to send LE Flow Control Credit packet to peer Bluetooth device, when it is capable of receiving additional LE-frames. This is a non-blocking function.

This function is invoked when the device is expecting more data from the peer device and it gets an event indicating that the peer device is low on credits CYBLE_EVT_L2CAP_CBFC_RX_CREDIT_IND for which it needs to respond by sending credits by invoking this function. Once the peer device receives these credits, it gets CYBLE_EVT_L2CAP_CBFC_TX_CREDIT_IND event indicating the same. It is the responsibility of the application layer of the device sending the credit to keep track of the total number of credits and making sure that it does not exceed 65535.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.24 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *localCid* | This parameter specifies the local channel end-point for the L2CAP channel. For the initiator of L2CAP channel establishment, this must be set to the value indicated by the CYBLE_EVT_L2CAP_CBFC_CONN_CNF event. For the responder, the upper layer protocol obtains this value when it receives the event CYBLE_EVT_L2CAP_CBFC_CONN_IND. |
| *credit* | The credit value field represents number of credits the receiving device can increment. The credit value field is a number between 1 and 65535. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_L2CAP_CONNECTION_ENTITY_NOT_FOUND | L2CAP connection instance is not present |

**CYBLE_API_RESULT_T CyBle_L2capChannelDataWrite (uint8 *bdHandle*, uint16 *localCid*, uint8 **buffer*, uint16 *bufferLen*)**

This function sends a data packet on the L2CAP CBFC channel. This is a blocking function.

This API function generates 'CYBLE_EVT_L2CAP_CBFC_DATA_WRITE_IND' event which is kept for backward compatibility and the user should handle CYBLE_API_RESULT_T to determine whether the last data packet was sent out properly.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 3.4 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle. |
| *localCid* | This parameter specifies the local channel end-point for the L2CAP channel. For the initiator of L2CAP channel establishment, this must be set to the value indicated by the CYBLE_EVT_L2CAP_CBFC_CONN_CNF event. For the responder, the upper layer protocol obtains this value when it receives the event CYBLE_EVT_L2CAP_CBFC_CONN_IND. |
| *buffer* | Buffer containing packet to be sent. |
| *bufferLen* | L2CAP Data Packet length. It shall be of lesser than the size of both local L2CAP MTU & peer L2CAP MTU size. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If "buffer" is NULL |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_NO_CONNECTION | No Link Layer connection is present |
| CYBLE_L2CAP_CHANNEL_NOT_FOUND | No L2CAP channel found corresponding to CID |
| CYBLE_L2CAP_NOT_ENOUGH_CREDITS | Not Enough Credits to transfer data |

**CYBLE_API_RESULT_T CyBle_L2capDisconnectReq (uint16 *localCid*)**

This function initiates sending of an L2CAP Disconnect Request (CYBLE_EVT_L2CAP_CBFC_DISCONN_IND event received by the peer device) command to the remote L2CAP entity to initiate disconnection of the referred L2CAP channel. This is a non-blocking function.

Disconnection of the L2CAP channel always succeeds - either by reception of the L2CAP Disconnect Response from the peer, or by timeout. In any case, L2CAP will confirm disconnection of the channel, by calling the CYBLE_EVT_L2CAP_CBFC_DISCONN_CNF event.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.6 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *localCid* | This parameter specifies the local channel end-point for the L2CAP channel.<br>• For initiator of L2CAP channel establishment, this must be set to the value indicated by the event CYBLE_EVT_L2CAP_CBFC_CONN_CNF.<br>• For the responder, the upper layer protocol obtains this value when it receives the event CYBLE_EVT_L2CAP_CBFC_CONN_IND. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_OPERATION | No Link Layer connection is present |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_L2CAP_CONNECTION_ENTITY_NOT_FOUND | No connection entity found which can be disconnected |

### CYBLE_API_RESULT_T CyBle_L2capLeConnectionParamUpdateRequest (uint8 *bdHandle*, CYBLE_GAP_CONN_UPDATE_PARAM_T **connParam*)

This function sends the connection parameter update request to the Master of the link. This is a non-blocking function. This function can only be used from device connected in LE slave role.

To send connection parameter update request from the master to the slave, use CyBle_GapcConnectionParamUpdateRequest() function. This function results in CYBLE_EVT_L2CAP_CONN_PARAM_UPDATE_REQ event at the Master's end.

Refer to Bluetooth 4.1 core specification, Volume 3, Part A, section 4.20 for more details about this operation.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *connParam* | Pointer to a variable of type CYBLE_GAP_CONN_UPDATE_PARAM_T which indicates the response to the Connection Parameter Update Request |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If "connParam" is NULL |
| CYBLE_ERROR_INVALID_OPERATION | Connection Parameter Update Request is not allowed |

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_NO_CONNECTION | No Link Layer connection is present |

**CYBLE_API_RESULT_T CyBle_L2capLeConnectionParamUpdateResponse (uint8 *bdHandle*, uint16 *result*)**

This API function sends the connection parameter update response to slave. This API function can only be used from device connected in LE master role.

**Parameters:**

| | |
|---|---|
| *bdHandle* | Peer device handle |
| *result* | This field indicates the response to the Connection Parameter Update Request |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | If 'result' is invalid (greater than connection parameter reject code i.e., 0x0001) |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_NO_CONNECTION | No Link Layer connection is present |

# L2CAP Definitions and Data Structures

## Description

Contains the L2CAP specific definitions and data structures used in the L2CAP APIs.

## Data Structures

- struct CYBLE_L2CAP_CBFC_CONNECT_PARAM_T
- struct CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T
- struct CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T
- struct CYBLE_L2CAP_CBFC_DISCONN_CNF_PARAM_T
- struct CYBLE_L2CAP_CBFC_RX_PARAM_T
- struct CYBLE_L2CAP_CBFC_LOW_RX_CREDIT_PARAM_T
- struct CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T
- struct CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T

## Enumerations

- enum CYBLE_L2CAP_COMMAND_REJ_REASON_T
- enum CYBLE_L2CAP_RESULT_PARAM_T

## Data Structure Documentation

### struct CYBLE_L2CAP_CBFC_CONNECT_PARAM_T

**Data Fields**

- uint16 mtu
- uint16 mps
- uint16 credit

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_CONNECT_PARAM_T::mtu**

L2CAP MTU - Maximum SDU Size

The L2CAP MTU field specifies the maximum SDU size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Request can receive on this channel. L2CAP implementations shall support a minimum L2CAP MTU size of 23 octets.

**uint16 CYBLE_L2CAP_CBFC_CONNECT_PARAM_T::mps**

MPS - Maximum PDU Size

The MPS field specifies the maximum payload size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Request is capable of receiving on this channel. L2CAP implementations shall support a minimum MPS of 23 octets and may support an MPS up to 65488 octets.

**uint16 CYBLE_L2CAP_CBFC_CONNECT_PARAM_T::credit**

Initial number of Credits

The initial credit value indicates the number of LE-frames that the peer device can send to the L2CAP layer entity sending the LE Credit Based Connection Request. The initial credit value shall be in the range of 0 to

1.

### struct CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T

**Data Fields**

- uint8 bdHandle
- uint16 lCid
- uint16 psm
- CYBLE_L2CAP_CBFC_CONNECT_PARAM_T connParam

**Field Documentation**

**uint8 CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T::bdHandle**

bd handle of the remote device

**uint16 CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T::lCid**

Local CID

**uint16 CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T::psm**

Local PSM value

**CYBLE_L2CAP_CBFC_CONNECT_PARAM_T CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T::connParam**

L2CAP Credit based flow Connection parameter

### struct CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T

**Data Fields**

- uint8 bdHandle
- uint16 lCid
- uint16 response
- CYBLE_L2CAP_CBFC_CONNECT_PARAM_T connParam

**Field Documentation**

**uint8 CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T::bdHandle**
bd handle of the remote device

**uint16 CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T::lCid**
Local CID

**uint16 CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T::response**
Response codes for Connection parameter update request

**CYBLE_L2CAP_CBFC_CONNECT_PARAM_T CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T::connParam**
L2CAP Credit based flow Connection parameter

**struct CYBLE_L2CAP_CBFC_DISCONN_CNF_PARAM_T**

**Data Fields**

- uint16 lCid
- CYBLE_L2CAP_RESULT_PARAM_T result

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_DISCONN_CNF_PARAM_T::lCid**
Local CID

**CYBLE_L2CAP_RESULT_PARAM_T CYBLE_L2CAP_CBFC_DISCONN_CNF_PARAM_T::result**
The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed or is pending.

**struct CYBLE_L2CAP_CBFC_RX_PARAM_T**

**Data Fields**

- uint16 lCid
- CYBLE_L2CAP_RESULT_PARAM_T result
- uint8 * rxData
- uint16 rxDataLength

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_RX_PARAM_T::lCid**
Local CID

**CYBLE_L2CAP_RESULT_PARAM_T CYBLE_L2CAP_CBFC_RX_PARAM_T::result**
A result value of 0x0000 indicates success, while a non-zero value indicates an error condition (e.g. peer device violating credit flow, or L2CAP MTU size limit)

**uint8* CYBLE_L2CAP_CBFC_RX_PARAM_T::rxData**
Received L2cap Data

**uint16 CYBLE_L2CAP_CBFC_RX_PARAM_T::rxDataLength**
Received L2cap Data Length

**struct CYBLE_L2CAP_CBFC_LOW_RX_CREDIT_PARAM_T**

**Data Fields**

- uint16 lCid
- uint16 credit

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_LOW_RX_CREDIT_PARAM_T::lCid**
Local CID

**uint16 CYBLE_L2CAP_CBFC_LOW_RX_CREDIT_PARAM_T::credit**
The number of credits (LE-frames)

**struct CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T**

**Data Fields**

- uint16 lCid
- CYBLE_L2CAP_RESULT_PARAM_T result
- uint16 credit

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T::lCid**
Local CID

**CYBLE_L2CAP_RESULT_PARAM_T CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T::result**
A result value of 0x0000 indicates success, while a non-zero value indicates an error condition (e.g. credit overflow, if total number of credits crosses specification defined maximum limit of 0xFFFF)

**uint16 CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T::credit**
The number of credits (LE-frames)

**struct CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T**

**Data Fields**

- uint16 lCid
- CYBLE_L2CAP_RESULT_PARAM_T result
- uint8 * buffer
- uint16 bufferLength

**Field Documentation**

**uint16 CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T::lCid**
Local CID

**CYBLE_L2CAP_RESULT_PARAM_T CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T::result**
The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed or is pending.

**uint8* CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T::buffer**
Currently NULL. For future usage

**uint16 CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T::bufferLength**
Currently 0. For future usage

## Enumeration Type Documentation

**enum CYBLE_L2CAP_COMMAND_REJ_REASON_T**
Reason for command reject event - CYBLE_EVT_L2CAP_COMMAND_REJ

**Enumerator**

***CYBLE_L2CAP_COMMAND_NOT_UNDERSTOOD*** Command Not Understood

***CYBLE_L2CAP_SIGNALLING_MTU_EXCEEDED*** Signaling L2CAP MTU exceeded

***CYBLE_L2CAP_INVALID_CID_IN_REQUEST*** Invalid Connection Identifier in request

**enum CYBLE_L2CAP_RESULT_PARAM_T**
The result code of call back structures for L2CAP

**Enumerator**

***CYBLE_L2CAP_RESULT_SUCCESS*** Operation Successful

***CYBLE_L2CAP_RESULT_COMMAND_TIMEOUT*** Command timeout, if l2cap signaling channel timeout occurs, app should disconnect.

***CYBLE_L2CAP_RESULT_INCORRECT_SDU_LENGTH*** Invalid sdu length

***CYBLE_L2CAP_RESULT_NOT_ENOUGH_CREDITS*** Not enough credit to perform this operation

***CYBLE_L2CAP_RESULT_CREDIT_OVERFLOW*** Credit overflow. Total credit exceeded 65535 (maximum)

***CYBLE_L2CAP_RESULT_UNACCEPTABLE_CREDIT_VALUE*** Invalid credit value, receive credit is Zero

# BLE Common Events

## Description

The BLE stack generates events to notify the application on various status alerts concerning the stack. These can be generic stack events or can be specific to GAP, GATT or L2CAP layers. The service specific events are handled separately in BLE Service-Specific Events.

## Macros

- #define CYBLE_EVT_HOST_STACK_T CYBLE_EVENT_T

## Enumerations

- enum CYBLE_EVENT_T
- enum CYBLE_HCI_ERROR_T

## Macro Definition Documentation

### #define CYBLE_EVT_HOST_STACK_T CYBLE_EVENT_T

Alias of CYBLE_EVENT_T, which is used internally by Stack

## Enumeration Type Documentation

### enum CYBLE_EVENT_T

Host stack events. Generic events: 0x01 to 0x1F GAP events: 0x20 to 0x3F GATT events: 0x40 to 0x6F L2CAP events: 0x70 to 0x7F Future use: 0x80 to 0xFF

**Enumerator**

***CYBLE_EVT_HOST_INVALID*** This event is triggered by BLE stack when stack is in a bad state, Restarting stack is the only way to get out of the state

***CYBLE_EVT_STACK_ON*** This event is received when BLE stack is initialized and turned ON by invoking CyBle_StackInit () function.

***CYBLE_EVT_TIMEOUT*** This event is received when there is a timeout and application needs to handle the event. Timeout reason is defined by CYBLE_TO_REASON_CODE_T.

***CYBLE_EVT_HARDWARE_ERROR*** This event indicates that some internal hardware error has occurred. Reset of the hardware may be required.

***CYBLE_EVT_HCI_STATUS*** This event is triggered by 'Host Stack' if 'Controller' responds with an error code for any HCI command. Event parameter returned will be an HCI error code as defined in Bluetooth 4.1 core specification, Volume 2, Part D, section 1.3 or User can refer CYBLE_HCI_ERROR_T for HCI error codes. This event will be received only if there is an error.

***CYBLE_EVT_STACK_BUSY_STATUS*** This event is triggered by host stack if BLE stack is busy or not. Event Parameter corresponding to this event will indicate the state of BLE stack's internal protocol buffers for the application to safely initiate data transactions (GATT, GAP Security, and L2CAP transactions) with the peer BLE device. Event parameter is of type uint8.

CYBLE_STACK_STATE_BUSY (0x01) = CYBLE_STACK_STATE_BUSY indicates application that BLE stack's internal buffers are about to be filled, and the remaining buffers are required to respond peer BLE device After this event, application shall not initiate (GATT, GAP Security and L2CAP data transactions). However application shall respond to peer initiated transactions to prevent BLE protocol timeouts to occur. Application initiated data transactions can be resumed after CYBLE_EVT_STACK_BUSY_STATUS event with parameter 'CYBLE_STACK_STATE_FREE' is received.

CYBLE_STACK_STATE_FREE (0x00) = CYBLE_STACK_STATE_FREE indicates application that pending transactions are completed and sufficient buffers are available to process application initiated transactions. The 'CYBLE_EVT_STACK_BUSY_STATUS' event with 'CYBLE_STACK_STATE_FREE' is indicated to application if BLE Stack's internal buffer state has transitioned from 'CYBLE_STACK_STATE_BUSY' to 'CYBLE_STACK_STATE_FREE'.

To increase BLE stack's internal buffers count and achieve better throughput for attribute MTU greater then 32, use MaxAttrNoOfBuffer parameter in the Expression view of the Advanced tab.

***CYBLE_EVT_MEMORY_REQUEST*** This event is received when stack wants application to provide memory to process remote request. Event parameter is of type CYBLE_MEMORY_REQUEST_T. This event is automatically handled by the component for the CYBLE_PREPARED_WRITE_REQUEST request. The component allocates sufficient memory for the long write request with assumption that attribute MTU size is negotiated to the minimum possible value. Application could use dynamic memory allocation to save static RAM memory consumption. To enable this event for application level, set EnableExternalPrepWriteBuff parameter in the Expression view of the Advanced tab to the true.

***CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT*** This event is triggered every time a device is discovered; pointer to structure of type CYBLE_GAPC_ADV_REPORT_T is returned as the event parameter.

***CYBLE_EVT_GAP_AUTH_REQ*** This event is received by Peripheral and Central devices. When it is received by Peripheral, peripheral needs to Call CyBle_GappAuthReqReply() to reply to authentication request from Central.

When this event is received by Central, that means the slave has requested Central to initiate authentication procedure. Central needs to call CyBle_GappAuthReq() to initiate authentication procedure. Pointer to structure of type CYBLE_GAP_AUTH_INFO_T is returned as the event parameter.

***CYBLE_EVT_GAP_PASSKEY_ENTRY_REQUEST*** This event indicates that the device has to send passkey to be used during the pairing procedure. CyBle_GapAuthPassKeyReply() is required to be called with valid parameters on receiving this event.

Refer to Bluetooth Core Spec. 4.1, Part H, Section 2.3.5.1 Selecting STK Generation Method.

Nothing is returned as part of the event parameter.

***CYBLE_EVT_GAP_PASSKEY_DISPLAY_REQUEST*** This event indicates that the device needs to display passkey during the pairing procedure.

Refer to Bluetooth Core Spec. 4.1, Part H, Section 2.3.5.1 Selecting STK Generation Method.

Pointer to data of type 'uint32' is returned as part of the event parameter. Passkey can be any 6-decimal-digit value.

***CYBLE_EVT_GAP_AUTH_COMPLETE*** This event indicates that the authentication procedure has been completed.

The event parameter contains the security information as defined by CYBLE_GAP_AUTH_INFO_T. This event is generated at the end of the following three operations: Authentication is initiated with a newly connected device Encryption is initiated with a connected device that is already bonded Re-Encryption is initiated with a connected device with link already encrypted During encryption/re-encryption, the Encryption Information exchanged during the pairing process is used to encrypt/re-encrypt the link. As this does not

modify any of the authentication parameters with which the devices were paired, this event is generated with NULL event data and the result of the encryption operation.

***CYBLE_EVT_GAP_AUTH_FAILED*** Authentication process failed between two devices. The return value of type CYBLE_GAP_AUTH_FAILED_REASON_T indicates the reason for failure.

***CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP*** Peripheral device has started/stopped advertising. This event is generated after making a call to CyBle_GappEnterDiscoveryMode and CyBle_GappExitDiscoveryMode functions. The event parameter contains the status which is of type 'uint8'.

If the data is '0x00', it indicates 'success'; Anything else indicates 'failure'.

***CYBLE_EVT_GAP_DEVICE_CONNECTED*** This event is generated at the GAP Peripheral end after connection is completed with peer Central device. For GAP Central device, this event is generated as in acknowledgment of receiving this event successfully by BLE Controller. Once connection is done, no more event is required but if fails to establish connection, 'CYBLE_EVT_GAP_DEVICE_DISCONNECTED' is passed to application. ' CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE' event is triggered instead of 'CYBLE_EVT_GAP_DEVICE_CONNECTED', if Link Layer Privacy is enabled in component customizer. Event parameter is a pointer to a structure of type CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T.

***CYBLE_EVT_GAP_DEVICE_DISCONNECTED*** Disconnected from remote device or failed to establish connection. Parameter returned with the event contains pointer to the reason for disconnection, which is of type uint8. For details refer core spec 4.2, vol2, part D or User can refer CYBLE_HCI_ERROR_T for HCI error codes

***CYBLE_EVT_GAP_ENCRYPT_CHANGE*** Encryption change event for active connection. 'evParam' can be decoded as evParam[0] = 0x00 -> Encryption OFF evParam[0] = 0x01 -> Encryption ON Any other value of evParam[0] -> Error

This is an informative event for application when there is a change in encryption. Application may choose to ignore it.

***CYBLE_EVT_GAP_CONNECTION_UPDATE_COMPLETE*** This event is generated at the GAP Central and the Peripheral end after connection parameter update is requested from the host to the controller. Event parameter is a pointer to a structure of type CYBLE_GAP_CONN_PARAM_UPDATED_IN_CONTROLLER_T.

***CYBLE_EVT_GAPC_SCAN_START_STOP*** Central device has started/stopped scanning. This event is generated after making a call to CyBle_GapcStartDiscovery and CyBle_GapcStopDiscovery APIs. The event parameter contains the status, which is of type 'uint8'.

If the data is '0x00', it indicates 'success'; Anything else indicates 'failure'.

***CYBLE_EVT_GAP_KEYINFO_EXCHNGE_CMPLT*** Indication that the SMP keys exchange with peer device is complete, the event handler is expected to store the peer device keys, especially IRK which is used to resolve the peer device after the connection establishment.

Event parameter returns data of type CYBLE_GAP_SMP_KEY_DIST_T containing the peer device keys.

***CYBLE_EVT_GAP_NUMERIC_COMPARISON_REQUEST*** This event indicates that the device needs to display passkey during secure connection pairing procedure. CyBle_GapAuthPassKeyReply() is required to be called with valid parameters on receiving this event. Since no key to be entered by the user for Numeric comparison, parameter passkey for the function CyBle_GapAuthPassKeyReply will be ignored. Event parameter is a pointer to a 6 digit Passkey value.

***CYBLE_EVT_GAP_KEYPRESS_NOTIFICATION*** This event is generated when keypress (Secure connections) is received from peer device.

***CYBLE_EVT_GAP_OOB_GENERATED_NOTIFICATION*** This event is generated when OOB generation for Secure connections is complete. Event parameter is of type 'CYBLE_GAP_OOB_DATA_T'

***CYBLE_EVT_GAP_DATA_LENGTH_CHANGE*** The LE Data Length Change event notifies the Host of a change to either the maximum Payload length or the maximum transmission time of Data Channel PDUs in either direction. The values reported are the maximum that will actually be used on the connection following the change. Event parameter is of type 'CYBLE_GAP_CONN_DATA_LENGTH_T'

***CYBLE_EVT_GAP_ENHANCE_CONN_COMPLETE*** The LE Enhanced Connection Complete event indicates application that a new connection has been created when Link Layer Privacy is enabled in component customizer. Event parameter is of type 'CYBLE_GAP_ENHANCE_CONN_COMPLETE_T'

***CYBLE_EVT_GAPC_DIRECT_ADV_REPORT*** The LE Direct Advertising Report event indicates that directed advertisements have been received where the advertiser is using a resolvable private address for the InitA field in the ADV_DIRECT_IND PDU and the Scanning_Filter_Policy is equal to 0x02 or 0x03. Event parameter is of type 'CYBLE_GAPC_DIRECT_ADV_REPORT_T'

***CYBLE_EVT_GAP_SMP_NEGOTIATED_AUTH_INFO*** SMP negotiated auth info event is raised as soon as SMP has completed pairing properties (feature exchange) negotiation. The event parameter is CYBLE_GAP_AUTH_INFO_T. CYBLE_GAP_AUTH_INFO_T will have the negotiated parameter, the pairing should either pass with these negotiated parameters or may fail. This event is applicable to both GAP Central and GAP Peripheral devices. In GAP Peripheral, this event is called from API function CyBle_GappAuthReqReply context.

***CYBLE_EVT_GAP_CONN_ESTB*** This event is generated when connection got established

***CYBLE_EVT_GAP_SCAN_REQ_RECVD*** SCAN_REQ received event User has to explicitly call CyBle_SetAppEventMask() by setting scan req event mask

***CYBLE_EVT_GAP_AUTH_REQ_REPLY_ERR*** This event is generated when in the CYBLE_EVT_GAP_AUTH_REQ component event handler CyBle_GappAuthReqReply() returned not CYBLE_ERROR_OK value. It's possible when the bonded device is full and application tries to initiate pairing with bonding enabled. Event parameter is of type 'CYBLE_API_RESULT_T'. Application will have to handle this event by removing an oldest (or any other) device from the bond list and call CyBle_GappAuthReqReply() function again.

***CYBLE_EVT_GAP_SMP_LOC_P256_KEYS_GEN_AND_SET_COMPLETE*** This event is generated when the local P-256 public-private key pair generation is completed and new keys are stored in the BLE Stack for SC pairing procedure. Event parameter is a pointer to structure of type CYBLE_GAP_SMP_LOCAL_P256_KEYS.

***CYBLE_EVT_GATTC_ERROR_RSP*** The event is received by the Client when the Server cannot perform the requested operation and sends out an error response. Event parameter is a pointer to a structure of type CYBLE_GATTC_ERR_RSP_PARAM_T.

***CYBLE_EVT_GATT_CONNECT_IND*** This event is generated at the GAP Peripheral end after connection is completed with peer Central device. For GAP Central device, this event is generated as in acknowledgment of receiving this event successfully by BLE Controller. Once connection is done, no more event is required but if fails to establish connection, 'CYBLE_EVT_GATT_DISCONNECT_IND' is passed to application. Event parameter is a pointer to a structure of type CYBLE_CONN_HANDLE_T.

***CYBLE_EVT_GATT_DISCONNECT_IND*** GATT is disconnected. Nothing is returned as part of the event parameter.

***CYBLE_EVT_GATTS_XCNHG_MTU_REQ*** 'GATT MTU Exchange Request' received from GATT client device. Event parameter contains the MTU size of type CYBLE_GATT_XCHG_MTU_PARAM_T.

***CYBLE_EVT_GATTC_XCHNG_MTU_RSP*** 'GATT MTU Exchange Response' received from server device. Event parameter is a pointer to a structure of type CYBLE_GATT_XCHG_MTU_PARAM_T.

***CYBLE_EVT_GATTC_READ_BY_GROUP_TYPE_RSP*** 'Read by Group Type Response' received from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_READ_BY_GRP_RSP_PARAM_T.

***CYBLE_EVT_GATTC_READ_BY_TYPE_RSP*** 'Read by Type Response' received from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_READ_BY_TYPE_RSP_PARAM_T.

***CYBLE_EVT_GATTC_FIND_INFO_RSP*** 'Find Information Response' received from server device. Event parameter is a pointer to a structure of type 'CYBLE_GATTC_FIND_INFO_RSP_PARAM_T.

***CYBLE_EVT_GATTC_FIND_BY_TYPE_VALUE_RSP*** 'Find by Type Value Response' received from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_FIND_BY_TYPE_RSP_PARAM_T.

***CYBLE_EVT_GATTC_READ_RSP*** 'Read Response' from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_READ_RSP_PARAM_T.

***CYBLE_EVT_GATTC_READ_BLOB_RSP*** 'Read Blob Response' from server. Event parameter is a pointer to a structure of type CYBLE_GATTC_READ_RSP_PARAM_T.

***CYBLE_EVT_GATTC_READ_MULTI_RSP*** 'Read Multiple Responses' from server. Event parameter is a pointer to a structure of type CYBLE_GATTC_READ_RSP_PARAM_T. The 'actualLen' field should be ignored as it is unused in this event response.

***CYBLE_EVT_GATTS_WRITE_REQ*** 'Write Request' from client device. Event parameter is a pointer to a structure of type CYBLE_GATTS_WRITE_REQ_PARAM_T .

***CYBLE_EVT_GATTC_WRITE_RSP*** 'Write Response' from server device. Event parameter is a pointer to a structure of type CYBLE_CONN_HANDLE_T.

***CYBLE_EVT_GATTS_WRITE_CMD_REQ*** 'Write Command' Request from client device. Event parameter is a pointer to a structure of type CYBLE_GATTS_WRITE_CMD_REQ_PARAM_T.

***CYBLE_EVT_GATTS_PREP_WRITE_REQ*** 'Prepare Write' Request from client device. Event parameter is a pointer to a structure of type CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T.

***CYBLE_EVT_GATTS_EXEC_WRITE_REQ*** 'Execute Write' request from client device. Event parameter is a pointer to a structure of type 'CYBLE_GATTS_EXEC_WRITE_REQ_T' This event will be triggered before GATT DB is modified. GATT Db will be updated only if there is no error condition provided by application. In case of error condition triggered during stack validation, partial write will occur. Write will be canceled from that handle where error has occurred and error response corresponding to that handle will be sent to remote. If at any point of time 'CYBLE_GATT_EXECUTE_WRITE_CANCEL_FLAG' is received in execWriteFlag fields of 'CYBLE_GATTS_EXEC_WRITE_REQ_T' structure, then all previous writes are canceled. For execute cancel scenario, all elements of 'CYBLE_GATTS_EXEC_WRITE_REQ_T' should be ignored except execWriteFlag and connHandle.

***CYBLE_EVT_GATTC_EXEC_WRITE_RSP*** 'Execute Write' response from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_EXEC_WRITE_RSP_T.

***CYBLE_EVT_GATTC_HANDLE_VALUE_NTF*** Notification data received from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T.

***CYBLE_EVT_GATTC_HANDLE_VALUE_IND*** Indication data received from server device. Event parameter is a pointer to a structure of type CYBLE_GATTC_HANDLE_VALUE_IND_PARAM_T.

***CYBLE_EVT_GATTS_HANDLE_VALUE_CNF*** Confirmation to indication response from client device. Event parameter is a pointer to a structure of type CYBLE_CONN_HANDLE_T.

***CYBLE_EVT_GATTS_DATA_SIGNED_CMD_REQ*** Confirmation to indication response from client device. Event parameter is a pointer to a structure of type CYBLE_GATTS_SIGNED_WRITE_CMD_REQ_PARAM_T. if value.val parameter is set to Zero, then signature is not matched and ignored by stack

***CYBLE_EVT_GATTC_STOP_CMD_COMPLETE*** Event indicating that GATT group procedure has stopped or completed, this event occurs only if application has called CyBle_GattcStopCmd API function. Event parameters shall be ignored

***CYBLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ*** Event parameter type is CYBLE_GATTS_CHAR_VAL_READ_REQ_T. It is triggered on server side when client sends read request and when characteristic has CYBLE_GATT_DB_ATTR_CHAR_VAL_RD_EVENT property set. This event could be ignored by application unless it need to response by error response which needs to be set in gattErrorCode field of event parameter. Application can update attribute value when this event is received

***CYBLE_EVT_GATTC_LONG_PROCEDURE_END*** Event indicates that GATT long procedure is end and stack will not send any further requests to peer. Either this event or 'CYBLE_EVT_GATTC_ERROR_RSP' will be received by application. This event may get triggered for below GATT long procedures:

1. CyBle_GattcDiscoverAllPrimaryServices
2. CyBle_GattcDiscoverPrimaryServiceByUuid
3. CyBle_GattcFindIncludedServices

4. CyBle_GattcDiscoverAllCharacteristics
5. CyBle_GattcDiscoverCharacteristicByUuid
6. CyBle_GattcDiscoverAllCharacteristicDescriptors
7. CyBle_GattcReadLongCharacteristicValues
8. CyBle_GattcReadLongCharacteristicDescriptors

Event parameter is ATT opcode for the corresponding long GATT Procedure.

***CYBLE_EVT_L2CAP_CONN_PARAM_UPDATE_REQ*** This event indicates the connection parameter update received from the remote device. The application is expected to reply to L2CAP using the CyBle_L2capLeConnectionParamUpdateResponse() function to respond to the remote device, whether parameters are accepted or rejected.

Event Parameter pointer points to data of type 'CYBLE_GAP_CONN_UPDATE_PARAM_T'

***CYBLE_EVT_L2CAP_CONN_PARAM_UPDATE_RSP*** This event indicates the connection parameter update response received from the master. Event Parameter pointer points to data with two possible values: Accepted = 0x0000 Rejected = 0x0001

Data is of type unit16.

***CYBLE_EVT_L2CAP_COMMAND_REJ*** This event indicates that the request send over l2cap signaling has been rejected. Event parameter is a pointer to a structure of type CYBLE_L2CAP_COMMAND_REJ_REASON_T.

***CYBLE_EVT_L2CAP_CBFC_CONN_IND*** This event is used to inform application of the incoming L2CAP CBFC Connection Request. Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_CONN_IND_PARAM_T is returned.

***CYBLE_EVT_L2CAP_CBFC_CONN_CNF*** This event is used to inform application of the L2CAP CBFC Connection Response/Confirmation. Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_CONN_CNF_PARAM_T is returned.

***CYBLE_EVT_L2CAP_CBFC_DISCONN_IND*** This event is used to inform application of the L2CAP CBFC Disconnection Request received from the Peer device. Event parameter is a pointer to Local CID of type unit16.

***CYBLE_EVT_L2CAP_CBFC_DISCONN_CNF*** This event is used to inform application of the L2CAP CBFC Disconnection confirmation/Response received from the Peer device. Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_DISCONN_CNF_PARAM_T.

***CYBLE_EVT_L2CAP_CBFC_DATA_READ*** This event is used to inform application of data received over L2CAP CBFC channel. Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_RX_PARAM_T.

***CYBLE_EVT_L2CAP_CBFC_RX_CREDIT_IND*** This event is used to inform the application of receive credits reached low mark. After receiving L2CAP data/payload from peer device for a specification Channel, the available credits are calculated.

If the credit count goes below the low mark, this event is called to inform the application of the condition, so that if the application wants it can send more credits to the peer device.

Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_LOW_RX_CREDIT_PARAM_T.

***CYBLE_EVT_L2CAP_CBFC_TX_CREDIT_IND*** This event is used to inform application of having received transmit credits. This event is called on receiving LE Flow Control Credit from peer device.

Event parameter is a pointer to a structure of type CYBLE_L2CAP_CBFC_LOW_TX_CREDIT_PARAM_T.

If the 'result' field of the received data is non-zero, this indicates an error. If the sum of 'credit' field value and the previously available credit at the peer device receiving credit information exceeds 65535, it indicates a 'credit overflow' error.

In case of error, the peer device receiving this event should initiate disconnection of the L2CAP channel by invoking CyBle_L2capDisconnectReq () function.

***CYBLE_EVT_L2CAP_CBFC_DATA_WRITE_IND*** This event is used to inform application of data transmission completion over L2CAP CBFC channel. Event parameter is of type 'CYBLE_L2CAP_CBFC_DATA_WRITE_PARAM_T'. L2CAP CBFC application must wait for this event before

transmitting the next CBFC L2CAP data. Application can send next data only when CYBLE_EVT_L2CAP_CBFC_DATA_WRITE_IND event is received for previous sent data and CYBLE_EVT_STACK_BUSY_STATUS is received with status CYBLE_STACK_STATE_FREE.

This event will be deprecated in future. It is only kept for backward compatibility. It is not recommended to be used by new design

***CYBLE_EVT_QUAL_SMP_PAIRING_REQ_RSP*** Tester to manipulate pairing request or response PDU. Event parameter is a pointer to 1 bytes data. Tester can manipulate the bits of the byte

***CYBLE_EVT_QUAL_SMP_LOCAL_PUBLIC_KEY*** Tester to manipulate local Public Key. Event parameter is a pointer to local public key of size 64 Bytes. Tester can manipulate the bits/bytes

***CYBLE_EVT_QUAL_SMP_PAIRING_FAILED_CMD*** Tester to assign pairing failed error code. Event parameter is a pointer to 16 bits value. Tester should assign error code to lower bits

***CYBLE_EVT_PENDING_FLASH_WRITE*** This event is used to inform application that flash write is pending Stack internal data structures are modified and require backup.

***CYBLE_EVT_LE_PING_AUTH_TIMEOUT*** LE PING Authentication Timeout Event to indicate that peer device has not responded with the valid MIC packet within the application configured ping authentication time.

***CYBLE_EVT_HCI_PKT*** This event is used to inform application that an HCI event has been received from controller. Event parameter is of type 'CYBLE_HCI_PKT_PARAMS_T'

This event will only be trigger when user register for SoftTransport by calling CyBle_HciSoftTransportEnable()

***CYBLE_EVT_FLASH_CORRUPT*** This event is used to inform application that bonding information stored in flash is corrupted.

***CYBLE_EVT_MAX*** Maximum value of CYBLE_EVENT_T type

**enum CYBLE_HCI_ERROR_T**

HCI Error codes defined by BT Spec

**Enumerator**

***CYBLE_HCI_COMMAND_SUCCEEDED*** Command success

***CYBLE_HCI_UNKNOWN_HCI_COMMAND_ERROR*** Unknown HCI Command

***CYBLE_HCI_NO_CONNECTION_ERROR*** Unknown Connection Identifier

***CYBLE_HCI_HARDWARE_FAILURE_ERROR*** Hardware Failure

***CYBLE_HCI_PAGE_TIMEOUT_ERROR*** Page Timeout

***CYBLE_HCI_AUTHENTICATION_FAILURE_ERROR*** Authentication Failure

***CYBLE_HCI_KEY_MISSING_ERROR*** PIN or Key Missing

***CYBLE_HCI_MEMORY_FULL_ERROR*** Memory Capacity Exceeded

***CYBLE_HCI_CONNECTION_TIMEOUT_ERROR*** Connection Timeout

***CYBLE_HCI_MAX_NUMBER_OF_CONNECTIONS_ERROR*** Connection Limit Exceeded

***CYBLE_HCI_MAX_SCO_CONNECTIONS_REACHED_ERROR*** Synchronous Connection Limit to a Device Exceeded

***CYBLE_HCI_ACL_CONNECTION_EXISTS_ERROR*** ACL Connection Already Exists

***CYBLE_HCI_COMMAND_DISALLOWED_ERROR*** Command Disallowed

***CYBLE_HCI_HOST_REJECTED_LIMITED_RESOURCES_ERROR*** Connection Rejected due to Limited resources

***CYBLE_HCI_HOST_REJECTED_SECURITY_REASONS_ERROR*** Connection Rejected due to Security Reasons

***CYBLE_HCI_HOST_REJECTED_PERSONAL_DEVICE_ERROR*** Connection Rejected due to Unacceptable BD_ADDR

***CYBLE_HCI_CONNECTION_ACCEPT_TIMEOUT_EXCEEDED_ERROR*** Connection Accept Timeout Exceeded

***CYBLE_HCI_UNSUPPORTED_FEATURE_OR_PARAMETER_ERROR*** Unsupported Feature or Parameter Value

***CYBLE_HCI_INVALID_HCI_COMMAND_PARAMETERS_ERROR*** Invalid HCI Command Parameters

***CYBLE_HCI_CONNECTION_TERMINATED_USER_ERROR*** remote user terminated Connection

***CYBLE_HCI_CONNECTION_TERMINATED_LOW_RESOURCES_ERROR*** Remote Device Terminated Connection due to Low Resources

***CYBLE_HCI_CONNECTION_TERMINATED_POWER_OFF_ERROR*** Remote Device Terminated Connection due to Power Off

***CYBLE_HCI_CONNECTION_TERMINATED_LOCAL_HOST_ERROR*** Connection Terminated By Local Host

***CYBLE_HCI_REPEATED_ATTEMPTS_ERROR*** Repeated Attempts

***CYBLE_HCI_PAIRING_NOT_ALLOWED_ERROR*** Pairing Not Allowed

***CYBLE_HCI_UNKNOWN_LMP_PDU_ERROR*** Unknown LMP PDU

***CYBLE_HCI_UNSUPPORTED_REMOTE_FEATURE_ERROR*** Unsupported Remote Feature

***CYBLE_HCI_SCO_OFFSET_REJECTED_ERROR*** SCO Offset Rejected

***CYBLE_HCI_SCO_INTERVAL_REJECTED_ERROR*** SCO Interval Rejected

***CYBLE_HCI_SCO_AIR_MODE_REJECTED_ERROR*** SCO Air Mode Rejected

***CYBLE_HCI_INVALID_LMP_PARAMETERS_ERROR*** Invalid LMP Parameters

***CYBLE_HCI_INVALID_LL_PARAMETERS_ERROR*** Invalid LL Parameters

***CYBLE_HCI_UNSPECIFIED_ERROR*** Unspecified error

***CYBLE_HCI_UNSUPPORTED_PARAMETER_VALUE_ERROR*** Unsupported LMP Parameter Value

***CYBLE_HCI_UNSUPPORTED_LL_PARAMETER_VALUE_ERROR*** Unsupported LL Parameter Value

***CYBLE_HCI_SWITCH_NOT_ALLOWED_ERROR*** Role Change Not Allowed

***CYBLE_HCI_LMP_RESPONSE_TIMEOUT_ERROR*** LMP Response Timeout

***CYBLE_HCI_LL_RESPONSE_TEMEOUT_ERROR*** LL Response Timeout

***CYBLE_HCI_LMP_ERROR_TRANSACTION_COLLISION_ERROR*** LMP Error Transaction Collision

***CYBLE_HCI_PDU_NOT_ALLOWED_ERROR*** LMP PDU Not Allowed

***CYBLE_HCI_ENCRYPTION_MODE_NOT_ACCEPTABLE_ERROR*** Encryption Mode Not Acceptable

***CYBLE_HCI_UNIT_KEY_USED_ERROR*** Link Key cannot be changed

***CYBLE_HCI_QOS_NOT_SUPPORTED_ERROR*** Requested QoS Not Supported

***CYBLE_HCI_INSTANT_PASSED_ERROR*** Instant Passed

***CYBLE_HCI_PAIRING_WITH_UNIT_KEY_NOT_SUPPPORTED_ERROR*** Pairing with unit key not supported

***CYBLE_HCI_DIFFERENT_TRANSACTION_COLLISION*** Different Transaction Collision

***CYBLE_HCI_QOS_UNACCEPTABLE_PARAMETER*** QoS Unacceptable parameter

***CYBLE_HCI_QOS_REJECTED_ERROR*** QoS Rejected

***CYBLE_HCI_CHANNEL_CLASSIFICATION_NOT_SUPPORTED*** Channel Classification Not Supported

***CYBLE_HCI_INSUFFICIENT_SECURITY*** Insufficient security

***CYBLE_HCI_PARAMETER_OUT_OF_MANDATORY_RANGE*** parameter out of mandatory range

***CYBLE_HCI_ROLE_SWITCH_PENDING*** Role Switch Pending

***CYBLE_HCI_RESERVED_SLOT_VIOLATION*** Reserved Slot violate

***CYBLE_HCI_ROLE_SWITCH_FAILED*** Role switch failed

***CYBLE_HCI_EXTENDED_INQUIRY_RESPONSE_TOO_LARGE*** Extended inquiry response too large

***CYBLE_HCI_SECURE_SIMPLE_PAIRING_NOT_SUPPORTED_BY_HOST*** secure simple pairing not supported by host

***CYBLE_HCI_HOST_BUSY_PAIRING*** host busy pairing

***CYBLE_HCI_CONNECTION_REJECTED_DUE_TO_NO_SUITABLE_CHANNEL_FOUND*** Connection Rejected due to No suitable channel found

***CYBLE_HCI_CONTROLLER_BUSY*** Controller busy

***CYBLE_HCI_UNACCEPTABLE_CONNECTION_INTERVAL*** unacceptable connection interval

***CYBLE_HCI_UNACCEPTABLE_CONNECTION_PARAMETERS*** unacceptable connection parameters

***CYBLE_HCI_DIRECTED_ADVERTISING_TIMEOUT*** Directed Advertising Timeout

***CYBLE_HCI_CONNECTION_TERMINATED_DUE_TO_MIC_FAILURE*** Connection Terminated due to MIC Failure

***CYBLE_HCI_CONNECTION_FAILED_TO_BE_ESTABLISHED*** Connection failed to be established

***CYBLE_HCI_MAC_CONNECTION_FAILED*** MAC connection failed

***CYBLE_HCI_COARSE_CLOCK_ADJ_REJECTED_TRY_USING_CLOCK_DRAGGING*** Coarse Clock Adjustment Rejected but will try to adjust using clock

***CYBLE_HCI_LAST_ENTRY_BLUETOOTH_ERROR_CODE*** INVALID HCI ERROR CODE

## BLE Common Definitions and Data Structures

### Description

Contains definitions and structures that are common to all BLE common APIs. Note that some of these are also used in Service-specific APIs.

### Data Structures

- struct CYBLE_BLESS_PWR_IN_DB_T
- struct CYBLE_MEMORY_REQUEST_T
- struct CYBLE_BLESS_CLK_CFG_PARAMS_T
- struct CYBLE_STACK_LIB_VERSION_T
- struct CYBLE_STK_APP_DATA_BUFF_T
- struct CYBLE_DLE_CONFIG_PARAM_T
- struct CYBLE_PRIVACY_1_2_CONFIG_PARAM_T
- struct CYBLE_STACK_CONFIG_PARAM_T
- struct CYBLE_AES_CMAC_GENERATE_PARAM_T
- struct CYBLE_BLESS_EVENT_PARAM_T
- struct CYBLE_TRANSMITTER_TEST_PARAMS_T
- struct CYBLE_HCI_PKT_PARAMS_T
- struct CYBLE_UUID128_T
- union CYBLE_UUID_T
- struct CYBLE_CONN_HANDLE_T

### Typedefs

- typedef void(* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)
- typedef void(* CYBLE_APP_CB_T) (uint8 event, void *evParam)
- typedef void(* CYBLE_BLESS_CB_T) (uint32 event, void *evParam)
- typedef void(* AES_CMAC_APPL_CB) (void)

- typedef uint16 CYBLE_UUID16

## Enumerations

- enum CYBLE_STATE_T
- enum CYBLE_CLIENT_STATE_T
- enum CYBLE_API_RESULT_T
- enum CYBLE_LP_MODE_T
- enum CYBLE_BLESS_STATE_T
- enum CYBLE_BLESS_PWR_LVL_T
- enum CYBLE_BLESS_PHY_CH_GRP_ID_T
- enum CYBLE_BLESS_WCO_SCA_CFG_T
- enum CYBLE_BLESS_ECO_CLK_DIV_T
- enum CYBLE_PROTOCOL_REQ_T
- enum CYBLE_PKT_PAYLOAD_T
- enum CYBLE_HCI_PKT_TYPE_T
- enum CYBLE_TO_REASON_CODE_T

## Data Structure Documentation

### struct CYBLE_BLESS_PWR_IN_DB_T

**Data Fields**

- CYBLE_BLESS_PWR_LVL_T blePwrLevelInDbm
- CYBLE_BLESS_PHY_CH_GRP_ID_T bleSsChId

**Field Documentation**

**CYBLE_BLESS_PWR_LVL_T CYBLE_BLESS_PWR_IN_DB_T::blePwrLevelInDbm**

Output Power level

**CYBLE_BLESS_PHY_CH_GRP_ID_T CYBLE_BLESS_PWR_IN_DB_T::bleSsChId**

Channel group ID for which power level is to be read/written

### struct CYBLE_MEMORY_REQUEST_T

**Data Fields**

- CYBLE_PROTOCOL_REQ_T request
- uint8 allocFree
- void * configMemory

**Field Documentation**

**CYBLE_PROTOCOL_REQ_T CYBLE_MEMORY_REQUEST_T::request**

Protocol Request type

**uint8 CYBLE_MEMORY_REQUEST_T::allocFree**

event parameter is generated to allocate memory or to free up previously allocated memory CYBLE_ALLOC_MEMORY (0) = to allocate memory for request type, CYBLE_FREE_MEMORY (1) = free previously allocated memory for the request type

**void* CYBLE_MEMORY_REQUEST_T::configMemory**

This is an output parameter which application needs to fill and pass to BLE Stack as per below table:

| request | memory |
|---|---|
| CYBLE_PREPARED_WRITE_REQUEST | CYBLE_PREPARE_WRITE_REQUEST_MEMORY_T |

**struct CYBLE_BLESS_CLK_CFG_PARAMS_T**

**Data Fields**

- CYBLE_BLESS_WCO_SCA_CFG_T bleLlSca
- CYBLE_BLESS_ECO_CLK_DIV_T bleLlClockDiv
- uint16 ecoXtalStartUpTime

**Field Documentation**

**CYBLE_BLESS_WCO_SCA_CFG_T CYBLE_BLESS_CLK_CFG_PARAMS_T::bleLlSca**
Sleep Clock accuracy in PPM, 32Khz Cycles

**CYBLE_BLESS_ECO_CLK_DIV_T CYBLE_BLESS_CLK_CFG_PARAMS_T::bleLlClockDiv**
Link Layer clock divider

**uint16 CYBLE_BLESS_CLK_CFG_PARAMS_T::ecoXtalStartUpTime**
ECO crystal startup time in multiple of 62.5us

**struct CYBLE_STACK_LIB_VERSION_T**

**Data Fields**

- uint16 majorVersion
- uint16 minorVersion
- uint16 patch
- uint16 buildNumber

**Field Documentation**

**uint16 CYBLE_STACK_LIB_VERSION_T::majorVersion**
The major version of the library

**uint16 CYBLE_STACK_LIB_VERSION_T::minorVersion**
The minor version of the library

**uint16 CYBLE_STACK_LIB_VERSION_T::patch**
The patch number of the library

**uint16 CYBLE_STACK_LIB_VERSION_T::buildNumber**
The build number of the library

**struct CYBLE_STK_APP_DATA_BUFF_T**

**Data Fields**

- uint16 bufferSize
- uint8 bufferUnits

**Field Documentation**

**uint16 CYBLE_STK_APP_DATA_BUFF_T::bufferSize**
Size of the buffer chunk

**uint8 CYBLE_STK_APP_DATA_BUFF_T::bufferUnits**
Number of the buffers units of 'bufferSize'

**struct CYBLE_DLE_CONFIG_PARAM_T**

**Data Fields**

- uint16 dleMaxTxCapability
- uint16 dleMaxRxCapability
- uint8 dleNumTxBuffer

**Field Documentation**

**uint16 CYBLE_DLE_CONFIG_PARAM_T::dleMaxTxCapability**
DLE max Tx capability

**uint16 CYBLE_DLE_CONFIG_PARAM_T::dleMaxRxCapability**
DLE max Rx capability

**uint8 CYBLE_DLE_CONFIG_PARAM_T::dleNumTxBuffer**
DLE number of Tx buffers

### struct CYBLE_PRIVACY_1_2_CONFIG_PARAM_T

**Data Fields**

- uint8 resolvingListSize

**Field Documentation**

**uint8 CYBLE_PRIVACY_1_2_CONFIG_PARAM_T::resolvingListSize**
Maximum number of possible entries in resolving list

### struct CYBLE_STACK_CONFIG_PARAM_T

**Data Fields**

- CYBLE_DLE_CONFIG_PARAM_T * dleConfig
- CYBLE_PRIVACY_1_2_CONFIG_PARAM_T * privacyConfig
- uint16 feature_mask

**Field Documentation**

**CYBLE_DLE_CONFIG_PARAM_T* CYBLE_STACK_CONFIG_PARAM_T::dleConfig**
Configuration parameter for DLE feature

**CYBLE_PRIVACY_1_2_CONFIG_PARAM_T* CYBLE_STACK_CONFIG_PARAM_T::privacyConfig**
Configuration parameter for LL Privacy feature

**uint16 CYBLE_STACK_CONFIG_PARAM_T::feature_mask**
The feature set mask used to control usage of specified feature in BLE stack. If a feature is not selected then associated parameter pointer can be NULL.

### struct CYBLE_AES_CMAC_GENERATE_PARAM_T

**Data Fields**

- uint8 * buffer
- uint16 size
- uint8 * key
- uint8 * mac
- AES_CMAC_APPL_CB appl_callback

**Field Documentation**

**uint8* CYBLE_AES_CMAC_GENERATE_PARAM_T::buffer**
pointer to message for which AES CMAC has to be calculated, LSB should be first

**uint16 CYBLE_AES_CMAC_GENERATE_PARAM_T::size**
size of the message buffer

**uint8* CYBLE_AES_CMAC_GENERATE_PARAM_T::key**
AES CMAC 128-bit Key, LSB should be first

**uint8* CYBLE_AES_CMAC_GENERATE_PARAM_T::mac**
output-parameter, Buffer to hold generated MAC of 16 bytes. Output is LSB first

**AES_CMAC_APPL_CB CYBLE_AES_CMAC_GENERATE_PARAM_T::appl_callback**
Callback to notify when the AES-CMAC generation is completed. Once this callback is called, check for the output-parameter, which contains generated cmac

**struct CYBLE_BLESS_EVENT_PARAM_T**

**Data Fields**

- uint32 BlessStateMask
- CYBLE_BLESS_CB_T bless_evt_app_cb

**Field Documentation**

**uint32 CYBLE_BLESS_EVENT_PARAM_T::BlessStateMask**
Bless state Event mask

**CYBLE_BLESS_CB_T CYBLE_BLESS_EVENT_PARAM_T::bless_evt_app_cb**
User callback function

**struct CYBLE_TRANSMITTER_TEST_PARAMS_T**

**Data Fields**

- uint8 tx_frequency
- uint8 length_of_test_data
- CYBLE_PKT_PAYLOAD_T packet_payload

**Field Documentation**

**uint8 CYBLE_TRANSMITTER_TEST_PARAMS_T::tx_frequency**
"N = (F  2402) / 2 Range: 0x00  0x27. Frequency Range : 2402 MHz to 2480 MHz"

**uint8 CYBLE_TRANSMITTER_TEST_PARAMS_T::length_of_test_data**
length of the test data

**CYBLE_PKT_PAYLOAD_T CYBLE_TRANSMITTER_TEST_PARAMS_T::packet_payload**
payload sequence

**struct CYBLE_HCI_PKT_PARAMS_T**

**Data Fields**

- CYBLE_HCI_PKT_TYPE_T pkt_type
- uint16 length
- uint8 * buffer

**Field Documentation**

**CYBLE_HCI_PKT_TYPE_T CYBLE_HCI_PKT_PARAMS_T::pkt_type**
HCI packet type

**uint16 CYBLE_HCI_PKT_PARAMS_T::length**
length of the command

**uint8* CYBLE_HCI_PKT_PARAMS_T::buffer**
Command buffer

**struct CYBLE_UUID128_T**

**Data Fields**

- uint8 value [16u]

**Field Documentation**

**uint8 CYBLE_UUID128_T::value[16u]**
128 Bit UUID

**union CYBLE_UUID_T**

**Data Fields**

- CYBLE_UUID16 uuid16
- CYBLE_UUID128_T uuid128

**Field Documentation**

**CYBLE_UUID16 CYBLE_UUID_T::uuid16**
16 Bit UUID

**CYBLE_UUID128_T CYBLE_UUID_T::uuid128**
128 Bit UUID

**struct CYBLE_CONN_HANDLE_T**

**Data Fields**

- uint8 bdHandle
- uint8 attId

**Field Documentation**

**uint8 CYBLE_CONN_HANDLE_T::bdHandle**
Identifies the peer device(s) bonded or in current connection. Stack supports CYBLE_GAP_MAX_BONDED_DEVICE+1 devices. first device connected is assigned value CYBLE_GAP_MAX_BONDED_DEVICE. If previous device is bonded then current device will be assigned value CYBLE_GAP_MAX_BONDED_DEVICE-1, else CYBLE_GAP_MAX_BONDED_DEVICE.

**uint8 CYBLE_CONN_HANDLE_T::attId**
Identifies the ATT Instance. Current implementation supports only one att instance (0) due to availability of only on fixed channel for att. This parameter is introduced as part of connection handle to keep the interface unchanged event if new Bluetooth spec defines more fixed channels for ATT payload.

## Typedef Documentation

**typedef void(* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)**
Event callback function prototype to receive events from BLE component

**typedef void(* CYBLE_APP_CB_T) (uint8 event, void *evParam)**
event callback function prototype to receive events from stack

**typedef void(* CYBLE_BLESS_CB_T) (uint32 event, void *evParam)**
event callback function prototype to receive Bless State events from stack

**typedef void(* AES_CMAC_APPL_CB) (void)**
Application callback function prototype to notify when AES CMAC generation is completed

**typedef uint16 CYBLE_UUID16**
GATT 16 Bit UUID

## Enumeration Type Documentation

**enum CYBLE_STATE_T**

Event handler state machine type

**Enumerator**

***CYBLE_STATE_STOPPED*** BLE is turned off

***CYBLE_STATE_INITIALIZING*** Initializing state

***CYBLE_STATE_CONNECTED*** Peer device is connected

***CYBLE_STATE_ADVERTISING*** Advertising process

***CYBLE_STATE_SCANNING*** Scanning process

***CYBLE_STATE_CONNECTING*** Connecting

***CYBLE_STATE_DISCONNECTED*** Essentially idle state

**enum CYBLE_CLIENT_STATE_T**

Client State type

**Enumerator**

***CYBLE_CLIENT_STATE_CONNECTED*** Server device is connected

***CYBLE_CLIENT_STATE_SRVC_DISCOVERING*** Server services are being discovered

***CYBLE_CLIENT_STATE_INCL_DISCOVERING*** Server included services are being discovered

***CYBLE_CLIENT_STATE_CHAR_DISCOVERING*** Server characteristics are being discovered

***CYBLE_CLIENT_STATE_DESCR_DISCOVERING*** Server char. descriptors are being discovered

***CYBLE_CLIENT_STATE_DISCOVERED*** Server is discovered

***CYBLE_CLIENT_STATE_DISCONNECTING*** Server is disconnecting

***CYBLE_CLIENT_STATE_DISCONNECTED_DISCOVERED*** Server is disconnected but discovered

***CYBLE_CLIENT_STATE_DISCONNECTED*** Essentially initial client state

**enum CYBLE_API_RESULT_T**

Common error codes received as API result

**Enumerator**

***CYBLE_ERROR_OK*** No Error occurred

***CYBLE_ERROR_INVALID_PARAMETER*** At least one of the input parameters is invalid

***CYBLE_ERROR_INVALID_OPERATION*** Operation is not permitted

***CYBLE_ERROR_MEMORY_ALLOCATION_FAILED*** An internal error occurred in the stack

***CYBLE_ERROR_INSUFFICIENT_RESOURCES*** Insufficient resources to perform requested operation

***CYBLE_ERROR_OOB_NOT_AVAILABLE*** OOB data not available

***CYBLE_ERROR_NO_CONNECTION*** Connection is required to perform requested operation. Connection not present

***CYBLE_ERROR_NO_DEVICE_ENTITY*** No device entity to perform requested operation

***CYBLE_ERROR_REPEATED_ATTEMPTS*** Attempted repeat operation is not allowed

***CYBLE_ERROR_GAP_ROLE*** GAP role is incorrect

***CYBLE_ERROR_TX_POWER_READ*** Error reading TC power

***CYBLE_ERROR_BT_ON_NOT_COMPLETED*** BLE Initialization failed

***CYBLE_ERROR_SEC_FAILED*** Security operation failed

***CYBLE_ERROR_L2CAP_PSM_WRONG_ENCODING*** L2CAP PSM encoding is incorrect

***CYBLE_ERROR_L2CAP_PSM_ALREADY_REGISTERED*** L2CAP PSM has already been registered

***CYBLE_ERROR_L2CAP_PSM_NOT_REGISTERED*** L2CAP PSM has not been registered

***CYBLE_ERROR_L2CAP_CONNECTION_ENTITY_NOT_FOUND*** L2CAPconnection entity not found

***CYBLE_ERROR_L2CAP_CHANNEL_NOT_FOUND*** L2CAP channel not found

***CYBLE_ERROR_L2CAP_PSM_NOT_IN_RANGE*** Specified PSM is out of range

***CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE*** GATT DB error codes Invalid attribute handle

***CYBLE_ERROR_DEVICE_ALREADY_EXISTS*** Device cannot be added to whitelist as it has already been added

***CYBLE_ERROR_FLASH_WRITE_NOT_PERMITED*** Write to flash is not permitted

***CYBLE_ERROR_MIC_AUTH_FAILED*** MIC Authentication failure

***CYBLE_ERROR_HARDWARE_FAILURE*** Controller error codes. These come directly from controller (not host stack) Hardware Failure

***CYBLE_ERROR_UNSUPPORTED_FEATURE_OR_PARAMETER_VALUE*** Unsupported feature or parameter value

***CYBLE_ERROR_FLASH_WRITE*** Error in flash Write

***CYBLE_ERROR_LL_SAME_TRANSACTION_COLLISION*** LL same transaction collision

***CYBLE_ERROR_CONTROLLER_BUSY*** Controller Busy

***CYBLE_ERROR_MAX*** All other errors not covered in the above list map to this error code

***CYBLE_ERROR_NTF_DISABLED*** Characteristic notifications disabled

***CYBLE_ERROR_IND_DISABLED*** Characteristic indications disabled

***CYBLE_ERROR_INVALID_STATE*** The state is not valid for current operation

***CYBLE_ERROR_STACK_BUSY*** Stack is Busy

**enum CYBLE_LP_MODE_T**

BLE power modes

**Enumerator**

***CYBLE_BLESS_ACTIVE*** Link Layer engine and Digital modem clocked from ECO. The CPU can access the BLE Sub-System (BLESS) registers. This mode collectively denotes Tx Mode, Rx Mode, and Idle mode of BLESS.

***CYBLE_BLESS_SLEEP*** The clock to the link layer engine and digital modem is gated. The ECO continues to run to maintain the link layer timing.

***CYBLE_BLESS_DEEPSLEEP*** The ECO is stopped and WCO is used to maintain link layer timing. RF transceiver is turned off completely to reduce leakage current. BLESS logic is kept powered ON from the SRSS deep sleep regulator for retention.

***CYBLE_BLESS_HIBERNATE*** External power is available but all internal LDOs are turned off.

***CYBLE_BLESS_INVALID*** Invalid mode

**enum CYBLE_BLESS_STATE_T**

BLESS Power enum reflecting power states supported by BLESS radio

**Enumerator**

***CYBLE_BLESS_STATE_ACTIVE*** BLESS state is ACTIVE

***CYBLE_BLESS_STATE_EVENT_CLOSE*** BLESS state is EVENT_CLOSE

***CYBLE_BLESS_STATE_SLEEP*** BLESS state is SLEEP

***CYBLE_BLESS_STATE_ECO_ON*** BLESS state is ECO_ON

***CYBLE_BLESS_STATE_ECO_STABLE*** BLESS state is ECO_STABLE

***CYBLE_BLESS_STATE_DEEPSLEEP*** BLESS state is DEEPSLEEP

***CYBLE_BLESS_STATE_HIBERNATE*** BLESS state is HIBERNATE

***CYBLE_BLESS_STATE_INVALID*** BLESS state is INVALID

**enum CYBLE_BLESS_PWR_LVL_T**

BLESS Power enum reflecting power level values supported by BLESS radio

**Enumerator**

***CYBLE_LL_PWR_LVL_NEG_18_DBM*** ABS PWR = -18dBm, PA_Gain = 0x01

***CYBLE_LL_PWR_LVL_NEG_12_DBM*** ABS PWR = -12dBm, PA_Gain = 0x02

***CYBLE_LL_PWR_LVL_NEG_6_DBM*** ABS PWR = -6dBm, PA_Gain = 0x03

***CYBLE_LL_PWR_LVL_NEG_3_DBM*** ABS PWR = -3dBm, PA_Gain = 0x04

***CYBLE_LL_PWR_LVL_NEG_2_DBM*** ABS PWR = -2dBm, PA_Gain = 0x05

***CYBLE_LL_PWR_LVL_NEG_1_DBM*** ABS PWR = -1dBm, PA_Gain = 0x06

***CYBLE_LL_PWR_LVL_0_DBM*** ABS PWR = 0dBm, PA_Gain = 0x07

***CYBLE_LL_PWR_LVL_3_DBM*** ABS PWR = 3dBm, PA_Gain = 0x07, PWR_GAIN level is same as 0 dBm, but the ABS_PWR is amplified and applied for both Connection and Advertising channel.

***CYBLE_LL_PWR_LVL_MAX*** ABS PWR = 3dBm, PA_Gain = 0x07

**enum CYBLE_BLESS_PHY_CH_GRP_ID_T**

BLE channel group ID

**Enumerator**

***CYBLE_LL_ADV_CH_TYPE*** Advertisement channel type

***CYBLE_LL_CONN_CH_TYPE*** Connection channel type

***CYBLE_LL_MAX_CH_TYPE*** Maximum value of CYBLE_BLESS_PHY_CH_GRP_ID_T type

**enum CYBLE_BLESS_WCO_SCA_CFG_T**

BLE WCO sleep clock accuracy configuration

**Enumerator**

***CYBLE_LL_SCA_251_TO_500_PPM*** SCA 251 to 500 PPM

***CYBLE_LL_SCA_151_TO_250_PPM*** SCA 151 to 250 PPM

***CYBLE_LL_SCA_101_TO_150_PPM*** SCA 101 to 150 PPM

***CYBLE_LL_SCA_076_TO_100_PPM*** SCA 076 to 100 PPM

***CYBLE_LL_SCA_051_TO_075_PPM*** SCA 051 to 075 PPM

***CYBLE_LL_SCA_031_TO_050_PPM*** SCA 031 to 050 PPM

***CYBLE_LL_SCA_021_TO_030_PPM*** SCA 021 to 030 PPM

***CYBLE_LL_SCA_000_TO_020_PPM*** SCA 000 to 020 PPM

***CYBLE_LL_SCA_IN_PPM_INVALID*** Invalid PPM

**enum CYBLE_BLESS_ECO_CLK_DIV_T**

BLE ECO clock divider

**Enumerator**

***CYBLE_LL_ECO_CLK_DIV_1*** Link Layer clock divider = 1

***CYBLE_LL_ECO_CLK_DIV_2*** Link Layer clock divider = 2

***CYBLE_LL_ECO_CLK_DIV_4*** Link Layer clock divider = 4

***CYBLE_LL_ECO_CLK_DIV_8*** Link Layer clock divider = 8

***CYBLE_LL_ECO_CLK_DIV_INVALID*** Invalid Link Layer clock divider

**enum CYBLE_PROTOCOL_REQ_T**

BLE Stack memory request type

**Enumerator**

***CYBLE_PREPARED_WRITE_REQUEST*** Memory requested for prepare write request

***CYBLE_INVALID_REQUEST*** Invalid request

**enum CYBLE_PKT_PAYLOAD_T**

DTM Payload sequence in SoC mode

**Enumerator**

***CYBLE_PAYLOAD_VAL_ZERO*** PRBS9 sequence '11111111100000111101 (in transmission order) as described in [Vol 6] Part F, Section 4.1.5

***CYBLE_PAYLOAD_VAL_ONE*** Repeated 11110000 (in transmission order) sequence as described in [Vol 6] Part F, Section 4.1.5

***CYBLE_PAYLOAD_VAL_TWO*** Repeated 10101010 (in transmission order) sequence as described in [Vol 6] Part F, Section 4.1.5

***CYBLE_PAYLOAD_VAL_THREE*** PRBS15 sequence as described in [Vol 6] Part F, Section 4.1.5

***CYBLE_PAYLOAD_VAL_FOUR*** Repeated 11111111 (in transmission order) sequence

***CYBLE_PAYLOAD_VAL_FIVE*** Repeated 00000000 (in transmission order) sequence

***CYBLE_PAYLOAD_VAL_SIX*** Repeated 00001111 (in transmission order) sequence

***CYBLE_PAYLOAD_VAL_SEVEN*** Repeated 01010101 (in transmission order) sequence

**enum CYBLE_HCI_PKT_TYPE_T**

HCI Packet type enum

**Enumerator**

***CYBLE_HCI_CMD_PKT_TYPE*** HCI Command packet type

***CYBLE_HCI_ACL_DATA_PKT_TYPE*** HCI ACL data packet type

***CYBLE_HCI_SYNC_DATA_PKT_TYPE*** HCI Synchronous packet type

***CYBLE_HCI_EVENT_PKT_TYPE*** HCI Event packet type

**enum CYBLE_TO_REASON_CODE_T**

BLE stack timeout. This is received with CYBLE_EVT_TIMEOUT event It is application's responsibility to disconnect or keep the channel on depends on type of timeouts. i.e. GATT procedure timeout: Application may choose to disconnect.

**Enumerator**

***CYBLE_GAP_ADV_MODE_TO*** Advertisement time set by application has expired

***CYBLE_GAP_SCAN_TO*** Scan time set by application has expired

***CYBLE_GATT_RSP_TO*** GATT procedure timeout

***CYBLE_GENERIC_TO*** Generic timeout

# BLE Service-Specific APIs

## Description

This section describes BLE Service-specific APIs. The Service APIs are only included in the design if the Service is added to the selected Profile in the component GUI. These are interfaces for the BLE application to use during BLE connectivity. The service specific APIs internally use the BLE Stack APIs to achieve the Service use case.

Refer to the Special Interest Group Web Site for links to the latest specifications and other documentation.

Many of the APIs will generate Service-specific events. The events are also used in the Service-specific callback functions. These are documented in BLE Service-Specific Events.

## Modules

- BLE Service-Specific Events
  *The BLE stack generates service-specific events to notify the application that a service specific status change needs attention. For general stack events, refer to BLE Common Events.*
- Apple Notification Center Service (ANCS)
  *The Apple Notification Center Service provides iOS notifications from Apple devices for accessories.*
- Alert Notification Service (ANS)
  *The Alert Notification Service exposes alert information in a device.*
- Automation IO Service (AIOS)
  *The Automation IO Service enables a device to connect and interact with an Automation IO Module (IOM) in order to access digital and analog signals.*
- Battery Service (BAS)
  *The Battery Service exposes the battery level of a single battery or set of batteries in a device.*
- Body Composition Service (BCS)
  *The Body Composition Service exposes data related to body composition from a body composition analyzer (Server) intended for consumer healthcare as well as sports/fitness applications.*
- Blood Pressure Service (BLS)
  *The Blood Pressure Service exposes blood pressure and other data related to a non-invasive blood pressure monitor for consumer and professional healthcare applications.*
- Bond Management Service (BMS)
  *The Bond Management Service defines how a peer Bluetooth device can manage the storage of bond information, especially the deletion of it, on the Bluetooth device supporting this service.*
- Continuous Glucose Monitoring Service (CGMS)
  *The Continuous Glucose Monitoring Service exposes glucose measurement and other data related to a personal CGM sensor for healthcare applications.*
- Cycling Power Service (CPS)
  *The Cycling Power Service (CPS) exposes power- and force-related data and optionally speed- and cadence-related data from a Cycling Power sensor (GATT Server) intended for sports and fitness applications.*
- Cycling Speed and Cadence Service (CSCS)
  *The Cycling Speed and Cadence (CSC) Service exposes speed-related data and/or cadence-related data while using the Cycling Speed and Cadence sensor (Server).*
- Current Time Service (CTS)
  *The Current Time Service defines how a Bluetooth device can expose time information to other Bluetooth devices.*
- Device Information Service (DIS)
  *The Device Information Service exposes manufacturer and/or vendor information about a device.*

- Environmental Sensing Service (ESS)
  *The Environmental Sensing Service exposes measurement data from an environmental sensor intended for sports and fitness applications.*
- Glucose Service (GLS)
  *The Glucose Service exposes glucose and other data related to a personal glucose sensor for consumer healthcare applications and is not designed for clinical use.*
- HID Service (HIDS)
  *The HID Service exposes data and associated formatting for HID Devices and HID Hosts.*
- Heart Rate Service (HRS)
  *The Heart Rate Service exposes heart rate and other data related to a heart rate sensor intended for fitness applications.*
- HTTP Proxy Service (HPS)
  *The HTTP Proxy Service allows a Client device, typically a sensor, to communicate with a Web Server through a gateway device.*
- Health Thermometer Service (HTS)
  *The Health Thermometer Service exposes temperature and other data related to a thermometer used for healthcare applications.*
- Immediate Alert Service (IAS)
  *The Immediate Alert Service exposes a control point to allow a peer device to cause the device to immediately alert.*
- Indoor Positioning Service (IPS)
  *The Indoor Positioning exposes coordinates and other location related information via an advertisement or indicates that the device address can be used for location look-up, enabling mobile devices to find their position.*
- Link Loss Service (LLS)
  *The Link Loss Service uses the Alert Level Characteristic to cause an alert in the device when the link is lost.*
- Location and Navigation Service (LNS)
  *The Location and Navigation Service exposes location and navigation-related data from a Location and Navigation sensor (Server) intended for outdoor activity applications.*
- Next DST Change Service (NDCS)
  *The Next DST Change Service enables a BLE device that has knowledge about the next occurrence of a DST change to expose this information to another Bluetooth device. The Service uses the "Time with DST" Characteristic and the functions exposed in this Service are used to interact with that Characteristic.*
- Phone Alert Status Service (PASS)
  *The Phone Alert Status Service uses the Alert Status Characteristic and Ringer Setting Characteristic to expose the phone alert status and uses the Ringer Control Point Characteristic to control the phone's ringer into mute or enable.*
- Pulse Oximeter Service (PLXS)
  *The Pulse Oximeter (PLX) Service exposes pulse oximetry data related to a non-invasive pulse oximetry sensor for consumer and professional healthcare applications.*
- Running Speed and Cadence Service (RSCS)
  *The Running Speed and Cadence (RSC) Service exposes speed, cadence and other data related to fitness applications such as the stride length and the total distance the user has travelled while using the Running Speed and Cadence sensor (Server).*
- Reference Time Update Service (RTUS)
  *The Reference Time Update Service enables a Bluetooth device that can update the system time using the reference time such as a GPS receiver to expose a control point and expose the accuracy (drift) of the local system time compared to the reference time source.*

- Scan Parameters Service (ScPS)
  *The Scan Parameters Service enables a Server device to expose a Characteristic for the GATT Client to write its scan interval and scan window on the Server device, and enables a Server to request a refresh of the GATT Client scan interval and scan window.*
- TX Power Service (TPS)
  *The Tx Power Service uses the Tx Power Level Characteristic to expose the current transmit power level of a device when in a connection.*
- User Data Service (UDS)
  *The User Data Service exposes user-related data in the sports and fitness environment. This allows remote access and update of user data by a Client as well as the synchronization of user data between a Server and a Client.*
- Wireless Power Transfer Service (WPTS)
  *The Wireless Power Transfer Service enables communication between Power Receiver Unit and Power Transmitter Unit in the Wireless Power Transfer systems.*
- Weight Scale Service (WSS)
  *The Weight Scale Service exposes weight and related data from a weight scale (Server) intended for consumer healthcare as well as sports/fitness applications.*
- Custom Service
  *This section contains the description of structs used for Custom Services.*

# BLE Service-Specific Events

## Description

The BLE stack generates service-specific events to notify the application that a service specific status change needs attention. For general stack events, refer to BLE Common Events.

## Enumerations

- enum CYBLE_EVT_T

## Enumeration Type Documentation

**enum CYBLE_EVT_T**

Service specific events

**Enumerator**

***CYBLE_EVT_GATTS_INDICATION_ENABLED*** GATT Server - Indications for GATT Service's "Service Changed" Characteristic were enabled. The parameter of this event is a structure of CYBLE_GATTS_WRITE_REQ_PARAM_T type.

***CYBLE_EVT_GATTS_INDICATION_DISABLED*** GATT Server - Indications for GATT Service's "Service Changed" Characteristic were disabled. The parameter of this event is a structure of CYBLE_GATTS_WRITE_REQ_PARAM_T type.

***CYBLE_EVT_GATTC_INDICATION*** GATT Client - GATT Service's "Service Changed" Characteristic Indications were received. The parameter of this event is a structure of CYBLE_GATTC_HANDLE_VALUE_IND_PARAM_T type.

***CYBLE_EVT_GATTC_SRVC_DISCOVERY_FAILED*** GATT Client - Service discovery procedure failed. This event may be generated on calling CyBle_GattcDiscoverAllPrimaryServices(). No parameters passed for this event.

***CYBLE_EVT_GATTC_INCL_DISCOVERY_FAILED*** GATT Client - Discovery of included services failed. This event may be generated on calling CyBle_GattcFindIncludedServices(). No parameters passed for this event.

***CYBLE_EVT_GATTC_CHAR_DISCOVERY_FAILED*** GATT Client - Discovery of service's characteristics failed. This event may be generated on calling CyBle_GattcDiscoverAllCharacteristics() or CyBle_GattcReadUsingCharacteristicUuid(). No parameters passed for this event.

***CYBLE_EVT_GATTC_DESCR_DISCOVERY_FAILED*** GATT Client - Discovery of service's characteristics failed. This event may be generated on calling CyBle_GattcDiscoverAllCharacteristicDescriptors(). No parameters passed for this event.

***CYBLE_EVT_GATTC_SRVC_DUPLICATION*** GATT Client - Duplicate service record was found during server device discovery. The parameter of this event is a structure of uint16 (UUID16) type.

***CYBLE_EVT_GATTC_CHAR_DUPLICATION*** GATT Client - Duplicate service's characteristic record was found during server device discovery. The parameter of this event is a structure of uint16 (UUID16) type.

***CYBLE_EVT_GATTC_DESCR_DUPLICATION*** GATT Client - Duplicate service's characteristic descriptor record was found during server device discovery. The parameter of this event is a structure of uint16 (UUID16) type.

***CYBLE_EVT_GATTC_SRVC_DISCOVERY_COMPLETE*** GATT Client - Service discovery procedure completed successfully. This event may be generated on calling CyBle_GattcDiscoverAllPrimaryServices(). No parameters passed for this event.

***CYBLE_EVT_GATTC_INCL_DISCOVERY_COMPLETE*** GATT Client - Included services discovery is completed successfully. This event may be generated on calling CyBle_GattcFindIncludedServices(). No parameters passed for this event.

***CYBLE_EVT_GATTC_CHAR_DISCOVERY_COMPLETE*** GATT Client - Discovery of service's characteristics discovery is completed successfully. This event may be generated on calling CyBle_GattcDiscoverAllCharacteristics() or CyBle_GattcReadUsingCharacteristicUuid(). No parameters passed for this event.

***CYBLE_EVT_GATTC_DISC_SKIPPED_SERVICE*** GATT Client - The service (not defined in the GATT database) was found during the server device discovery. The discovery procedure skips this service. This event parameter is a structure of the CYBLE_DISC_SRVC128_INFO_T type.

***CYBLE_EVT_GATTC_DISCOVERY_COMPLETE*** GATT Client - Discovery of remote device completed successfully. No parameters passed for this event.

***CYBLE_EVT_AIOSS_NOTIFICATION_ENABLED*** AIOS Server - Notifications for Automation Input Output Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_NOTIFICATION_DISABLED*** AIOS Server - Notifications for Automation Input Output Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_INDICATION_ENABLED*** AIOS Server - Indication for Automation Input Output Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_INDICATION_DISABLED*** AIOSS Server - Indication for Automation Input Output Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_INDICATION_CONFIRMED*** AIOS Server - Automation Input Output Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_CHAR_WRITE*** AIOS Server - Write Request for Automation Input Output Service Characteristic was received. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSS_DESCR_WRITE*** AIOSS Server - Write Request for Automation Input Output Service Characteristic Descriptor was received. The parameter of this event is a structure of CYBLE_AIOSS_DESCR_VALUE_T type.

***CYBLE_EVT_AIOSC_NOTIFICATION*** AIOS Client - Automation Input Output Characteristic Service Notification was received. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSC_INDICATION*** AIOS Client - Automation Input Output Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSC_READ_CHAR_RESPONSE*** AIOS Client - Read Response for Read Request for Automation Input Output Service Characteristic Value. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSC_WRITE_CHAR_RESPONSE*** AIOS Client - Write Response for Write Request for Automation Input Output Service Characteristic Value. The parameter of this event is a structure of CYBLE_AIOS_CHAR_VALUE_T type.

***CYBLE_EVT_AIOSC_READ_DESCR_RESPONSE*** AIOS Client - Read Response for Read Request for Automation Input Output Service Characteristic Descriptor Read Request. The parameter of this event is a structure of CYBLE_AIOS_DESCR_VALUE_T type.

***CYBLE_EVT_AIOSC_WRITE_DESCR_RESPONSE*** AIOS Client - Write Response for Write Request for Automation Input Output Service Client Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_AIOS_DESCR_VALUE_T type.

***CYBLE_EVT_AIOSC_ERROR_RESPONSE*** AIOS Client - Error Response for Write Request for Automation Input Output Service Characteristic Value. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSS_NOTIFICATION_ENABLED*** ANCS Server - Notifications for Apple Notification Center Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSS_NOTIFICATION_DISABLED*** ANCS Server - Notifications for Apple Notification Center Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSS_WRITE_CHAR*** ANCS Server - Write Request for Apple Notification Center Service Characteristic was received. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSC_NOTIFICATION*** ANCS Client - Apple Notification Center Characteristic Service Notification was received. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSC_WRITE_CHAR_RESPONSE*** ANCS Client - Write Response for Write Request for Apple Notification Center Service Characteristic Value. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANCSC_READ_DESCR_RESPONSE*** ANCS Client - Read Response for Read Request for Apple Notification Center Service Characteristic Descriptor Read Request. The parameter of this event is a structure of CYBLE_ANCS_DESCR_VALUE_T type.

***CYBLE_EVT_ANCSC_WRITE_DESCR_RESPONSE*** ANCS Client - Write Response for Write Request for Apple Notification Center Service Client Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_ANCS_DESCR_VALUE_T type.

***CYBLE_EVT_ANCSC_ERROR_RESPONSE*** ANCS Client - Error Response for Write Request for Apple Notification Center Service Characteristic Value. The parameter of this event is a structure of CYBLE_ANCS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSS_NOTIFICATION_ENABLED*** ANS Server - Notifications for Alert Notification Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSS_NOTIFICATION_DISABLED*** ANS Server - Notifications for Alert Notification Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSS_CHAR_WRITE*** ANS Server - Write Request for Alert Notification Service Characteristic was received. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSC_NOTIFICATION*** ANS Client - Alert Notification Characteristic Service Notification was received. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSC_READ_CHAR_RESPONSE*** ANS Client - Read Response for Alert Notification Service Characteristic Value. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSC_WRITE_CHAR_RESPONSE*** ANS Client - Write Response for Write Request for Alert Notification Service Characteristic Value. The parameter of this event is a structure of CYBLE_ANS_CHAR_VALUE_T type.

***CYBLE_EVT_ANSC_READ_DESCR_RESPONSE*** ANS Client - Read Response for Read Request for Alert Notification Service Characteristic Descriptor Read Request. The parameter of this event is a structure of CYBLE_ANS_DESCR_VALUE_T type.

***CYBLE_EVT_ANSC_WRITE_DESCR_RESPONSE*** ANS Client - Write Response for Write Request for Alert Notification Service Client Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_ANS_DESCR_VALUE_T type.

***CYBLE_EVT_BASS_NOTIFICATION_ENABLED*** BAS Server - Notifications for Battery Level Characteristic were enabled. The parameter of this event is a structure of CYBLE_BAS_CHAR_VALUE_T type.

***CYBLE_EVT_BASS_NOTIFICATION_DISABLED*** BAS Server - Notifications for Battery Level Characteristic were disabled. The parameter of this event is a structure of CYBLE_BAS_CHAR_VALUE_T type.

***CYBLE_EVT_BASC_NOTIFICATION*** BAS Client - Battery Level Characteristic Notification was received. The parameter of this event is a structure of CYBLE_BAS_CHAR_VALUE_T type.

***CYBLE_EVT_BASC_READ_CHAR_RESPONSE*** BAS Client - Read Response for Battery Level Characteristic Value. The parameter of this event is a structure of CYBLE_BAS_CHAR_VALUE_T type.

***CYBLE_EVT_BASC_READ_DESCR_RESPONSE*** BAS Client - Read Response for Battery Level Characteristic Descriptor Read Request. The parameter of this event is a structure of CYBLE_BAS_DESCR_VALUE_T type.

***CYBLE_EVT_BASC_WRITE_DESCR_RESPONSE*** BAS Client - Write Response for Battery Level Client Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_BAS_DESCR_VALUE_T type.

***CYBLE_EVT_BCSS_INDICATION_ENABLED*** BCS Server - Indication for Body Composition Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_BCS_CHAR_VALUE_T type.

***CYBLE_EVT_BCSS_INDICATION_DISABLED*** BCS Server - Indication for Body Composition Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_BCS_CHAR_VALUE_T type.

***CYBLE_EVT_BCSS_INDICATION_CONFIRMED*** BCS Server - Body Composition Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_BCS_CHAR_VALUE_T type.

***CYBLE_EVT_BCSC_INDICATION*** BCS Client - Body Composition Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_BCS_CHAR_VALUE_T type.

***CYBLE_EVT_BCSC_READ_CHAR_RESPONSE*** BCS Client - Read Response for Read Request of Body Composition Service Characteristic value. The parameter of this event is a structure of CYBLE_BCS_CHAR_VALUE_T type.

***CYBLE_EVT_BCSC_READ_DESCR_RESPONSE*** BCS Client - Read Response for Read Request of Body Composition Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_BCS_DESCR_VALUE_T type.

***CYBLE_EVT_BCSC_WRITE_DESCR_RESPONSE*** BCS Client - Write Response for Write Request of Body Composition Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_BCS_DESCR_VALUE_T type.

***CYBLE_EVT_BLSS_INDICATION_ENABLED*** BLS Server - Indication for Blood Pressure Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSS_INDICATION_DISABLED*** BLS Server - Indication for Blood Pressure Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSS_INDICATION_CONFIRMED*** BLS Server - Blood Pressure Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSS_NOTIFICATION_ENABLED*** BLS Server - Notifications for Blood Pressure Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type.

***CYBLE_EVT_BLSS_NOTIFICATION_DISABLED*** BLS Server - Notifications for Blood Pressure Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSC_INDICATION*** BLS Client - Blood Pressure Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSC_NOTIFICATION*** BLS Client - Blood Pressure Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSC_READ_CHAR_RESPONSE*** BLS Client - Read Response for Read Request of Blood Pressure Service Characteristic value. The parameter of this event is a structure of CYBLE_BLS_CHAR_VALUE_T type

***CYBLE_EVT_BLSC_READ_DESCR_RESPONSE*** BLS Client - Read Response for Read Request of Blood Pressure Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_BLS_DESCR_VALUE_T type

***CYBLE_EVT_BLSC_WRITE_DESCR_RESPONSE*** BLS Client - Write Response for Write Request of Blood Pressure Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_BLS_DESCR_VALUE_T type

***CYBLE_EVT_BMSS_WRITE_CHAR*** BMS Server - Write Request for Bond Management was received. The parameter of this event is a structure of CYBLE_BMS_CHAR_VALUE_T type.

***CYBLE_EVT_BMSC_READ_CHAR_RESPONSE*** BMS Client - Read Response for Read Request of Bond Management Service Characteristic value. The parameter of this event is a structure of CYBLE_BMS_CHAR_VALUE_T type

***CYBLE_EVT_BMSC_WRITE_CHAR_RESPONSE*** BMS Client - Write Response for Write Request of Bond Management Service Characteristic value. The parameter of this event is a structure of CYBLE_BMS_CHAR_VALUE_T type.

***CYBLE_EVT_BMSC_READ_DESCR_RESPONSE*** BMS Client - Read Response for Read Request of Bond Management Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_BMS_DESCR_VALUE_T type.

***CYBLE_EVT_CGMSS_INDICATION_ENABLED*** CGMS Server - Indication for Continuous Glucose Monitoring Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSS_INDICATION_DISABLED*** CGMS Server - Indication for Continuous Glucose Monitoring Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSS_INDICATION_CONFIRMED*** CGMS Server - Continuous Glucose Monitoring Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSS_NOTIFICATION_ENABLED*** CGMS Server - Notifications for Continuous Glucose Monitoring Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSS_NOTIFICATION_DISABLED*** CGMS Server - Notifications for Continuous Glucose Monitoring Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSS_WRITE_CHAR*** CGMS Server - Write Request for Continuous Glucose Monitoring Service was received. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSC_INDICATION*** CGMS Client - Continuous Glucose Monitoring Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSC_NOTIFICATION*** CGMS Client - Continuous Glucose Monitoring Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSC_READ_CHAR_RESPONSE*** CGMS Client - Read Response for Read Request of Continuous Glucose Monitoring Service Characteristic value. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSC_WRITE_CHAR_RESPONSE*** CGMS Client - Write Response for Write Request of Continuous Glucose Monitoring Service Characteristic value. The parameter of this event is a structure of CYBLE_CGMS_CHAR_VALUE_T type.

***CYBLE_EVT_CGMSC_READ_DESCR_RESPONSE*** CGMS Client - Read Response for Read Request of Continuous Glucose Monitoring Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_CGMS_DESCR_VALUE_T type.

***CYBLE_EVT_CGMSC_WRITE_DESCR_RESPONSE*** CGMS Client - Write Response for Write Request of Continuous Glucose Monitoring Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_CGMS_DESCR_VALUE_T type.

***CYBLE_EVT_CPSS_NOTIFICATION_ENABLED*** CPS Server - Notifications for Cycling Power Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type.

***CYBLE_EVT_CPSS_NOTIFICATION_DISABLED*** CPS Server - Notifications for Cycling Power Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_INDICATION_ENABLED*** CPS Server - Indication for Cycling Power Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_INDICATION_DISABLED*** CPS Server - Indication for Cycling Power Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_INDICATION_CONFIRMED*** CPS Server - Cycling Power Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_BROADCAST_ENABLED*** CPS Server - Broadcast for Cycling Power Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_BROADCAST_DISABLED*** CPS Server - Broadcast for Cycling Power Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSS_CHAR_WRITE*** CPS Server - Write Request for Cycling Power Service Characteristic was received. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type.

***CYBLE_EVT_CPSC_NOTIFICATION*** CPS Client - Cycling Power Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSC_INDICATION*** CPS Client - Cycling Power Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSC_READ_CHAR_RESPONSE*** CPS Client - Read Response for Read Request of Cycling Power Service Characteristic value. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type

***CYBLE_EVT_CPSC_WRITE_CHAR_RESPONSE*** CPS Client - Write Response for Write Request of Cycling Power Service Characteristic value. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type.

***CYBLE_EVT_CPSC_READ_DESCR_RESPONSE*** CPS Client - Read Response for Read Request of Cycling Power Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_CPS_DESCR_VALUE_T type.

***CYBLE_EVT_CPSC_WRITE_DESCR_RESPONSE*** CPS Client - Write Response for Write Request of Cycling Power Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_CPS_DESCR_VALUE_T type.

***CYBLE_EVT_CPSC_SCAN_PROGRESS_RESULT*** CPS Client - This event is triggered every time a device receive non-connectable undirected advertising event. The parameter of this event is a structure of CYBLE_CPS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_NOTIFICATION_ENABLED*** CSCS Server - Notifications for Cycling Speed and Cadence Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_NOTIFICATION_DISABLED*** CSCS Server - Notifications for Cycling Speed and Cadence Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_INDICATION_ENABLED*** CSCS Server - Indication for Cycling Speed and Cadence Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_INDICATION_DISABLED*** CSCS Server - Indication for Cycling Speed and Cadence Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_INDICATION_CONFIRMATION*** CSCS Server - Cycling Speed and Cadence Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSS_CHAR_WRITE*** CSCS Server - Write Request for Cycling Speed and Cadence Service Characteristic was received. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSC_NOTIFICATION*** CSCS Client - Cycling Speed and Cadence Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSC_INDICATION*** CSCS Client - Cycling Speed and Cadence Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSC_READ_CHAR_RESPONSE*** CSCS Client - Read Response for Read Request of Cycling Speed and Cadence Service Characteristic value. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSC_WRITE_CHAR_RESPONSE*** CSCS Client - Write Response for Write Request of Cycling Speed and Cadence Service Characteristic value. The parameter of this event is a structure of CYBLE_CSCS_CHAR_VALUE_T type.

***CYBLE_EVT_CSCSC_READ_DESCR_RESPONSE*** CSCS Client - Read Response for Read Request of Cycling Speed and Cadence Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_CSCS_DESCR_VALUE_T type.

***CYBLE_EVT_CSCSC_WRITE_DESCR_RESPONSE*** CSCS Client - Write Response for Write Request of Cycling Speed and Cadence Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_CSCS_DESCR_VALUE_T type.

***CYBLE_EVT_CTSS_NOTIFICATION_ENABLED*** CTS Server - Notification for Current Time Characteristic was enabled. The parameter of this event is a structure of CYBLE_CTS_CHAR_VALUE_T type.

***CYBLE_EVT_CTSS_NOTIFICATION_DISABLED*** CTS Server - Notification for Current Time Characteristic was disabled. The parameter of this event is a structure of CYBLE_CTS_CHAR_VALUE_T type.

***CYBLE_EVT_CTSS_CHAR_WRITE*** CTS Server - Write Request for Current Time Service Characteristic was received. The parameter of this event is a structure of CYBLE_CTS_CHAR_VALUE_T type. When this event is received the user is responsible for performing any kind of data verification and writing the data to the GATT database in case of successful verification or setting the error using CyBle_SetGattError() in case of data verification failure.

***CYBLE_EVT_CTSC_NOTIFICATION*** CTS Client - Current Time Characteristic Notification was received. The parameter of this event is a structure of CYBLE_CTS_CHAR_VALUE_T type.

***CYBLE_EVT_CTSC_READ_CHAR_RESPONSE*** CTS Client - Read Response for Current Time Characteristic Value Read Request. The parameter of this event is a structure of CYBLE_CTS_CHAR_VALUE_T type.

***CYBLE_EVT_CTSC_READ_DESCR_RESPONSE*** CTS Client - Read Response for Current Time Client Characteristic Configuration Descriptor Value Read Request. The parameter of this event is a structure of CYBLE_CTS_DESCR_VALUE_T type.

***CYBLE_EVT_CTSC_WRITE_DESCR_RESPONSE*** CTS Client - Write Response for Current Time Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_CTS_DESCR_VALUE_T type.

***CYBLE_EVT_CTSC_WRITE_CHAR_RESPONSE*** CTS Client - Write Response for Current Time or Local Time Information Characteristic Value. The parameter of this event is a structure of CYBLE_CTS_DESCR_VALUE_T type.

***CYBLE_EVT_DISC_READ_CHAR_RESPONSE*** DIS Client - Read Response for a Read Request for a Device Information Service Characteristic. The parameter of this event is a structure of CYBLE_DIS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_NOTIFICATION_ENABLED*** ESS Server - Notifications for Environmental Sensing Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_NOTIFICATION_DISABLED*** ESS Server - Notifications for Environmental Sensing Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_INDICATION_ENABLED*** ESS Server - Indication for Environmental Sensing Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_INDICATION_DISABLED*** ESS Server - Indication for Environmental Sensing Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_INDICATION_CONFIRMATION*** ESS Server - Environmental Sensing Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_CHAR_WRITE*** ESS Server - Write Request for Environmental Sensing Service Characteristic was received. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSS_DESCR_WRITE*** ESS Server - Write Request for Environmental Sensing Service Characteristic Descriptor was received. The parameter of this event is a structure of CYBLE_ESS_DESCR_VALUE_T type. This event is generated only when write for CYBLE_ESS_CHAR_USER_DESCRIPTION_DESCR, CYBLE_ESS_ES_TRIGGER_SETTINGS_DESCR or CYBLE_ESS_ES_CONFIG_DESCR occurred.

***CYBLE_EVT_ESSC_NOTIFICATION*** ESS Client - Environmental Sensing Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSC_INDICATION*** ESS Client - Environmental Sensing Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSC_READ_CHAR_RESPONSE*** ESS Client - Read Response for Read Request of Environmental Sensing Service Characteristic value. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSC_WRITE_CHAR_RESPONSE*** ESS Client - Write Response for Write Request of Environmental Sensing Service Characteristic value. The parameter of this event is a structure of CYBLE_ESS_CHAR_VALUE_T type.

***CYBLE_EVT_ESSC_READ_DESCR_RESPONSE*** ESS Client - Read Response for Read Request of Environmental Sensing Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_ESS_DESCR_VALUE_T type.

***CYBLE_EVT_ESSC_WRITE_DESCR_RESPONSE*** ESS Client - Write Response for Write Request of Environmental Sensing Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_ESS_DESCR_VALUE_T type.

***CYBLE_EVT_GLSS_INDICATION_ENABLED*** GLS Server - Indication for Glucose Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSS_INDICATION_DISABLED*** GLS Server - Indication for Glucose Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSS_INDICATION_CONFIRMED*** GLS Server - Glucose Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSS_NOTIFICATION_ENABLED*** GLS Server - Notifications for Glucose Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSS_NOTIFICATION_DISABLED*** GLS Server - Notifications for Glucose Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSS_WRITE_CHAR*** GLS Server - Write Request for Glucose Service was received. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSC_INDICATION*** GLS Client - Glucose Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSC_NOTIFICATION*** GLS Client - Glucose Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSC_READ_CHAR_RESPONSE*** GLS Client - Read Response for Read Request of Glucose Service Characteristic value. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSC_WRITE_CHAR_RESPONSE*** GLS Client - Write Response for Write Request of Glucose Service Characteristic value. The parameter of this event is a structure of CYBLE_GLS_CHAR_VALUE_T type.

***CYBLE_EVT_GLSC_READ_DESCR_RESPONSE*** GLS Client - Read Response for Read Request of Glucose Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_GLS_DESCR_VALUE_T type.

***CYBLE_EVT_GLSC_WRITE_DESCR_RESPONSE*** GLS Client - Write Response for Write Request of Glucose Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_GLS_DESCR_VALUE_T type.

***CYBLE_EVT_HIDSS_NOTIFICATION_ENABLED*** HIDS Server - Notifications for HID service were enabled. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_NOTIFICATION_DISABLED*** HIDS Server - Notifications for HID service were disabled. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_BOOT_MODE_ENTER*** HIDS Server - Enter boot mode request. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_REPORT_MODE_ENTER*** HIDS Server - Enter report mode request. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_SUSPEND*** HIDS Server - Enter suspend mode request. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_EXIT_SUSPEND*** HIDS Server - Exit suspend mode request. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSS_REPORT_CHAR_WRITE*** HIDS Server - Write Report characteristic request. The parameter of this event is a structure of CYBLE_HIDSS_REPORT_VALUE_T type.

***CYBLE_EVT_HIDSC_NOTIFICATION*** HIDS Client - HID Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSC_READ_CHAR_RESPONSE*** HIDS Client - Read Response for Read Request of HID Service Characteristic value. The parameter of this event is a structure of CYBLE_HIDS_DESCR_VALUE_T type.

***CYBLE_EVT_HIDSC_WRITE_CHAR_RESPONSE*** HIDS Client - Write Response for Write Request of HID Service Characteristic value. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HIDSC_READ_DESCR_RESPONSE*** HIDS Client - Read Response for Read Request of HID Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_HIDS_DESCR_VALUE_T type.

***CYBLE_EVT_HIDSC_WRITE_DESCR_RESPONSE*** HIDS Client - Write Response for Write Request of HID Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_HIDS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSS_NOTIFICATION_ENABLED*** HPS Server - Notification for HTTP Proxy Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSS_NOTIFICATION_DISABLED*** HPS Server - Notification for HTTP Proxy Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSS_CHAR_WRITE*** HPS Server - Write Request for HTTP Proxy Service Characteristic was received. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSC_NOTIFICATION*** HPS Client - HTTP Proxy Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSC_READ_CHAR_RESPONSE*** HPS Client - Read Response for Read Request of HTTP Proxy Service Characteristic value. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HPSC_READ_DESCR_RESPONSE*** HPS Client - Read Response for Read Request of HTTP Proxy Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_HPS_DESCR_VALUE_T type.

***CYBLE_EVT_HPSC_WRITE_DESCR_RESPONSE*** HPS Client - Write Response for Write Request of HTTP Proxy Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_HPS_DESCR_VALUE_T type.

***CYBLE_EVT_HPSC_WRITE_CHAR_RESPONSE*** HPS Client - Write Response for Write Request of HPS Service Characteristic value. The parameter of this event is a structure of CYBLE_HPS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSS_ENERGY_EXPENDED_RESET*** HRS Server - Reset Energy Expended. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSS_NOTIFICATION_ENABLED*** HRS Server - Notification for Heart Rate Measurement Characteristic was enabled. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSS_NOTIFICATION_DISABLED*** HRS Server - Notification for Heart Rate Measurement Characteristic was disabled. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSC_NOTIFICATION*** HRS Client - Heart Rate Measurement Characteristic Notification was received. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSC_READ_CHAR_RESPONSE*** HRS Client - Read Response for Read Request of HRS Service Characteristic value. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSC_WRITE_CHAR_RESPONSE*** HRS Client - Write Response for Write Request of HRS Service Characteristic value. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSC_READ_DESCR_RESPONSE*** HRS Client - Read Response for Read Request of HRS Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HRSC_WRITE_DESCR_RESPONSE*** HRS Client - Write Response for Write Request of HRS Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_HRS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_NOTIFICATION_ENABLED*** HTS Server - Notifications for Health Thermometer Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_NOTIFICATION_DISABLED*** HTS Server - Notifications for Health Thermometer Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_INDICATION_ENABLED*** HTS Server - Indication for Health Thermometer Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_INDICATION_DISABLED*** HTS Server - Indication for Health Thermometer Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_INDICATION_CONFIRMED*** HTS Server - Health Thermometer Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSS_CHAR_WRITE*** HTS Server - Write Request for Health Thermometer Service Characteristic was received. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSC_NOTIFICATION*** HTS Client - Health Thermometer Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSC_INDICATION*** HTS Client - Health Thermometer Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSC_READ_CHAR_RESPONSE*** HTS Client - Read Response for Read Request of Health Thermometer Service Characteristic value. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSC_WRITE_CHAR_RESPONSE*** HTS Client - Write Response for Write Request of Health Thermometer Service Characteristic value. The parameter of this event is a structure of CYBLE_HTS_CHAR_VALUE_T type.

***CYBLE_EVT_HTSC_READ_DESCR_RESPONSE*** HTS Client - Read Response for Read Request of Health Thermometer Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_HTS_DESCR_VALUE_T type.

***CYBLE_EVT_HTSC_WRITE_DESCR_RESPONSE*** HTS Client - Write Response for Write Request of Health Thermometer Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_HTS_DESCR_VALUE_T type.

***CYBLE_EVT_IASS_WRITE_CHAR_CMD*** IAS Server - Write command request for Alert Level Characteristic. The parameter of this event is a structure of CYBLE_IAS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSS_WRITE_CHAR*** IPS Server - Write Request for Indoor Positioning Service Characteristic was received. The parameter of this event is a structure of CYBLE_IPSS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSC_READ_CHAR_RESPONSE*** IPS Client - Read Response for Read Request of Indoor Positioning Service Characteristic value. The parameter of this event is a structure of CYBLE_IPS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSC_READ_MULTIPLE_CHAR_RESPONSE*** IPS Client - Read Multiple Response for Read Multiple Request of Indoor Positioning Service Characteristic value. The parameter of this event is a structure of CYBLE_IPS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSC_WRITE_CHAR_RESPONSE*** IPS Client - Write Response for Write Request of Indoor Positioning Service Characteristic value. The parameter of this event is a structure of CYBLE_IPS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSC_READ_DESCR_RESPONSE*** IPS Client - Read Response for Read Request of Indoor Positioning Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_IPS_DESCR_VALUE_T type.

***CYBLE_EVT_IPSC_WRITE_DESCR_RESPONSE*** IPS Client - Write Response for Write Request of Indoor Positioning Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_IPS_DESCR_VALUE_T type.

***CYBLE_EVT_IPSC_ERROR_RESPONSE*** IPS Client - Error Response for Write Request for Indoor Positioning Service Characteristic Value. The parameter of this event is a structure of CYBLE_IPS_CHAR_VALUE_T type.

***CYBLE_EVT_IPSC_READ_BLOB_RSP*** IPS Client - Read Response for Long Read Request of Indoor Positioning Service Characteristic value. The parameter of this event is a structure of CYBLE_IPS_CHAR_VALUE_T type.

***CYBLE_EVT_LLSS_WRITE_CHAR_REQ*** LLS Server - Write request for Alert Level Characteristic. The parameter of this event is a structure of CYBLE_LLS_CHAR_VALUE_T type.

***CYBLE_EVT_LLSC_READ_CHAR_RESPONSE*** LLS Client - Read response for Alert Level Characteristic. The parameter of this event is a structure of CYBLE_LLS_CHAR_VALUE_T type.

***CYBLE_EVT_LLSC_WRITE_CHAR_RESPONSE*** LLS Client - Write response for write request of Alert Level Characteristic. The parameter of this event is a structure of CYBLE_LLS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_INDICATION_ENABLED*** LNS Server - Indication for Location and Navigation Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_INDICATION_DISABLED*** LNS Server - Indication for Location and Navigation Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_INDICATION_CONFIRMED*** LNS Server - Location and Navigation Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_NOTIFICATION_ENABLED*** LNS Server - Notifications for Location and Navigation Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_NOTIFICATION_DISABLED*** LNS Server - Notifications for Location and Navigation Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSS_WRITE_CHAR*** LNS Server - Write Request for Location and Navigation Service Characteristic was received. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSC_INDICATION*** LNS Client - Location and Navigation Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSC_NOTIFICATION*** LNS Client - Location and Navigation Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSC_READ_CHAR_RESPONSE*** LNS Client - Read Response for Read Request of Location and Navigation Service Characteristic value. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSC_WRITE_CHAR_RESPONSE*** LNS Client - Write Response for Write Request of Location and Navigation Service Characteristic value. The parameter of this event is a structure of CYBLE_LNS_CHAR_VALUE_T type.

***CYBLE_EVT_LNSC_READ_DESCR_RESPONSE*** LNS Client - Read Response for Read Request of Location and Navigation Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_LNS_DESCR_VALUE_T type.

***CYBLE_EVT_LNSC_WRITE_DESCR_RESPONSE*** LNS Client - Write Response for Write Request of Location and Navigation Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_LNS_DESCR_VALUE_T type.

***CYBLE_EVT_NDCSC_READ_CHAR_RESPONSE*** NDCS Client - Read Response for Read Request of Next DST Change Service Characteristic value. The parameter of this event is a structure of CYBLE_NDCS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSS_NOTIFICATION_ENABLED*** PASS Server - Notifications for Phone Alert Status Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_PASS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSS_NOTIFICATION_DISABLED*** PASS Server - Notifications for Phone Alert Status Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_PASS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSS_WRITE_CHAR*** PASS Server - Write Request for Phone Alert Status Service Characteristic was received. The parameter of this event is a structure of CYBLE_PASS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSC_NOTIFICATION*** PASS Client - Phone Alert Status Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_PASS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSC_READ_CHAR_RESPONSE*** PASS Client - Read Response for Read Request of Phone Alert Status Service Characteristic value. The parameter of this event is a structure of CYBLE_PASS_CHAR_VALUE_T type.

***CYBLE_EVT_PASSC_READ_DESCR_RESPONSE*** PASS Client - Read Response for Read Request of Phone Alert Status Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_PASS_DESCR_VALUE_T type.

***CYBLE_EVT_PASSC_WRITE_DESCR_RESPONSE*** PASS Client - Write Response for Write Request of Phone Alert Status Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_PASS_DESCR_VALUE_T type.

***CYBLE_EVT_PLXSS_WRITE_CHAR*** PLXS Server - Write Request for Pulse Oximeter Service Characteristic was received. The parameter of this event is a structure of CYBLE_PLXSS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSS_NOTIFICATION_ENABLED*** PLXS Server - Notifications for Pulse Oximeter Characteristic were enabled. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSS_NOTIFICATION_DISABLED*** PLXS Server - Notifications for Pulse Oximeter Characteristic was disabled. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSS_INDICATION_ENABLED*** PLXS Server - Indication for Pulse Oximeter Characteristic was enabled. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSS_INDICATION_DISABLED*** PLXS Server - Indication for Pulse Oximeter Characteristic was disabled. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSS_INDICATION_CONFIRMED*** PLXS Server - Pulse Oximeter Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSC_NOTIFICATION*** PLXS Client - Pulse Oximeter Characteristic Notification was received. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSC_INDICATION*** PLXS Client - Pulse Oximeter Characteristic Indication was received. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSC_READ_CHAR_RESPONSE*** PLXS Client - Read Response for Read Request of Pulse Oximeter Service Characteristic value. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSC_WRITE_CHAR_RESPONSE*** PLXS Client - Write Response for Write Request of Pulse Oximeter Service Characteristic value. The parameter of this event is a structure of CYBLE_PLXS_CHAR_VALUE_T type.

***CYBLE_EVT_PLXSC_READ_DESCR_RESPONSE*** PLXS Client - Read Response for Read Request of Pulse Oximeter Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_PLXS_DESCR_VALUE_T type.

***CYBLE_EVT_PLXSC_WRITE_DESCR_RESPONSE*** PLXS Client - Write Response for Write Request of Pulse Oximeter Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_PLXS_DESCR_VALUE_T type.

***CYBLE_EVT_PLXSC_TIMEOUT*** PLXS Client - PLX RACP procedure timeout was received. The parameter of this event is a structure of the cy_stc_ble_plxs_char_value_t type.

***CYBLE_EVT_RSCSS_NOTIFICATION_ENABLED*** RSCS Server - Notifications for Running Speed and Cadence Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSS_NOTIFICATION_DISABLED*** RSCS Server - Notifications for Running Speed and Cadence Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSS_INDICATION_ENABLED*** RSCS Server - Indication for Running Speed and Cadence Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSS_INDICATION_DISABLED*** RSCS Server - Indication for Running Speed and Cadence Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSS_INDICATION_CONFIRMATION*** RSCS Server - Running Speed and Cadence Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSS_CHAR_WRITE*** RSCS Server - Write Request for Running Speed and Cadence Service Characteristic was received. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSC_NOTIFICATION*** RSCS Client - Running Speed and Cadence Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSC_INDICATION*** RSCS Client - Running Speed and Cadence Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSC_READ_CHAR_RESPONSE*** RSCS Client - Read Response for Read Request of Running Speed and Cadence Service Characteristic value. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSC_WRITE_CHAR_RESPONSE*** RSCS Client - Write Response for Write Request of Running Speed and Cadence Service Characteristic value. The parameter of this event is a structure of CYBLE_RSCS_CHAR_VALUE_T type.

***CYBLE_EVT_RSCSC_READ_DESCR_RESPONSE*** RSCS Client - Read Response for Read Request of Running Speed and Cadence Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_RSCS_DESCR_VALUE_T type.

***CYBLE_EVT_RSCSC_WRITE_DESCR_RESPONSE*** RSCS Client - Write Response for Write Request of Running Speed and Cadence Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_RSCS_DESCR_VALUE_T type.

***CYBLE_EVT_RTUSS_WRITE_CHAR_CMD*** RTUS Server - Write command request for Reference Time Update Characteristic value. The parameter of this event is a structure of CYBLE_RTUS_CHAR_VALUE_T type.

***CYBLE_EVT_RTUSC_READ_CHAR_RESPONSE*** RTUS Client - Read Response for Read Request of Reference Time Update Service Characteristic value. The parameter of this event is a structure of CYBLE_RTUS_CHAR_VALUE_T type.

***CYBLE_EVT_SCPSS_NOTIFICATION_ENABLED*** ScPS Server - Notifications for Scan Refresh Characteristic were enabled. The parameter of this event is a structure of CYBLE_SCPS_CHAR_VALUE_T type.

***CYBLE_EVT_SCPSS_NOTIFICATION_DISABLED*** ScPS Server - Notifications for Scan Refresh Characteristic were disabled. The parameter of this event is a structure of CYBLE_SCPS_CHAR_VALUE_T type.

***CYBLE_EVT_SCPSS_SCAN_INT_WIN_CHAR_WRITE*** ScPS Client - Read Response for Scan Interval Window Characteristic Value of Scan Parameters Service. The parameter of this event is a structure of CYBLE_SCPS_CHAR_VALUE_T type.

***CYBLE_EVT_SCPSC_NOTIFICATION*** ScPS Client - Scan Refresh Characteristic Notification was received. The parameter of this event is a structure of CYBLE_SCPS_CHAR_VALUE_T type.

***CYBLE_EVT_SCPSC_READ_DESCR_RESPONSE*** ScPS Client - Read Response for Scan Refresh Characteristic Descriptor Read Request. The parameter of this event is a structure of CYBLE_SCPS_DESCR_VALUE_T type.

***CYBLE_EVT_SCPSC_WRITE_DESCR_RESPONSE*** ScPS Client - Write Response for Scan Refresh Client Characteristic Configuration Descriptor Value. The parameter of this event is a structure of CYBLE_SCPS_DESCR_VALUE_T type.

***CYBLE_EVT_TPSS_NOTIFICATION_ENABLED*** TPS Server - Notification for Tx Power Level Characteristic was enabled. The parameter of this event is a structure of CYBLE_TPS_CHAR_VALUE_T type.

***CYBLE_EVT_TPSS_NOTIFICATION_DISABLED*** TPS Server - Notification for Tx Power Level Characteristic was disabled. The parameter of this event is a structure of CYBLE_TPS_CHAR_VALUE_T type.

***CYBLE_EVT_TPSC_NOTIFICATION*** TPS Client - Tx Power Level Characteristic Notification. The parameter of this event is a structure of CYBLE_TPS_CHAR_VALUE_T type.

***CYBLE_EVT_TPSC_READ_CHAR_RESPONSE*** TPS Client - Read Response for Tx Power Level Characteristic Value Read Request. The parameter of this event is a structure of CYBLE_TPS_CHAR_VALUE_T type.

***CYBLE_EVT_TPSC_READ_DESCR_RESPONSE*** TPS Client - Read Response for Tx Power Level Client Characteristic Configuration Descriptor Value Read Request. The parameter of this event is a structure of CYBLE_TPS_DESCR_VALUE_T type.

***CYBLE_EVT_TPSC_WRITE_DESCR_RESPONSE*** TPS Client - Write Response for Tx Power Level Characteristic Descriptor Value Write Request. The parameter of this event is a structure of CYBLE_TPS_DESCR_VALUE_T type.

***CYBLE_EVT_UDSS_INDICATION_ENABLED*** UDS Server - Indication for User Data Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_INDICATION_DISABLED*** UDS Server - Indication for User Data Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_INDICATION_CONFIRMED*** UDS Server - User Data Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_NOTIFICATION_ENABLED*** UDS Server - Notifications for User Data Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_NOTIFICATION_DISABLED*** UDS Server - Notifications for User Data Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_READ_CHAR*** UDS Server - Read Request for User Data Service Characteristic was received. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSS_WRITE_CHAR*** UDS Server - Write Request for User Data Service Characteristic was received. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSC_INDICATION*** UDS Client - User Data Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSC_NOTIFICATION*** UDS Client - User Data Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSC_READ_CHAR_RESPONSE*** UDS Client - Read Response for Read Request of User Data Service Characteristic value. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSC_WRITE_CHAR_RESPONSE*** UDS Client - Write Response for Write Request of User Data Service Characteristic value. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_UDSC_READ_DESCR_RESPONSE*** UDS Client - Read Response for Read Request of User Data Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_UDS_DESCR_VALUE_T type.

***CYBLE_EVT_UDSC_WRITE_DESCR_RESPONSE*** UDS Client - Write Response for Write Request of User Data Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_UDS_DESCR_VALUE_T type.

***CYBLE_EVT_UDSC_ERROR_RESPONSE*** UDS Client - Error Response for Write Request for User Data Service Characteristic Value. The parameter of this event is a structure of CYBLE_UDS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_NOTIFICATION_ENABLED*** WPTS Server - Notifications for Wireless Power Transfer Service Characteristic were enabled. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_NOTIFICATION_DISABLED*** WPTS Server - Notifications for Wireless Power Transfer Service Characteristic were disabled. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_INDICATION_ENABLED*** WPTS Server - Indication for Wireless Power Transfer Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_INDICATION_DISABLED*** WPTS Server - Indication for Wireless Power Transfer Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_INDICATION_CONFIRMED*** WPTS Server - Wireless Power Transfer Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSS_WRITE_CHAR*** WPTS Server - Write Request for Wireless Power Transfer Service Characteristic was received. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSC_NOTIFICATION*** WPTS Client - Wireless Power Transfer Service Characteristic Notification was received. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSC_INDICATION*** WPTS Client - Wireless Power Transfer Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSC_WRITE_CHAR_RESPONSE*** WPTS Client - Write Response for Read Request of Wireless Power Transfer Service Characteristic value. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSC_READ_CHAR_RESPONSE*** WPTS Client - Read Response for Read Request of Wireless Power Transfer Service Characteristic value. The parameter of this event is a structure of CYBLE_WPTS_CHAR_VALUE_T type.

***CYBLE_EVT_WPTSC_READ_DESCR_RESPONSE*** WPTS Client - Read Response for Read Request of Wireless Power Transfer Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_WPTS_DESCR_VALUE_T type.

***CYBLE_EVT_WPTSC_WRITE_DESCR_RESPONSE*** WPTS Client - Write Response for Write Request of Wireless Power Transfer Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_WPTS_DESCR_VALUE_T type.

***CYBLE_EVT_WSSS_INDICATION_ENABLED*** WSS Server - Indication for Weight Scale Service Characteristic was enabled. The parameter of this event is a structure of CYBLE_WSS_CHAR_VALUE_T type.

***CYBLE_EVT_WSSS_INDICATION_DISABLED*** WSS Server - Indication for Weight Scale Service Characteristic was disabled. The parameter of this event is a structure of CYBLE_WSS_CHAR_VALUE_T type.

***CYBLE_EVT_WSSS_INDICATION_CONFIRMED*** WSS Server - Weight Scale Service Characteristic Indication was confirmed. The parameter of this event is a structure of CYBLE_WSS_CHAR_VALUE_T type.

***CYBLE_EVT_WSSC_INDICATION*** WSS Client - Weight Scale Service Characteristic Indication was received. The parameter of this event is a structure of CYBLE_WSS_CHAR_VALUE_T type.

***CYBLE_EVT_WSSC_READ_CHAR_RESPONSE*** WSS Client - Read Response for Read Request of Weight Scale Service Characteristic value. The parameter of this event is a structure of CYBLE_WSS_CHAR_VALUE_T type.

***CYBLE_EVT_WSSC_READ_DESCR_RESPONSE*** WSS Client - Read Response for Read Request of Weight Scale Service Characteristic Descriptor Read request. The parameter of this event is a structure of CYBLE_WSS_DESCR_VALUE_T type.

***CYBLE_EVT_WSSC_WRITE_DESCR_RESPONSE*** WSS Client - Write Response for Write Request of Weight Scale Service Characteristic Configuration Descriptor value. The parameter of this event is a structure of CYBLE_WSS_DESCR_VALUE_T type.

***CYBLE_DEBUG_EVT_BLESS_INT*** Event from BLESS interrupt, enebled when StackMode parameter is set to Debug in the expression view of the customizer's General tab.

# Apple Notification Center Service (ANCS)

## Description

The Apple Notification Center Service provides iOS notifications from Apple devices for accessories.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The ANCS API names begin with CyBle_Ancs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- ANCS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- ANCS Server Functions
  *APIs unique to ANCS designs configured as a GATT Server role.*
- ANCS Client Functions
  *APIs unique to ANCS designs configured as a GATT Client role.*
- ANCS Definitions and Data Structures
  *Contains the ANCS specific definitions and data structures used in the ANCS APIs.*

# ANCS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Ancs

## Functions

- void CyBle_AncsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_AncsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service-specific attribute operations. Service-specific write requests from a peer device will not be handled with an unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for ANCS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam), where:<br>• eventCode indicates The event that triggered this callback.<br>• eventParam contains The parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# ANCS Server Functions

## Description

APIs unique to ANCS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Ancss

## Functions

- CYBLE_API_RESULT_T CyBle_AncssSetCharacteristicValue (CYBLE_ANCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AncssGetCharacteristicValue (CYBLE_ANCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AncssGetCharacteristicDescriptor (CYBLE_ANCS_CHAR_INDEX_T charIndex, CYBLE_ANCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AncssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AncssSetCharacteristicValue (CYBLE_ANCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AncssGetCharacteristicValue (CYBLE_ANCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - A characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AncssGetCharacteristicDescriptor (CYBLE_ANCS_CHAR_INDEX_T *charIndex*, CYBLE_ANCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The Characteristic Descriptor value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - A characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AncssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic value, as identified by the charIndex. On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_ANCSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle that consists of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# ANCS Client Functions

## Description

APIs unique to ANCS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Ancsc

## Functions

- CYBLE_API_RESULT_T CyBle_AncscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_AncscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANCS_CHAR_INDEX_T charIndex, CYBLE_ANCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AncscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANCS_CHAR_INDEX_T charIndex, CYBLE_ANCS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AncscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_ANCSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANCS service-specific callback is registered (with CyBle_AncsRegisterAttrCallback):

- CYBLE_EVT_ANCSC_WRITE_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, etc.) are provided with an event parameter structure of type CYBLE_ANCS_CHAR_VALUE_T.

Otherwise (if the ANCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AncscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANCS_CHAR_INDEX_T *charIndex*, CYBLE_ANCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server, as identified by its charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_ANCSS_NOTIFICATION_ENABLED.
- CYBLE_EVT_ANCSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successfull execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANCS service-specific callback is registered (with CyBle_AncsRegisterAttrCallback):

- CYBLE_EVT_ANCSC_WRITE_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index etc.) are provided with an event parameter structure of type CYBLE_ANCS_DESCR_VALUE_T.

Otherwise (if the ANCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T.

**CYBLE_API_RESULT_T CyBle_AncscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANCS_CHAR_INDEX_T *charIndex*, CYBLE_ANCS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANCS service-specific callback is registered (with CyBle_AncsRegisterAttrCallback):

- CYBLE_EVT_ANCSC_READ_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index, value, etc.) are provided with an event parameter structure of type CYBLE_ANCS_DESCR_VALUE_T.

Otherwise (if the ANCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# ANCS Definitions and Data Structures

## Description

Contains the ANCS specific definitions and data structures used in the ANCS APIs.

## Data Structures

- struct CYBLE_ANCSS_CHAR_T
- struct CYBLE_ANCSS_T
- struct CYBLE_ANCSC_CHAR_T
- struct CYBLE_ANCSC_T
- struct CYBLE_ANCS_CHAR_VALUE_T
- struct CYBLE_ANCS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_ANCS_CHAR_INDEX_T
- enum CYBLE_ANCS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_ANCSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_ANCS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANCSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_ANCSS_CHAR_T::descrHandle[CYBLE_ANCS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_ANCSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_ANCSS_CHAR_T charInfo [CYBLE_ANCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANCSS_T::serviceHandle**
ANC Service handle

**CYBLE_ANCSS_CHAR_T CYBLE_ANCSS_T::charInfo[CYBLE_ANCS_CHAR_COUNT]**
ANC Service characteristics info array

**struct CYBLE_ANCSC_CHAR_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_ANCS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_ANCSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANCSC_CHAR_T::valueHandle**
Handle of server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_ANCSC_CHAR_T::descrHandle[CYBLE_ANCS_DESCR_COUNT]**
ANCS client char. descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANCSC_CHAR_T::endHandle**
Characteristic End Handle

**struct CYBLE_ANCSC_T**

**Data Fields**

- CYBLE_ANCSC_CHAR_T charInfo [CYBLE_ANCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_ANCSC_CHAR_T CYBLE_ANCSC_T::charInfo[CYBLE_ANCS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_ANCS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ANCS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ANCS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_ANCS_CHAR_INDEX_T CYBLE_ANCS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_ANCS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_ANCS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**struct CYBLE_ANCS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ANCS_CHAR_INDEX_T charIndex
- CYBLE_ANCS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ANCS_DESCR_VALUE_T::connHandle**

Peer device handle

**CYBLE_ANCS_CHAR_INDEX_T CYBLE_ANCS_DESCR_VALUE_T::charIndex**

Index of service characteristic

**CYBLE_ANCS_DESCR_INDEX_T CYBLE_ANCS_DESCR_VALUE_T::descrIndex**

Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_ANCS_DESCR_VALUE_T::value**

Descriptor value

### Enumeration Type Documentation

**enum CYBLE_ANCS_CHAR_INDEX_T**

ANC Service Characteristics indexes

**Enumerator**

***CYBLE_ANCS_NS*** Notification Source characteristic index

***CYBLE_ANCS_CP*** Control Point characteristic index

***CYBLE_ANCS_DS*** Data Source characteristic index

***CYBLE_ANCS_CHAR_COUNT*** Total count of ANCS characteristics

**enum CYBLE_ANCS_DESCR_INDEX_T**

ANC Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_ANCS_CCCD*** Client Characteristic Configuration descriptor index

***CYBLE_ANCS_DESCR_COUNT*** Total count of ANCS descriptors

## Alert Notification Service (ANS)

### Description

The Alert Notification Service exposes alert information in a device.

This information includes:

- Type of alert occurring in a device
- Additional text information such as the caller's ID or sender's ID
- Count of new alerts
- Count of unread alert items

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The ANS API names begin with CyBle_Ans. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- ANS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- ANS Server Functions
  *APIs unique to ANS designs configured as a GATT Server role.*
- ANS Client Functions
  *APIs unique to ANS designs configured as a GATT Client role.*
- ANS Definitions and Data Structures
  *Contains the ANS specific definitions and data structures used in the ANS APIs.*

## ANS Server and Client Function

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Ans

### Functions

- void CyBle_AnsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_AnsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for Alert Notification Service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive service specific events from the BLE Component. The definition of CYBLE_CALLBACK_T for Alert Notification Service is,<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_ANSS_NOTIFICATION_ENABLED)<br>• eventParam contains the parameters corresponding to the current event (e.g. Pointer to CYBLE_ANS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# ANS Server Functions

## Description

APIs unique to ANS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Anss

## Functions

- CYBLE_API_RESULT_T CyBle_AnssSetCharacteristicValue (CYBLE_ANS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AnssGetCharacteristicValue (CYBLE_ANS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AnssGetCharacteristicDescriptor (CYBLE_ANS_CHAR_INDEX_T charIndex, CYBLE_ANS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AnssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AnssSetCharacteristicValue (CYBLE_ANS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of Alert Notification Service, which is a value identified by charIndex, to the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic of type CYBLE_ANS_CHAR_INDEX_T. The valid values are,<br>• CYBLE_ANS_SUPPORTED_NEW_ALERT_CAT<br>• CYBLE_ANS_SUPPORTED_UNREAD_ALERT_CAT |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_AnssGetCharacteristicValue (CYBLE_ANS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of Alert Notification Service. The value is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic of type CYBLE_ANS_CHAR_INDEX_T. The valid values are,<br>• CYBLE_ANS_NEW_ALERT<br>• CYBLE_ANS_UNREAD_ALERT_STATUS |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_AnssGetCharacteristicDescriptor (CYBLE_ANS_CHAR_INDEX_T *charIndex*, CYBLE_ANS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of the specified characteristic of Alert Notification Service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic of type CYBLE_ANS_CHAR_INDEX_T. The valid values are,<br>• CYBLE_ANS_NEW_ALERT<br>• CYBLE_ANS_UNREAD_ALERT_STATUS |
| *descrIndex* | The index of the service characteristic descriptor of type CYBLE_ANS_DESCR_INDEX_T. The valid value is,<br>• CYBLE_ANS_CCCD |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_AnssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification with the characteristic value, as specified by its charIndex, to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_ANSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic of type CYBLE_ANS_CHAR_INDEX_T. The valid values are,<br>• CYBLE_ANS_UNREAD_ALERT_STATUS;<br>• CYBLE_ANS_NEW_ALERT. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The function completed successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter is failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.

- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# ANS Client Functions

## Description

APIs unique to ANS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Ansc

## Functions

- CYBLE_API_RESULT_T CyBle_AnscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_AnscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AnscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANS_CHAR_INDEX_T charIndex, CYBLE_ANS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AnscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ANS_CHAR_INDEX_T charIndex, uint8 descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AnscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANS_CHAR_INDEX_T *charIndex*)**

Sends a request to the peer device to get a characteristic value, as identified by its charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully;
- CYBLE_ERROR_INVALID_STATE - The component in in invalid state for current operation.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANS service-specific callback is registered (with CyBle_AnsRegisterAttrCallback):

- CYBLE_EVT_ANSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_ANS_CHAR_VALUE_T.

Otherwise (if the ANS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).

- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AnscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_ANSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | Size of the Characteristic value attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - The component in in invalid state for current operation.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANS service-specific callback is registered (with CyBle_AnsRegisterAttrCallback):

- CYBLE_EVT_ANSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_ANS_CHAR_VALUE_T.

Otherwise (if the ANS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AnscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANS_CHAR_INDEX_T *charIndex*, CYBLE_ANS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a request to the peer device to set the characteristic descriptor of the specified characteristic of Alert Notification Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_ANSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_ANSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *charIndex* | The index of the ANS characteristic. |
| *descrIndex* | The index of the ANS characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |

| | |
|---|---|
| *attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - The component in in invalid state for current operation.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANS service-specific callback is registered (with CyBle_AnsRegisterAttrCallback):

- CYBLE_EVT_ANSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_ANS_DESCR_VALUE_T.

Otherwise (if the ANS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_AnscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ANS_CHAR_INDEX_T *charIndex*, uint8 *descrIndex*)

Sends a request to the peer device to get the characteristic descriptor of the specified characteristic of Alert Notification Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | BLE peer device connection handle. |
| *charIndex* | The index of the Service Characteristic. |
| *descrIndex* | The index of the Service Characteristic Descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - A request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The component is in invalid state for current operation.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Cannot process a request to send PDU due to invalid operation performed by the application.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ANS service-specific callback is registered (with CyBle_AnsRegisterAttrCallback):

- CYBLE_EVT_ANSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_ANS_DESCR_VALUE_T.

Otherwise (if the ANS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).

- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# ANS Definitions and Data Structures

## Description

Contains the ANS specific definitions and data structures used in the ANS APIs.

## Data Structures

- struct CYBLE_ANS_CHAR_VALUE_T
- struct CYBLE_ANS_DESCR_VALUE_T
- struct CYBLE_ANSS_CHAR_T
- struct CYBLE_ANSS_T
- struct CYBLE_SRVR_FULL_CHAR_INFO_T
- struct CYBLE_ANSC_T

## Enumerations

- enum CYBLE_ANS_CHAR_INDEX_T
- enum CYBLE_ANS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_ANS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ANS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ANS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_ANS_CHAR_INDEX_T CYBLE_ANS_CHAR_VALUE_T::charIndex**

Index of Alert Notification Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_ANS_CHAR_VALUE_T::value**

Pointer to Characteristic value

### struct CYBLE_ANS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ANS_CHAR_INDEX_T charIndex
- CYBLE_ANS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ANS_DESCR_VALUE_T::connHandle**
Connection handle

**CYBLE_ANS_CHAR_INDEX_T CYBLE_ANS_DESCR_VALUE_T::charIndex**
Characteristic index of Service

**CYBLE_ANS_DESCR_INDEX_T CYBLE_ANS_DESCR_VALUE_T::descrIndex**
Service Characteristic Descriptor index

**CYBLE_GATT_VALUE_T* CYBLE_ANS_DESCR_VALUE_T::value**
Pointer to value of Service Characteristic Descriptor value

**struct CYBLE_ANSS_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_ANS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANSS_CHAR_T::charHandle**
Handle of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANSS_CHAR_T::descrHandle[CYBLE_ANS_DESCR_COUNT]**
Handle of Descriptor

**struct CYBLE_ANSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_ANSS_CHAR_T charInfo [CYBLE_ANS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ANSS_T::serviceHandle**
Alert Notification Service handle

**CYBLE_ANSS_CHAR_T CYBLE_ANSS_T::charInfo[CYBLE_ANS_CHAR_COUNT]**
Array of Alert Notification Service Characteristics + Descriptors handles

**struct CYBLE_SRVR_FULL_CHAR_INFO_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descriptors [CYBLE_ANS_DESCR_COUNT]

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_SRVR_FULL_CHAR_INFO_T::charInfo**
Characteristic handle + properties

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SRVR_FULL_CHAR_INFO_T::endHandle**
End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_SRVR_FULL_CHAR_INFO_T::descriptors[CYBLE_ANS_DESCR_COUNT]**
Characteristic descriptors handles

**struct CYBLE_ANSC_T**

**Data Fields**

- CYBLE_SRVR_FULL_CHAR_INFO_T characteristics [CYBLE_ANS_CHAR_COUNT]

**Field Documentation**

**CYBLE_SRVR_FULL_CHAR_INFO_T CYBLE_ANSC_T::characteristics[CYBLE_ANS_CHAR_COUNT]**

Structure with Characteristic handles + properties of Alert Notification Service

### Enumeration Type Documentation

**enum CYBLE_ANS_CHAR_INDEX_T**

ANS Characteristic indexes

**Enumerator**

***CYBLE_ANS_SUPPORTED_NEW_ALERT_CAT*** Supported New Alert Category Characteristic index

***CYBLE_ANS_NEW_ALERT*** New Alert Characteristic index

***CYBLE_ANS_SUPPORTED_UNREAD_ALERT_CAT*** Supported Unread Alert Category Characteristic index

***CYBLE_ANS_UNREAD_ALERT_STATUS*** Unread Alert Status Characteristic index

***CYBLE_ANS_ALERT_NTF_CONTROL_POINT*** Alert Notification Control Point Characteristic index

***CYBLE_ANS_CHAR_COUNT*** Total count of ANS characteristics

**enum CYBLE_ANS_DESCR_INDEX_T**

ANS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_ANS_CCCD*** Client Characteristic Configuration Descriptor index

***CYBLE_ANS_DESCR_COUNT*** Total count of descriptors

## Automation IO Service (AIOS)

### Description

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

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The AIOS API names begin with CyBle_Aios. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- AIOS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- AIOS Server Functions
  *APIs unique to AIOS designs configured as a GATT Server role.*
- AIOS Client Functions
  *APIs unique to AIOS designs configured as a GATT Client role.*
- AIOS Definitions and Data Structures
  *Contains the AIOS specific definitions and data structures used in the AIOS APIs.*

# AIOS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Aios

## Functions

- void CyBle_AiosRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_AiosRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for AIOS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode: Indicates the event that triggered this callback (e.g. CYBLE_EVT_AIOS_NOTIFICATION_ENABLED).<br>• eventParam: Contains the parameters corresponding to the current event. (e.g. Pointer to CYBLE_AIOS_CHAR_VALUE_T structure that contains details of the characteristic for which the notification enabled event was triggered). |

# AIOS Server Functions

## Description

APIs unique to AIOS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Aioss

## Functions

- CYBLE_API_RESULT_T CyBle_AiossSetCharacteristicValue (CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AiossGetCharacteristicValue (CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AiossSetCharacteristicDescriptor (CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_AIOS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AiossGetCharacteristicDescriptor (CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_AIOS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AiossSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_AiossSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AiossSetCharacteristicValue (CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size (in bytes) of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AiossGetCharacteristicValue (CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic value of the service, which is a value identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AiossSetCharacteristicDescriptor (CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_AIOS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Set a characteristic descriptor of a specified characteristic of the Indoor Positioning Service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_AiossGetCharacteristicDescriptor (CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_AIOS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Automation Input Outtput Service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_AiossSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification with a characteristic value of the Automation Input Output Service, which is a value specified by charIndex, to the client's device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_AIOSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - A notification is not enabled by the client.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_AiossSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value of the Automation Input Output Service, which is a value specified by charIndex, to the client's device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_AIOSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the Aios service-specific callback is registered (with CyBle_AiosRegisterAttrCallback):

- CYBLE_EVT_AiosS_INDICATION_CONFIRMED -In case if the indication is successfully delivered to the peer device.

Otherwise (if the Aios service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - In case if the indication is successfully delivered to the peer device.

# AIOS Client Functions

## Description

APIs unique to AIOS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Aiosc

## Functions

- CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicValueWithoutResponse (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AioscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance)
- CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_AIOS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_AioscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_AIOS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_AIOS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicValueWithoutResponse (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server without response.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_AIOSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the AIOS service-specific callback is registered (with CyBle_AiosRegisterAttrCallback):

- CYBLE_EVT_AIOSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_AIOS_CHAR_VALUE_T.

Otherwise (if the AIOS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AioscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the AIOS service-specific callback is registered (with CyBle_AiosRegisterAttrCallback):

- CYBLE_EVT_AIOSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_AIOS_CHAR_VALUE_T.

Otherwise (if the AIOS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AioscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_AIOS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_AIOSS_DESCR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_AIOSS_NOTIFICATION_ENABLED ;

- CYBLE_EVT_AIOSS_NOTIFICATION_DISABLED;
- CYBLE_EVT_AIOSS_INDICATION_ENABLED;
- CYBLE_EVT_AIOSS_INDICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional Characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the AIOS service-specific callback is registered (with CyBle_AiosRegisterAttrCallback):

- CYBLE_EVT_AIOSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_AIOS_DESCR_VALUE_T.

Otherwise (if the AIOS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_AioscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_AIOS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_AIOS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional Characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the AIOS service-specific callback is registered (with CyBle_AiosRegisterAttrCallback):

- CYBLE_EVT_AIOSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_AIOS_DESCR_VALUE_T.

Otherwise (if the AIOS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# AIOS Definitions and Data Structures

## Description

Contains the AIOS specific definitions and data structures used in the AIOS APIs.

## Data Structures

- struct CYBLE_AIOSS_CHAR_T
- struct CYBLE_AIOSS_CHAR_INFO_PTR_T
- struct CYBLE_AIOSS_T
- struct CYBLE_AIOSC_CHAR_T
- struct CYBLE_AIOSC_CHAR_INFO_PTR_T
- struct CYBLE_AIOSC_T
- struct CYBLE_AIOS_CHAR_VALUE_T
- struct CYBLE_AIOS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_AIOS_CHAR_INDEX_T
- enum CYBLE_AIOS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_AIOSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_AIOS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_AIOSS_CHAR_T::charHandle**

Handles of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_AIOSS_CHAR_T::descrHandle[CYBLE_AIOS_DESCR_COUNT]**
Array of Descriptor handles

**struct CYBLE_AIOSS_CHAR_INFO_PTR_T**

**Data Fields**

- CYBLE_AIOSS_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_AIOSS_CHAR_T* CYBLE_AIOSS_CHAR_INFO_PTR_T::charInfoPtr**
Pointer to CYBLE_AIOSS_CHAR_T which holds information about specific AIO Characteristic

**struct CYBLE_AIOSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_AIOSS_CHAR_INFO_PTR_T charInfoAddr [CYBLE_AIOS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_AIOSS_T::serviceHandle**
Automation Input Output Service handle

**CYBLE_AIOSS_CHAR_INFO_PTR_T CYBLE_AIOSS_T::charInfoAddr[CYBLE_AIOS_CHAR_COUNT]**
Automation Input Output Service Array with pointers to Characteristic handles.

**struct CYBLE_AIOSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_AIOS_DESCR_COUNT]
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_AIOSC_CHAR_T::valueHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_AIOSC_CHAR_T::endHandle**
End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_AIOSC_CHAR_T::descrHandle[CYBLE_AIOS_DESCR_COUNT]**
Array of Descriptor handles

**uint8 CYBLE_AIOSC_CHAR_T::properties**
Properties for value field

**struct CYBLE_AIOSC_CHAR_INFO_PTR_T**

**Data Fields**

- CYBLE_AIOSC_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_AIOSC_CHAR_T* CYBLE_AIOSC_CHAR_INFO_PTR_T::charInfoPtr**
Pointer to CYBLE_AIOSC_CHAR_T which holds information about specific AIO Characteristic.

**struct CYBLE_AIOSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_AIOSC_CHAR_INFO_PTR_T charInfoAddr [CYBLE_AIOS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_AIOSC_T::serviceHandle**
Automation Input Output Service handle

**CYBLE_AIOSC_CHAR_INFO_PTR_T CYBLE_AIOSC_T::charInfoAddr[CYBLE_AIOS_CHAR_COUNT]**
Automation Input Output Service Array with pointers to characteristic information.

**struct CYBLE_AIOS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_AIOS_CHAR_INDEX_T charIndex
- uint8 charInstance
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_AIOS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_AIOS_CHAR_INDEX_T CYBLE_AIOS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**uint8 CYBLE_AIOS_CHAR_VALUE_T::charInstance**
Instance of specific service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_AIOS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_AIOS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**struct CYBLE_AIOS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_AIOS_CHAR_INDEX_T charIndex
- uint8 charInstance
- CYBLE_AIOS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_AIOS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_AIOS_CHAR_INDEX_T CYBLE_AIOS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**uint8 CYBLE_AIOS_DESCR_VALUE_T::charInstance**
Instance of specific service characteristic

**CYBLE_AIOS_DESCR_INDEX_T CYBLE_AIOS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_ERR_CODE_T CYBLE_AIOS_DESCR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_AIOS_DESCR_VALUE_T::value**
Characteristic value

### Enumeration Type Documentation

**enum CYBLE_AIOS_CHAR_INDEX_T**
AIOS Characteristic indexes
**Enumerator**
***CYBLE_AIOS_DIGITAL*** AIOS Digital characteristic
***CYBLE_AIOS_ANALOG*** AIOS Analog characteristic
***CYBLE_AIOS_AGGREGATE*** AIOS Aggregate characteristic
***CYBLE_AIOS_CHAR_COUNT*** Total count of AIOS characteristics

**enum CYBLE_AIOS_DESCR_INDEX_T**
AIOS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_AIOS_CCCD*** Client Characteristic Configuration Descriptor index
***CYBLE_AIOS_CHAR_PRESENTATION_FORMAT*** Characteristic Presentation Format Descriptor index
***CYBLE_AIOS_CHAR_USER_DESCRIPTION_DESCR*** Characteristic User Description Descriptor index
***CYBLE_AIOS_CHAR_EXTENDED_PROPERTIES*** Characteristic Extended Properties Descriptor index
***CYBLE_AIOS_VALUE_TRIGGER_SETTINGS*** AIO Value Trigger Settings Descriptor index
***CYBLE_AIOS_TIME_TRIGGER_SETTINGS*** AIO Time Trigger Settings Descriptor index
***CYBLE_AIOS_VRD*** Valid Range Descriptor index
***CYBLE_AIOS_NUM_OF_DIGITAL_DESCR*** Number of Digitals Descriptor index
***CYBLE_AIOS_DESCR_COUNT*** Total count of descriptors

## Battery Service (BAS)

### Description

The Battery Service exposes the battery level of a single battery or set of batteries in a device.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The BAS API names begin with CyBle_Bas. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- BAS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- BAS Server Functions
  *APIs unique to BAS designs configured as a GATT Server role.*

- BAS Client Functions
  *APIs unique to BAS designs configured as a GATT Client role.*
- BAS Definitions and Data Structures
  *Contains the BAS specific definitions and data structures used in the BAS APIs.*

# BAS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Bas

## Functions

- void CyBle_BasRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_BasRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive battery service events from the BLE Component. The definition of CYBLE_CALLBACK_T for Battery Service is,<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_BASS_NOTIFICATION_ENABLED)<br>• eventParam contains the parameters corresponding to the current event (e.g., pointer to CYBLE_BAS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered) |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# BAS Server Functions

## Description

APIs unique to BAS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Bass

## Functions

- CYBLE_API_RESULT_T CyBle_BassSetCharacteristicValue (uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_BassGetCharacteristicValue (uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BassGetCharacteristicDescriptor (uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, CYBLE_BAS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BassSendNotification (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BassSetCharacteristicValue (uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. |
| *charIndex* | The index of the service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. A battery level characteristic has 1 byte length. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_BassGetCharacteristicValue (uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Battery service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. A battery level characteristic has a 1 byte length. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_BassGetCharacteristicDescriptor (uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, CYBLE_BAS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Battery service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_BAS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_BassSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function updates the value of the Battery Level characteristic in the GATT database. If the client has configured a notification on the Battery Level characteristic, the function additionally sends this value using a GATT Notification message.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_BASC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle |
| *serviceIndex* | The index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. A battery level characteristic has 1 byte length. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# BAS Client Functions

## Description

APIs unique to BAS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Basc

## Functions

- CYBLE_API_RESULT_T CyBle_BascGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_BascSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, CYBLE_BAS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BascGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_BAS_CHAR_INDEX_T charIndex, CYBLE_BAS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BascGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic value from a server which is identified by charIndex.

This function call can result in generation of the following events based on the response from the server device.

- CYBLE_EVT_BASC_READ_CHAR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *serviceIndex* | Index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BAS service-specific callback is registered (with CyBle_BasRegisterAttrCallback):

- CYBLE_EVT_BASC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_BAS_CHAR_VALUE_T.

Otherwise (if the BAS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BascSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, CYBLE_BAS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a request to set characteristic descriptor of specified Battery Service characteristic on the server device.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_BASS_NOTIFICATION_ENABLED.
- CYBLE_EVT_BASS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *serviceIndex* | Index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_BAS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BAS service-specific callback is registered (with CyBle_BasRegisterAttrCallback):

- CYBLE_EVT_BASC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_BAS_DESCR_VALUE_T.

Otherwise (if the BAS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BascGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_BAS_CHAR_INDEX_T *charIndex*, CYBLE_BAS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get characteristic descriptor of specified Battery Service characteristic from the server device. This function call can result in generation of the following events based on the response from the server device.

- CYBLE_EVT_BASC_READ_DESCR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |

| | |
|---|---|
| *serviceIndex* | Index of the service instance. e.g. If two Battery Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a Battery service characteristic of type CYBLE_BAS_CHAR_INDEX_T. |
| *descrIndex* | The index of a Battery service characteristic descriptor of type CYBLE_BAS_DESCR_INDEX_T. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BAS service-specific callback is registered (with CyBle_BasRegisterAttrCallback):

- CYBLE_EVT_BASC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_BAS_DESCR_VALUE_T.

Otherwise (if the BAS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# BAS Definitions and Data Structures

## Description

Contains the BAS specific definitions and data structures used in the BAS APIs.

## Data Structures

- struct CYBLE_BASS_T
- struct CYBLE_BASC_T
- struct CYBLE_BAS_CHAR_VALUE_T
- struct CYBLE_BAS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_BAS_CHAR_INDEX_T
- enum CYBLE_BAS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_BASS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T batteryLevelHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cpfdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASS_T::serviceHandle**
Battery Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASS_T::batteryLevelHandle**
Battery Level characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASS_T::cpfdHandle**
Characteristic Presentation Format Descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASS_T::cccdHandle**
Client Characteristic Configuration descriptor handle

### struct CYBLE_BASC_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_SRVR_CHAR_INFO_T batteryLevel
- CYBLE_GATT_DB_ATTR_HANDLE_T cpfdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T rrdHandle

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BASC_T::connHandle**
Peer device handle

**CYBLE_SRVR_CHAR_INFO_T CYBLE_BASC_T::batteryLevel**
Battery Level characteristic info

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASC_T::cpfdHandle**
Characteristic Presentation Format descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASC_T::cccdHandle**
Client Characteristic Configuration descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BASC_T::rrdHandle**
Report Reference descriptor handle

### struct CYBLE_BAS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint8 serviceIndex
- CYBLE_BAS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BAS_CHAR_VALUE_T::connHandle**
Peer device handle

**uint8 CYBLE_BAS_CHAR_VALUE_T::serviceIndex**
Service instance

**CYBLE_BAS_CHAR_INDEX_T CYBLE_BAS_CHAR_VALUE_T::charIndex**
Index of a service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_BAS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_BAS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint8 serviceIndex
- CYBLE_BAS_CHAR_INDEX_T charIndex
- CYBLE_BAS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BAS_DESCR_VALUE_T::connHandle**
Peer device handle

**uint8 CYBLE_BAS_DESCR_VALUE_T::serviceIndex**
Service instance

**CYBLE_BAS_CHAR_INDEX_T CYBLE_BAS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_BAS_DESCR_INDEX_T CYBLE_BAS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_BAS_DESCR_VALUE_T::value**
Descriptor value

## Enumeration Type Documentation

**enum CYBLE_BAS_CHAR_INDEX_T**
BAS Characteristic indexes
**Enumerator**
***CYBLE_BAS_BATTERY_LEVEL*** Battery Level characteristic index
***CYBLE_BAS_CHAR_COUNT*** Total count of characteristics

**enum CYBLE_BAS_DESCR_INDEX_T**
BAS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_BAS_BATTERY_LEVEL_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_BAS_BATTERY_LEVEL_CPFD*** Characteristic Presentation Format descriptor index
***CYBLE_BAS_DESCR_COUNT*** Total count of descriptors

# Body Composition Service (BCS)

## Description

The Body Composition Service exposes data related to body composition from a body composition analyzer (Server) intended for consumer healthcare as well as sports/fitness applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The BCS API names begin with CyBle_Bcs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- BCS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- BCS Server Functions
  *APIs unique to BCS designs configured as a GATT Server role.*
- BCS Client Functions
  *APIs unique to BCS designs configured as a GATT Client role.*
- BCS Definitions and Data Structures
  *Contains the BCS specific definitions and data structures used in the BCS APIs.*

# BCS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Bcs

## Functions

- void CyBle_BcsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_BcsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode: Indicates the event that triggered this callback (e.g. CYBLE_EVT_BCSS_INDICATION_ENABLED).<br>• eventParam: Contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_BCS_CHAR_VALUE_T structure that contains details of the characteristic for which notification the enabled event was triggered). |

| | |
|---|---|

# BCS Server Functions

## Description

APIs unique to BCS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Bcss

## Functions

- CYBLE_API_RESULT_T CyBle_BcssSetCharacteristicValue (CYBLE_BCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BcssGetCharacteristicValue (CYBLE_BCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BcssSetCharacteristicDescriptor (CYBLE_BCS_CHAR_INDEX_T charIndex, CYBLE_BCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BcssGetCharacteristicDescriptor (CYBLE_BCS_CHAR_INDEX_T charIndex, CYBLE_BCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BcssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BcssSetCharacteristicValue (CYBLE_BCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a value for one of three characteristic values of the Body Composition Service. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a Body Composition Service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was written successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_BcssGetCharacteristicValue (CYBLE_BCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a characteristic value of the Body Composition Service, which is identified by charIndex from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the Body Composition Service characteristic. |
| *attrSize* | The size of the Body Composition Service characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_BcssSetCharacteristicDescriptor (CYBLE_BCS_CHAR_INDEX_T *charIndex*, CYBLE_BCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_BcssGetCharacteristicDescriptor (CYBLE_BCS_CHAR_INDEX_T *charIndex*, CYBLE_BCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a a characteristic descriptor of a specified characteristic of the Body Composition Service from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully. CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_BcssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value of the Body Composition Service, which is a value specified by charIndex, to the client's device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_BCSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BCS service-specific callback is registered (with CyBle_BcsRegisterAttrCallback):

- CYBLE_EVT_BCSS_INDICATION_CONFIRMED - If the indication is successfully delivered to the peer device.

Otherwise (if the BCS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - If the indication is successfully delivered to the peer device.

# BCS Client Functions

## Description

APIs unique to BCS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Bcsc

## Functions

- CYBLE_API_RESULT_T CyBle_BcscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BCS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_BcscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BCS_CHAR_INDEX_T charIndex, CYBLE_BCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BcscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BCS_CHAR_INDEX_T charIndex, CYBLE_BCS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BcscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BCS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BCS service-specific callback is registered (with CyBle_BcsRegisterAttrCallback):

- CYBLE_EVT_BCSC_READ_CHAR_RESPONSE - If the requested attribute is successfully read on the peer device, ! the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_BCS_CHAR_VALUE_T.

Otherwise (if the BCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BcscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BCS_CHAR_INDEX_T *charIndex*, CYBLE_BCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic descriptor to the server, which is identified by charIndex and descrIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_BCSS_INDICATION_ENABLED
- CYBLE_EVT_BCSS_INDICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BCS service-specific callback is registered (with CyBle_BcsRegisterAttrCallback):

- CYBLE_EVT_BCSC_WRITE_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index etc.) are provided with an event parameter structure of type CYBLE_BCS_DESCR_VALUE_T.

Otherwise (if the BCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BcscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BCS_CHAR_INDEX_T *charIndex*, CYBLE_BCS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BCS service-specific callback is registered (with CyBle_BcsRegisterAttrCallback):

- CYBLE_EVT_BCSC_READ_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index, value, etc.) are provided with an event parameter structure of type CYBLE_BCS_DESCR_VALUE_T.

Otherwise (if the BCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# BCS Definitions and Data Structures

## Description

Contains the BCS specific definitions and data structures used in the BCS APIs.

## Data Structures

- struct CYBLE_BCS_CHAR_VALUE_T
- struct CYBLE_BCS_DESCR_VALUE_T
- struct CYBLE_BCSS_CHAR_T
- struct CYBLE_BCSS_T
- struct CYBLE_BCSC_CHAR_T
- struct CYBLE_BCSC_T

## Enumerations

- enum CYBLE_BCS_CHAR_INDEX_T
- enum CYBLE_BCS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_BCS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_BCS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BCS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BCS_CHAR_INDEX_T CYBLE_BCS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_BCS_CHAR_VALUE_T::value**
Characteristic value

### struct CYBLE_BCS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_BCS_CHAR_INDEX_T charIndex
- CYBLE_BCS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BCS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BCS_CHAR_INDEX_T CYBLE_BCS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_BCS_DESCR_INDEX_T CYBLE_BCS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_VALUE_T* CYBLE_BCS_DESCR_VALUE_T::value**
Characteristic value

### struct CYBLE_BCSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_BCS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSS_CHAR_T::charHandle**
Handle of Characteristic Value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSS_CHAR_T::descrHandle[CYBLE_BCS_DESCR_COUNT]**
Array of Descriptor handles

### struct CYBLE_BCSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_BCSS_CHAR_T charInfo [CYBLE_BCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSS_T::serviceHandle**
Body Composition Service handle

**CYBLE_BCSS_CHAR_T CYBLE_BCSS_T::charInfo[CYBLE_BCS_CHAR_COUNT]**
Array of characteristics and descriptors handles

**struct CYBLE_BCSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSC_CHAR_T::valueHandle**
Handle of characteristic value

**uint8 CYBLE_BCSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSC_CHAR_T::endHandle**
End handle of a characteristic

**struct CYBLE_BCSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_BCSC_CHAR_T charInfo [CYBLE_BCS_CHAR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T bodyCompositionMeasurementCccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSC_T::serviceHandle**
Body Composition Service handle

**CYBLE_BCSC_CHAR_T CYBLE_BCSC_T::charInfo[CYBLE_BCS_CHAR_COUNT]**
Body Composition Service characteristics info structure

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BCSC_T::bodyCompositionMeasurementCccdHandle**
Body Composition Measurement Client Characteristic Configuration handle

## Enumeration Type Documentation

**enum CYBLE_BCS_CHAR_INDEX_T**
BCS Characteristic indexes
**Enumerator**
***CYBLE_BCS_BODY_COMPOSITION_FEATURE*** Body Composition Feature Characteristic index
***CYBLE_BCS_BODY_COMPOSITION_MEASUREMENT*** Body Composition Measurement Characteristic index
***CYBLE_BCS_CHAR_COUNT*** Total count of BCS Characteristics

**enum CYBLE_BCS_DESCR_INDEX_T**
BCS Characteristic Descriptors indexes
**Enumerator**

***CYBLE_BCS_CCCD*** Client Characteristic Configuration Descriptor index
***CYBLE_BCS_DESCR_COUNT*** Total count of Descriptors

# Blood Pressure Service (BLS)

## Description

The Blood Pressure Service exposes blood pressure and other data related to a non-invasive blood pressure monitor for consumer and professional healthcare applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The BLS API names begin with CyBle_Bls. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- BLS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- BLS Server Functions
  *APIs unique to BLS designs configured as a GATT Server role.*
- BLS Client Functions
  *APIs unique to BLS designs configured as a GATT Client role.*
- BLS Definitions and Data Structures
  *Contains the BLS specific definitions and data structures used in the BLS APIs.*

# BLS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Bls

## Functions

- void CyBle_BlsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_BlsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for Blood Pressure Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_BASS_NOTIFICATION_ENABLED) |

| | |
|---|---|
| | • eventParam contains the parameters corresponding to the current event (e.g. Pointer to CYBLE_BLS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

# BLS Server Functions

## Description

APIs unique to BLS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Blss

## Functions

- CYBLE_API_RESULT_T CyBle_BlssSetCharacteristicValue (CYBLE_BLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BlssGetCharacteristicValue (CYBLE_BLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BlssGetCharacteristicDescriptor (CYBLE_BLS_CHAR_INDEX_T charIndex, CYBLE_BLS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BlssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BlssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BlssSetCharacteristicValue (CYBLE_BLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the value of a characteristic which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_BlssGetCharacteristicValue (CYBLE_BLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Blood pressure service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_BlssGetCharacteristicDescriptor (CYBLE_BLS_CHAR_INDEX_T *charIndex*, CYBLE_BLS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Blood pressure service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent

**CYBLE_API_RESULT_T CyBle_BlssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_BLSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

**CYBLE_API_RESULT_T CyBle_BlssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication of the specified characteristic to the Client device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_BLSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BLS service-specific callback is registered (with CyBle_BlsRegisterAttrCallback):

- CYBLE_EVT_BLSS_INDICATION_CONFIRMED - In case if the indication is successfully delivered to the peer device.

Otherwise (if the BLS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - In case if the indication is successfully delivered to the peer device.

# BLS Client Functions

## Description

APIs unique to BLS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Blsc

## Functions

- CYBLE_API_RESULT_T CyBle_BlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BLS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_BlscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BLS_CHAR_INDEX_T charIndex, CYBLE_BLS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BlscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BLS_CHAR_INDEX_T charIndex, CYBLE_BLS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BLS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BLS service-specific callback is registered (with CyBle_BlsRegisterAttrCallback):

- CYBLE_EVT_BLSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_BLS_CHAR_VALUE_T.

Otherwise (if the BLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_BlscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BLS_CHAR_INDEX_T *charIndex*, CYBLE_BLS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)

Sends a request to set characteristic descriptor of specified Blood Pressure Service characteristic on the server device.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_BLSS_INDICATION_ENABLED
- CYBLE_EVT_BLSS_INDICATION_DISABLED
- CYBLE_EVT_BLSS_NOTIFICATION_ENABLED
- CYBLE_EVT_BLSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid

- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BLS service-specific callback is registered (with CyBle_BlsRegisterAttrCallback):

- CYBLE_EVT_BLSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_BLS_DESCR_VALUE_T.

Otherwise (if the BLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BlscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BLS_CHAR_INDEX_T *charIndex*, CYBLE_BLS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get characteristic descriptor of specified Blood Pressure Service characteristic from the server device. This function call can result in the generation of the following events based on the response from the server device.

- CYBLE_EVT_BLSC_READ_DESCR_RESPONSE
- CYBLE_EVT_GATTC_ERROR_RSP

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BLS service-specific callback is registered (with CyBle_BlsRegisterAttrCallback):

- CYBLE_EVT_BLSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_BLS_DESCR_VALUE_T.

Otherwise (if the BLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# BLS Definitions and Data Structures

## Description

Contains the BLS specific definitions and data structures used in the BLS APIs.

## Data Structures

- struct CYBLE_BLSS_CHAR_T
- struct CYBLE_BLSS_T
- struct CYBLE_BLSC_CHAR_T
- struct CYBLE_BLSC_T
- struct CYBLE_BLS_CHAR_VALUE_T
- struct CYBLE_BLS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_BLS_CHAR_INDEX_T
- enum CYBLE_BLS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_BLSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSS_CHAR_T::charHandle**

Blood Pressure Service characteristic's handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSS_CHAR_T::cccdHandle**

Blood Pressure Service char. descriptor's handle

### struct CYBLE_BLSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_BLSS_CHAR_T charInfo [CYBLE_BLS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSS_T::serviceHandle**

Blood Pressure Service handle

**CYBLE_BLSS_CHAR_T CYBLE_BLSS_T::charInfo[CYBLE_BLS_CHAR_COUNT]**

Array of Blood Pressure Service Characteristics + Descriptors handles

### struct CYBLE_BLSC_CHAR_T

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle

- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_BLSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSC_CHAR_T::valueHandle**
Handle of server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSC_CHAR_T::cccdHandle**
Blood Pressure client char. config. descriptor's handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BLSC_CHAR_T::endHandle**
Characteristic end handle

**struct CYBLE_BLSC_T**

**Data Fields**

- CYBLE_BLSC_CHAR_T charInfo [CYBLE_BLS_CHAR_COUNT]

**Field Documentation**

**CYBLE_BLSC_CHAR_T CYBLE_BLSC_T::charInfo[CYBLE_BLS_CHAR_COUNT]**
Structure with Characteristic handles + properties of Blood Pressure Service

**struct CYBLE_BLS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_BLS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BLS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BLS_CHAR_INDEX_T CYBLE_BLS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_BLS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_BLS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_BLS_CHAR_INDEX_T charIndex
- CYBLE_BLS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BLS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BLS_CHAR_INDEX_T CYBLE_BLS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_BLS_DESCR_INDEX_T CYBLE_BLS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_BLS_DESCR_VALUE_T::value**
Descriptor value

### Enumeration Type Documentation

**enum CYBLE_BLS_CHAR_INDEX_T**
Service Characteristics indexes
**Enumerator**
***CYBLE_BLS_BPM*** Blood Pressure Measurement characteristic index
***CYBLE_BLS_ICP*** Intermediate Cuff Pressure Context characteristic index
***CYBLE_BLS_BPF*** Blood Pressure Feature characteristic index
***CYBLE_BLS_CHAR_COUNT*** Total count of BLS characteristics

**enum CYBLE_BLS_DESCR_INDEX_T**
Service Characteristic Descriptors indexes
**Enumerator**
***CYBLE_BLS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_BLS_DESCR_COUNT*** Total count of BLS descriptors

## Bond Management Service (BMS)

### Description

The Bond Management Service defines how a peer Bluetooth device can manage the storage of bond information, especially the deletion of it, on the Bluetooth device supporting this service.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The BMS API names begin with CyBle_Bms. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- BMS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- BMS Server Functions
  *APIs unique to BMS designs configured as a GATT Server role.*
- BMS Client Functions
  *APIs unique to BMS designs configured as a GATT Client role.*
- BMS Definitions and Data Structures
  *Contains the BMS specific definitions and data structures used in the BMS APIs.*

## BMS Server and Client Function

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Bms

### Functions

- void CyBle_BmsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_BmsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for BM Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

## BMS Server Functions

### Description

APIs unique to BMS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Bmss

### Functions

- CYBLE_API_RESULT_T CyBle_BmssSetCharacteristicValue (CYBLE_BMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BmssGetCharacteristicValue (CYBLE_BMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BmssSetCharacteristicDescriptor (CYBLE_BMS_CHAR_INDEX_T charIndex, CYBLE_BMS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BmssGetCharacteristicDescriptor (CYBLE_BMS_CHAR_INDEX_T charIndex, CYBLE_BMS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_BmssSetCharacteristicValue (CYBLE_BMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_BmssGetCharacteristicValue (CYBLE_BMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where Characteristic value data should be stored. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_BmssSetCharacteristicDescriptor (CYBLE_BMS_CHAR_INDEX_T *charIndex*, CYBLE_BMS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_BMS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_BMS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_BmssGetCharacteristicDescriptor (CYBLE_BMS_CHAR_INDEX_T *charIndex*, CYBLE_BMS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_BMS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_BMS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

# BMS Client Functions

## Description

APIs unique to BMS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Bmsc

## Functions

- CYBLE_API_RESULT_T CyBle_BmscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BMS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_BmscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BmscReliableWriteCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_BmscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_BMS_CHAR_INDEX_T charIndex, CYBLE_BMS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_BmscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BMS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic value from a server which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BMS service-specific callback is registered (with CyBle_BmsRegisterAttrCallback):

- CYBLE_EVT_BMSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_BMS_CHAR_VALUE_T.

Otherwise (if the BMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BmscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute to the server. The function supports a long write procedure - it depends on the attrSize parameter - if it is larger than the current MTU size - 1, then the long write will be executed. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_BMSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

The Write response just confirms the operation success.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BMS service-specific callback is registered (with CyBle_BmsRegisterAttrCallback):

- CYBLE_EVT_BMSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_BMS_CHAR_VALUE_T.

Otherwise (if the BMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BmscReliableWriteCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to perform a reliable write command for the Bond Management Control Point characteristic (identified by charIndex) value attribute to the server.

The Write response just confirms the operation success.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BMS service-specific callback is registered (with CyBle_BmsRegisterAttrCallback):

- CYBLE_EVT_BMSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_BMS_CHAR_VALUE_T.

Otherwise (if the BMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_BmscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_BMS_CHAR_INDEX_T *charIndex*, CYBLE_BMS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the BMS service-specific callback is registered (with CyBle_BmsRegisterAttrCallback):

- CYBLE_EVT_BMSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_BMS_DESCR_VALUE_T.

Otherwise (if the BMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# BMS Definitions and Data Structures

## Description

Contains the BMS specific definitions and data structures used in the BMS APIs.

## Data Structures

- struct CYBLE_BMSS_CHAR_T
- struct CYBLE_BMSS_T
- struct CYBLE_BMSC_CHAR_T
- struct CYBLE_BMSC_T
- struct CYBLE_BMS_CHAR_VALUE_T
- struct CYBLE_BMS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_BMS_CHAR_INDEX_T
- enum CYBLE_BMS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_BMSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_BMS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSS_CHAR_T::charHandle**

Handle of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSS_CHAR_T::descrHandle[CYBLE_BMS_DESCR_COUNT]**

Handles of Descriptors

### struct CYBLE_BMSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_BMSS_CHAR_T charInfo [CYBLE_BMS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSS_T::serviceHandle**

Service handle

**CYBLE_BMSS_CHAR_T CYBLE_BMSS_T::charInfo[CYBLE_BMS_CHAR_COUNT]**
Service characteristics info array

**struct CYBLE_BMSC_CHAR_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_BMS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_BMSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSC_CHAR_T::valueHandle**
Handle of Server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSC_CHAR_T::descrHandle[CYBLE_BMS_DESCR_COUNT]**
Characteristics descriptors handles

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_BMSC_CHAR_T::endHandle**
Characteristic End Handle

**struct CYBLE_BMSC_T**

**Data Fields**

- CYBLE_BMSC_CHAR_T charInfo [CYBLE_BMS_CHAR_COUNT]

**Field Documentation**

**CYBLE_BMSC_CHAR_T CYBLE_BMSC_T::charInfo[CYBLE_BMS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_BMS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_BMS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BMS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BMS_CHAR_INDEX_T CYBLE_BMS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_BMS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_BMS_CHAR_VALUE_T::gattErrorCode**
GATT error code for checking the authorization code

**struct CYBLE_BMS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle

- CYBLE_BMS_CHAR_INDEX_T charIndex
- CYBLE_BMS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_BMS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_BMS_CHAR_INDEX_T CYBLE_BMS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_BMS_DESCR_INDEX_T CYBLE_BMS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_BMS_DESCR_VALUE_T::value**
Descriptor value

### Enumeration Type Documentation

**enum CYBLE_BMS_CHAR_INDEX_T**
Service Characteristics indexes

**Enumerator**

***CYBLE_BMS_BMCP*** Bond Management Control Point characteristic index

***CYBLE_BMS_BMFT*** Bond Management Feature characteristic index

***CYBLE_BMS_CHAR_COUNT*** Total count of BMS characteristics

**enum CYBLE_BMS_DESCR_INDEX_T**
Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_BMS_CEPD*** Characteristic Extended Properties descriptor index

***CYBLE_BMS_DESCR_COUNT*** Total count of BMS descriptors

## Continuous Glucose Monitoring Service (CGMS)

### Description

The Continuous Glucose Monitoring Service exposes glucose measurement and other data related to a personal CGM sensor for healthcare applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The CGMS API names begin with CyBle_Cgms. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- CGMS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- CGMS Server Functions
  *APIs unique to CGMS designs configured as a GATT Server role.*
- CGMS Client Functions
  *APIs unique to CGMS designs configured as a GATT Client role.*

- CGMS Definitions and Data Structures
  *Contains the CGMS specific definitions and data structures used in the CGMS APIs.*

# CGMS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Cgms

## Functions

- void CyBle_CgmsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_CgmsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for CGM Service is, typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# CGMS Server Functions

## Description

APIs unique to CGMS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Cgmss

## Functions

- CYBLE_API_RESULT_T CyBle_CgmssSetCharacteristicValue (CYBLE_CGMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmssGetCharacteristicValue (CYBLE_CGMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmssSetCharacteristicDescriptor (CYBLE_CGMS_CHAR_INDEX_T charIndex, CYBLE_CGMS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmssGetCharacteristicDescriptor (CYBLE_CGMS_CHAR_INDEX_T charIndex, CYBLE_CGMS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_CgmssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CgmssSetCharacteristicValue (CYBLE_CGMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_CgmssGetCharacteristicValue (CYBLE_CGMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where Characteristic value data should be stored. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_CgmssSetCharacteristicDescriptor (CYBLE_CGMS_CHAR_INDEX_T *charIndex*, CYBLE_CGMS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CGMS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CGMS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_CgmssGetCharacteristicDescriptor (CYBLE_CGMS_CHAR_INDEX_T *charIndex*, CYBLE_CGMS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CGMS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CGMS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_CgmssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic to the client device, as defined by the charIndex value.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_CGMSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consists of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the Characteristic value data that should be sent to the client device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

**CYBLE_API_RESULT_T CyBle_CgmssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication of the specified characteristic to the client device, as defined by the charIndex value.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_CGMSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consists of the device ID and ATT connection ID. |

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the Characteristic value data that should be sent to Client device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CGMS service-specific callback is registered (with CyBle_CgmsRegisterAttrCallback):

- CYBLE_EVT_CGMSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the CGMS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# CGMS Client Functions

## Description

APIs unique to CGMS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Cgmsc

## Functions

- CYBLE_API_RESULT_T CyBle_CgmscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_CgmscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex, CYBLE_CGMS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CgmscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CGMS_CHAR_INDEX_T charIndex, CYBLE_CGMS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CgmscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_CGMSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CGMS service-specific callback is registered (with CyBle_CgmsRegisterAttrCallback):

- CYBLE_EVT_CGMSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_CGMS_CHAR_VALUE_T.

Otherwise (if the CGMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_CgmscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*)

This function is used to read the characteristic Value from a server identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CGMS service-specific callback is registered (with CyBle_CgmsRegisterAttrCallback):

- CYBLE_EVT_CGMSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_CGMS_CHAR_VALUE_T.

Otherwise (if the CGMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CgmscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*, CYBLE_CGMS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the Characteristic Descriptor of the specified characteristic.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_CGMSS_INDICATION_ENABLED
- CYBLE_EVT_CGMSS_INDICATION_DISABLED
- CYBLE_EVT_CGMSS_NOTIFICATION_ENABLED
- CYBLE_EVT_CGMSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CGMS service-specific callback is registered (with CyBle_CgmsRegisterAttrCallback):

- CYBLE_EVT_CGMSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_CGMS_DESCR_VALUE_T.

Otherwise (if the CGMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CgmscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CGMS_CHAR_INDEX_T *charIndex*, CYBLE_CGMS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CGMS service-specific callback is registered (with CyBle_CgmsRegisterAttrCallback):

- CYBLE_EVT_CGMSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_CGMS_DESCR_VALUE_T.

Otherwise (if the CGMS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# CGMS Definitions and Data Structures

## Description

Contains the CGMS specific definitions and data structures used in the CGMS APIs.

## Data Structures

- struct CYBLE_CGMSS_CHAR_T
- struct CYBLE_CGMSS_T
- struct CYBLE_CGMSC_CHAR_T
- struct CYBLE_CGMSC_T
- struct CYBLE_CGMS_CHAR_VALUE_T
- struct CYBLE_CGMS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_CGMS_CHAR_INDEX_T
- enum CYBLE_CGMS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_CGMSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_CGMS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CGMSS_CHAR_T::charHandle**
Handle of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_CGMSS_CHAR_T::descrHandle[CYBLE_CGMS_DESCR_COUNT]**
Handles of Descriptors

### struct CYBLE_CGMSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_CGMSS_CHAR_T charInfo [CYBLE_CGMS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CGMSS_T::serviceHandle**
CGM Service handle

**CYBLE_CGMSS_CHAR_T CYBLE_CGMSS_T::charInfo[CYBLE_CGMS_CHAR_COUNT]**
CGM Service characteristics info array

### struct CYBLE_CGMSC_CHAR_T

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_CGMS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_CGMSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CGMSC_CHAR_T::valueHandle**
Handle of Server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_CGMSC_CHAR_T::descrHandle[CYBLE_CGMS_DESCR_COUNT]**
Characteristics descriptors handles

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CGMSC_CHAR_T::endHandle**
Characteristic End Handle

### struct CYBLE_CGMSC_T

**Data Fields**

- CYBLE_CGMSC_CHAR_T charInfo [CYBLE_CGMS_CHAR_COUNT]

**Field Documentation**

**CYBLE_CGMSC_CHAR_T CYBLE_CGMSC_T::charInfo[CYBLE_CGMS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_CGMS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CGMS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CGMS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_CGMS_CHAR_INDEX_T CYBLE_CGMS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_CGMS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_CGMS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**struct CYBLE_CGMS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CGMS_CHAR_INDEX_T charIndex
- CYBLE_CGMS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CGMS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_CGMS_CHAR_INDEX_T CYBLE_CGMS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_CGMS_DESCR_INDEX_T CYBLE_CGMS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_CGMS_DESCR_VALUE_T::value**
Descriptor value

## Enumeration Type Documentation

**enum CYBLE_CGMS_CHAR_INDEX_T**
Service Characteristics indexes

**Enumerator**

***CYBLE_CGMS_CGMT*** CGM Measurement characteristic index

***CYBLE_CGMS_CGFT*** CGM Feature characteristic index

***CYBLE_CGMS_CGST*** CGM Status characteristic index

***CYBLE_CGMS_SSTM*** CGM Session Start Time characteristic index

***CYBLE_CGMS_SRTM*** CGM Session Run Time characteristic index
***CYBLE_CGMS_RACP*** Record Access Control Point characteristic index
***CYBLE_CGMS_SOCP*** CGM Specific Ops Control Point characteristic index
***CYBLE_CGMS_CHAR_COUNT*** Total count of CGMS characteristics

**enum CYBLE_CGMS_DESCR_INDEX_T**

Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_CGMS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_CGMS_DESCR_COUNT*** Total count of CGMS descriptors

# Cycling Power Service (CPS)

## Description

The Cycling Power Service (CPS) exposes power- and force-related data and optionally speed- and cadence-related data from a Cycling Power sensor (GATT Server) intended for sports and fitness applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The CPS API names begin with CyBle_Cps. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- CPS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- CPS Server Functions
  *APIs unique to CPS designs configured as a GATT Server role.*
- CPS Client Functions
  *APIs unique to CPS designs configured as a GATT Client role.*
- CPS Definitions and Data Structures
  *Contains the CPS specific definitions and data structures used in the CPS APIs.*

# CPS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Cps

## Functions

- void CyBle_CpsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_CpsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for CPS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

## CPS Server Functions

### Description

APIs unique to CPS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Cpss

### Functions

- CYBLE_API_RESULT_T CyBle_CpssSetCharacteristicValue (CYBLE_CPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssGetCharacteristicValue (CYBLE_CPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssSetCharacteristicDescriptor (CYBLE_CPS_CHAR_INDEX_T charIndex, CYBLE_CPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssGetCharacteristicDescriptor (CYBLE_CPS_CHAR_INDEX_T charIndex, CYBLE_CPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpssStartBroadcast (uint16 advInterval, uint8 attrSize, uint8 *attrValue)
- void CyBle_CpssStopBroadcast (void)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_CpssSetCharacteristicValue (CYBLE_CPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_CpssGetCharacteristicValue (CYBLE_CPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service, which is a value identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_CpssSetCharacteristicDescriptor (CYBLE_CPS_CHAR_INDEX_T *charIndex*, CYBLE_CPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CPS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_CpssGetCharacteristicDescriptor (CYBLE_CPS_CHAR_INDEX_T *charIndex*, CYBLE_CPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CPS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_CpssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends notification with a characteristic value of the CPS, which is a value specified by charIndex, to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_CPSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the Client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the Client.

**CYBLE_API_RESULT_T CyBle_CpssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends indication with a characteristic value of the CPS, which is a value specified by charIndex, to the Client device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_CPSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the Client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the Client

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CPS service-specific callback is registered (with CyBle_CpsRegisterAttrCallback):

- CYBLE_EVT_CPSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the CPS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

**CYBLE_API_RESULT_T CyBle_CpssStartBroadcast (uint16 *advInterval*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to start broadcasting of the Cycling Power Measurement characteristic or update broadcasting data when it was started before. For update broadcasting data this function must be called when CyBle_GetBleSsState() returns CYBLE_BLESS_STATE_EVENT_CLOSE state.

It is available only in Broadcaster role.

**Parameters:**

| | |
|---|---|
| *advInterval* | Advertising interval in 625 us units. The valid range is from CYBLE_GAP_ADV_ADVERT_INTERVAL_NONCON_MIN to CYBLE_GAP_ADV_ADVERT_INTERVAL_MAX. |
| *attrSize* | The size of the characteristic value attribute. This size is limited by maximum advertising packet length and advertising header size. |
| *attrValue* | The pointer to the Cycling Power Measurement characteristic that include the mandatory fields (e.g. the Flags field and the Instantaneous Power field) and depending on the Flags field, some optional fields in a non connectable undirected advertising event. |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On passing an invalid parameter. |
| CYBLE_ERROR_INVALID_OPERATION | ADV Event is not closed, BLESS is active or ADV is not enabled. |

**void CyBle_CpssStopBroadcast (void )**

This function is used to stop broadcasting of the Cycling Power Measurement characteristic.

# CPS Client Functions

## Description

APIs unique to CPS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Cpsc

## Functions

- CYBLE_API_RESULT_T CyBle_CpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_CpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_CpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex, CYBLE_CPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CPS_CHAR_INDEX_T charIndex, CYBLE_CPS_DESCR_INDEX_T descrIndex)
- CYBLE_API_RESULT_T CyBle_CpscStartObserve (void)
- void CyBle_CpscStopObserve (void)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_CPSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be send to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CPS service-specific callback is registered (with CyBle_CpsRegisterAttrCallback):

- CYBLE_EVT_CPSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_CPS_CHAR_VALUE_T.

Otherwise (if the CPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

The Read Response returns the characteristic Value in the Attribute Value parameter.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CPS service-specific callback is registered (with CyBle_CpsRegisterAttrCallback):

- CYBLE_EVT_CPSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_CPS_CHAR_VALUE_T.

Otherwise (if the CPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_CpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*, CYBLE_CPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)

This function is used to write the characteristic descriptor to the server which is identified by charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_CPSS_NOTIFICATION_ENABLED
- CYBLE_EVT_CPSS_NOTIFICATION_DISABLED
- CYBLE_EVT_CPSS_INDICATION_ENABLED
- CYBLE_EVT_CPSS_INDICATION_DISABLED
- CYBLE_EVT_CPSS_BROADCAST_ENABLED
- CYBLE_EVT_CPSS_BROADCAST_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CPS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CPS service-specific callback is registered (with CyBle_CpsRegisterAttrCallback):

- CYBLE_EVT_CPSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_CPS_DESCR_VALUE_T.

Otherwise (if the CPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CPS_CHAR_INDEX_T *charIndex*, CYBLE_CPS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CPS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CPS_DESCR_INDEX_T. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CPS service-specific callback is registered (with CyBle_CpsRegisterAttrCallback):

- CYBLE_EVT_CPSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_CPS_DESCR_VALUE_T.

Otherwise (if the CPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CpscStartObserve (void )**

This function is used for observing GAP peripheral devices. A device performing the observer role receives only advertisement data from devices irrespective of their discoverable mode settings. Advertisement data received is provided by the event, CYBLE_EVT_CPSC_SCAN_PROGRESS_RESULT. This procedure sets the scanType sub parameter to passive scanning.

If 'scanTo' sub-parameter is set to zero value, then passive scanning procedure will continue until you call CyBle_GapcStopObserve(). Possible generated events are:

- CYBLE_EVT_CPSC_SCAN_PROGRESS_RESULT.

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation. |
| CYBLE_ERROR_INVALID_PARAMETER | On specifying NULL as input parameter for 'scanInfo' or if any element within 'scanInfo' has an invalid value. |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed. |

**void CyBle_CpscStopObserve (void )**

This function used to stop the discovery of devices. On stopping discovery operation, CYBLE_EVT_GAPC_SCAN_START_STOP event is generated. Application layer needs to keep track of the function call made before receiving this event to associate this event with either the start or stop discovery function.

Possible events generated are:

- CYBLE_EVT_GAPC_SCAN_START_STOP.

# CPS Definitions and Data Structures

## Description

Contains the CPS specific definitions and data structures used in the CPS APIs.

## Data Structures

- struct CYBLE_CPSS_CHAR_T
- struct CYBLE_CPSS_T
- struct CYBLE_CPSC_CHAR_T
- struct CYBLE_CPSC_T
- struct CYBLE_CPS_CHAR_VALUE_T
- struct CYBLE_CPS_DESCR_VALUE_T
- struct __attribute__

## Enumerations

- enum CYBLE_CPS_CHAR_INDEX_T
- enum CYBLE_CPS_DESCR_INDEX_T
- enum CYBLE_CPS_CP_OC_T
- enum CYBLE_CPS_CP_RC_T

- enum CYBLE_CPS_SL_VALUE_T

## Data Structure Documentation

### struct CYBLE_CPSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_CPS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSS_CHAR_T::charHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSS_CHAR_T::descrHandle[CYBLE_CPS_DESCR_COUNT]**
Handle of descriptor

### struct CYBLE_CPSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_CPSS_CHAR_T charInfo [CYBLE_CPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSS_T::serviceHandle**
Cycling Power Service handle

**CYBLE_CPSS_CHAR_T CYBLE_CPSS_T::charInfo[CYBLE_CPS_CHAR_COUNT]**
Cycling Power Service Characteristic handles

### struct CYBLE_CPSC_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_CPS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSC_CHAR_T::descrHandle[CYBLE_CPS_DESCR_COUNT]**
Handles of descriptors

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSC_CHAR_T::valueHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CPSC_CHAR_T::endHandle**
End handle of characteristic

**uint8 CYBLE_CPSC_CHAR_T::properties**
Properties for value field

### struct CYBLE_CPSC_T

**Data Fields**

- CYBLE_CPSC_CHAR_T charInfo [CYBLE_CPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_CPSC_CHAR_T CYBLE_CPSC_T::charInfo[CYBLE_CPS_CHAR_COUNT]**
Characteristics handles array

**struct CYBLE_CPS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CPS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CPS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_CPS_CHAR_INDEX_T CYBLE_CPS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_CPS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_CPS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CPS_CHAR_INDEX_T charIndex
- CYBLE_CPS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CPS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_CPS_CHAR_INDEX_T CYBLE_CPS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_CPS_DESCR_INDEX_T CYBLE_CPS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_VALUE_T* CYBLE_CPS_DESCR_VALUE_T::value**
Characteristic value

**struct __attribute__**

**Data Fields**

- uint16 year
- uint8 month
- uint8 day
- uint8 hours
- uint8 minutes
- uint8 seconds
- uint16 crankLength
- uint16 chainLength
- uint16 chainWeight
- uint16 spanLength
- CYBLE_CPS_DATE_TIME_T factoryCalibrationDate
- uint8 samplingRate

- int16 offsetCompensation

**Field Documentation**

**uint16 __attribute__::year**
Year

**uint8 __attribute__::month**
Month

**uint8 __attribute__::day**
Day

**uint8 __attribute__::hours**
Time - hours

**uint8 __attribute__::minutes**
Time - minutes

**uint8 __attribute__::seconds**
Time - seconds

**uint16 __attribute__::crankLength**
In millimeters with a resolution of 1/2 millimeter

**uint16 __attribute__::chainLength**
In millimeters with a resolution of 1 millimeter

**uint16 __attribute__::chainWeight**
In grams with a resolution of 1 gram

**uint16 __attribute__::spanLength**
In millimeters with a resolution of 1 millimeter

**CYBLE_CPS_DATE_TIME_T __attribute__::factoryCalibrationDate**
Use the same format as the Date Time characteristic

**uint8 __attribute__::samplingRate**
In Hertz with a resolution of 1 Hertz

**int16 __attribute__::offsetCompensation**
Either the raw force in Newton or the raw torque in 1/32 Newton meter based on the server capabilities. 0xFFFF means "Not Available"

## Enumeration Type Documentation

**enum CYBLE_CPS_CHAR_INDEX_T**
Characteristic indexes

**Enumerator**

***CYBLE_CPS_POWER_MEASURE*** Cycling Power Measurement characteristic index

***CYBLE_CPS_POWER_FEATURE*** Cycling Power Feature characteristic index

***CYBLE_CPS_SENSOR_LOCATION*** Sensor Location characteristic index

***CYBLE_CPS_POWER_VECTOR*** Cycling Power Vector characteristic index

***CYBLE_CPS_POWER_CP*** Cycling Power Control Point characteristic index

***CYBLE_CPS_CHAR_COUNT*** Total count of CPS characteristics

**enum CYBLE_CPS_DESCR_INDEX_T**
Characteristic Descriptors indexes

**Enumerator**

***CYBLE_CPS_CCCD*** Client Characteristic Configuration descriptor index

***CYBLE_CPS_SCCD*** Handle of the Server Characteristic Configuration descriptor

***CYBLE_CPS_DESCR_COUNT*** Total count of descriptors

**enum CYBLE_CPS_CP_OC_T**

Op Codes of the Cycling Power Control Point characteristic

**Enumerator**

***CYBLE_CPS_CP_OC_SCV*** Set Cumulative Value

***CYBLE_CPS_CP_OC_USL*** Update Sensor Location

***CYBLE_CPS_CP_OC_RSSL*** Request Supported Sensor Locations

***CYBLE_CPS_CP_OC_SCRL*** Set Crank Length

***CYBLE_CPS_CP_OC_RCRL*** Request Crank Length

***CYBLE_CPS_CP_OC_SCHL*** Set Chain Length

***CYBLE_CPS_CP_OC_RCHL*** Request Chain Length

***CYBLE_CPS_CP_OC_SCHW*** Set Chain Weight

***CYBLE_CPS_CP_OC_RCHW*** Request Chain Weight

***CYBLE_CPS_CP_OC_SSL*** Set Span Length

***CYBLE_CPS_CP_OC_RSL*** Request Span Length

***CYBLE_CPS_CP_OC_SOC*** Start Offset Compensation

***CYBLE_CPS_CP_OC_MCPMCC*** Mask Cycling Power Measurement Characteristic Content

***CYBLE_CPS_CP_OC_RSR*** Request Sampling Rate

***CYBLE_CPS_CP_OC_RFCD*** Request Factory Calibration Date

***CYBLE_CPS_CP_OC_SEOC*** Start Enhanced Offset Compensation

***CYBLE_CPS_CP_OC_RC*** Response Code

**enum CYBLE_CPS_CP_RC_T**

Response Code of the Cycling Power Control Point characteristic

**Enumerator**

***CYBLE_CPS_CP_RC_SUCCESS*** Response for successful operation.

***CYBLE_CPS_CP_RC_NOT_SUPPORTED*** Response if unsupported Op Code is received

***CYBLE_CPS_CP_RC_INVALID_PARAMETER*** Response if Parameter received does not meet the requirements of the service or is outside of the supported range of the Sensor

***CYBLE_CPS_CP_RC_OPERATION_FAILED*** Response if the requested procedure failed

**enum CYBLE_CPS_SL_VALUE_T**

Sensor Location characteristic value

**Enumerator**

***CYBLE_CPS_SL_OTHER*** Sensor Location - Other

***CYBLE_CPS_SL_TOP_OF_SHOE*** Sensor Location - Top of shoe

***CYBLE_CPS_SL_IN_SHOE*** Sensor Location - In shoe

***CYBLE_CPS_SL_HIP*** Sensor Location - Hip

***CYBLE_CPS_SL_FRONT_WHEEL*** Sensor Location - Front Wheel

***CYBLE_CPS_SL_LEFT_CRANK*** Sensor Location - Left Crank

***CYBLE_CPS_SL_RIGHT_CRANK*** Sensor Location - Right Crank

***CYBLE_CPS_SL_LEFT_PEDAL*** Sensor Location - Left Pedal
***CYBLE_CPS_SL_RIGHT_PEDAL*** Sensor Location - Right Pedal
***CYBLE_CPS_SL_FRONT_HUB*** Sensor Location - Front Hub
***CYBLE_CPS_SL_REAR_DROPOUT*** Sensor Location - Rear Dropout
***CYBLE_CPS_SL_CHAINSTAY*** Sensor Location - Chainstay
***CYBLE_CPS_SL_REAR_WHEEL*** Sensor Location - Rear Wheel
***CYBLE_CPS_SL_REAR_HUB*** Sensor Location - Rear Hub
***CYBLE_CPS_SL_CHEST*** Sensor Location - Chest
***CYBLE_CPS_SL_SPIDER*** Sensor Location - Spider
***CYBLE_CPS_SL_CHAIN_RING*** Sensor Location - Chain Ring
***CYBLE_CPS_SL_COUNT*** Total count of SL characteristics

# Cycling Speed and Cadence Service (CSCS)

## Description

The Cycling Speed and Cadence (CSC) Service exposes speed-related data and/or cadence-related data while using the Cycling Speed and Cadence sensor (Server).

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The CSCS API names begin with CyBle_Cscs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- CSCS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- CSCS Server Functions
  *APIs unique to CSCS designs configured as a GATT Server role.*
- CSCS Client Functions
  *APIs unique to CSCS designs configured as a GATT Client role.*
- CSCS Definitions and Data Structures
  *Contains the CSCS specific definitions and data structures used in the CSCS APIs.*

# CSCS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Cscs

## Functions

- void CyBle_CscsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_CscsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for Cycling Speed and Cadence Service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for CSCS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

## CSCS Server Functions

### Description

APIs unique to CSCS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Cscss

### Functions

- CYBLE_API_RESULT_T CyBle_CscssSetCharacteristicValue (CYBLE_CSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscssGetCharacteristicValue (CYBLE_CSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscssGetCharacteristicDescriptor (CYBLE_CSCS_CHAR_INDEX_T charIndex, CYBLE_CSCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_CscssSetCharacteristicValue (CYBLE_CSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets characteristic value of the Cycling Speed and Cadence Service, which is identified by charIndex, to the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CSCS_CHAR_INDEX_T. Valid values are,<br>• CYBLE_CSCS_CSC_FEATURE<br>• CYBLE_CSCS_SENSOR_LOCATION. |

| | |
|---|---|
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**CYBLE_API_RESULT_T CyBle_CscssGetCharacteristicValue (CYBLE_CSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Cycling Speed and Cadence Service, which is identified by charIndex, from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CSCS_CHAR_INDEX_T. Valid value is,<br>• CYBLE_CSCS_SC_CONTROL_POINT. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_CscssGetCharacteristicDescriptor (CYBLE_CSCS_CHAR_INDEX_T *charIndex*, CYBLE_CSCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Cycling Speed and Cadence Service, from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_CSCS_CHAR_INDEX_T. Valid values are,<br>• CYBLE_CSCS_CSC_MEASUREMENT<br>• CYBLE_CSCS_SC_CONTROL_POINT. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_CSCS_DESCR_INDEX_T. Valid value is<br>• CYBLE_CSCS_CCCD. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**CYBLE_API_RESULT_T CyBle_CscssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends notification with a characteristic value, which is specified by charIndex, of the Cycling Speed and Cadence Service to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_CSCSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CSCS_CHAR_INDEX_T. Valid value is<br>• CYBLE_CSCS_CSC_MEASUREMENT. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter is failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

**CYBLE_API_RESULT_T CyBle_CscssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends indication with a characteristic value, which is specified by charIndex, of the Cycling Speed and Cadence Service to the Client device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_CSCSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_CSCS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter is failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CSCS service-specific callback is registered (with CyBle_CscsRegisterAttrCallback):

- CYBLE_EVT_CSCSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the CSCS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# CSCS Client Functions

## Description

APIs unique to CSCS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Cscsc

## Functions

- CYBLE_API_RESULT_T CyBle_CscscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_CscscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex, CYBLE_CSCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CscscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CSCS_CHAR_INDEX_T charIndex, CYBLE_CSCS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CscscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_CSCSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | Size of the characteristic value attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully;
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CSCS service-specific callback is registered (with CyBle_CscsRegisterAttrCallback):

- CYBLE_EVT_CSCSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_CSCS_CHAR_VALUE_T.

Otherwise (if the CSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CscscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*)**

Sends a request to peer device to get characteristic value of the Cycling Speed and Cadence Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully;
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CSCS service-specific callback is registered (with CyBle_CscsRegisterAttrCallback):

- CYBLE_EVT_CSCSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_CSCS_CHAR_VALUE_T.

Otherwise (if the CSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CscscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*, CYBLE_CSCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a request to peer device to get characteristic descriptor of specified characteristic of the Cycling Speed and Cadence Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_CSCSS_NOTIFICATION_ENABLED
- CYBLE_EVT_CSCSS_NOTIFICATION_DISABLED
- CYBLE_EVT_CSCSS_INDICATION_ENABLED
- CYBLE_EVT_CSCSS_INDICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a CSCS characteristic. |
| *descrIndex* | The index of a CSCS characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CSCS service-specific callback is registered (with CyBle_CscsRegisterAttrCallback):

- CYBLE_EVT_CSCSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_CSCS_DESCR_VALUE_T.

Otherwise (if the CSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CscscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CSCS_CHAR_INDEX_T *charIndex*, CYBLE_CSCS_DESCR_INDEX_T *descrIndex*)**

Sends a request to peer device to get characteristic descriptor of specified characteristic of the Cycling Speed and Cadence Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a Service Characteristic. |
| *descrIndex* | The index of a Service Characteristic Descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the Client is not established.

- CYBLE_ERROR_INVALID_OPERATION - Cannot process a request to send PDU due to invalid operation performed by the application.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CSCS service-specific callback is registered (with CyBle_CscsRegisterAttrCallback):

- CYBLE_EVT_CSCSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_CSCS_DESCR_VALUE_T.

Otherwise (if the CSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# CSCS Definitions and Data Structures

## Description

Contains the CSCS specific definitions and data structures used in the CSCS APIs.

## Data Structures

- struct CYBLE_CSCS_CHAR_VALUE_T
- struct CYBLE_CSCS_DESCR_VALUE_T
- struct CYBLE_CSCSS_CHAR_T
- struct CYBLE_CSCSS_T
- struct CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T
- struct CYBLE_CSCSC_T

## Enumerations

- enum CYBLE_CSCS_CHAR_INDEX_T
- enum CYBLE_CSCS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_CSCS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CSCS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CSCS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_CSCS_CHAR_INDEX_T CYBLE_CSCS_CHAR_VALUE_T::charIndex**
Index of Cycling Speed and Cadence Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_CSCS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_CSCS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CSCS_CHAR_INDEX_T charIndex
- CYBLE_CSCS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CSCS_DESCR_VALUE_T::connHandle**
Connection handle

**CYBLE_CSCS_CHAR_INDEX_T CYBLE_CSCS_DESCR_VALUE_T::charIndex**
Characteristic index of the Service

**CYBLE_CSCS_DESCR_INDEX_T CYBLE_CSCS_DESCR_VALUE_T::descrIndex**
Characteristic Descriptor index

**CYBLE_GATT_VALUE_T* CYBLE_CSCS_DESCR_VALUE_T::value**
Pointer to value of the Service Characteristic Descriptor

**struct CYBLE_CSCSS_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_CSCS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CSCSS_CHAR_T::charHandle**
Handle of the Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_CSCSS_CHAR_T::descrHandle[CYBLE_CSCS_DESCR_COUNT]**
Handles of the Descriptors

**struct CYBLE_CSCSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_CSCSS_CHAR_T charInfo [CYBLE_CSCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CSCSS_T::serviceHandle**
Cycling Speed and Cadence Service handle

**CYBLE_CSCSS_CHAR_T CYBLE_CSCSS_T::charInfo[CYBLE_CSCS_CHAR_COUNT]**
Array of Cycling Speed and Cadence Service Characteristics and Descriptors handles

**struct CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo

- CYBLE_GATT_DB_ATTR_HANDLE_T descriptors [CYBLE_CSCS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T::charInfo**
Characteristic handle and properties

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T::descriptors[CYBLE_CSCS_DESCR_COUNT]**
Characteristic descriptors handles

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T::endHandle**
End handle of Characteristic

**struct CYBLE_CSCSC_T**

**Data Fields**

- CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T characteristics [CYBLE_CSCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_CSCSC_SRVR_FULL_CHAR_INFO_T**
**CYBLE_CSCSC_T::characteristics[CYBLE_CSCS_CHAR_COUNT]**
Characteristics handles array

### Enumeration Type Documentation

**enum CYBLE_CSCS_CHAR_INDEX_T**
Characteristic indexes
**Enumerator**
***CYBLE_CSCS_CSC_MEASUREMENT*** CSC Measurement Characteristic index
***CYBLE_CSCS_CSC_FEATURE*** CSC Feature Characteristic index
***CYBLE_CSCS_SENSOR_LOCATION*** CSC Sensor Location Characteristic index
***CYBLE_CSCS_SC_CONTROL_POINT*** CSC SC Control Point Characteristic index
***CYBLE_CSCS_CHAR_COUNT*** Total count of CSCS Characteristics

**enum CYBLE_CSCS_DESCR_INDEX_T**
Characteristic Descriptors indexes
**Enumerator**
***CYBLE_CSCS_CCCD*** Client Characteristic Configuration Descriptor index
***CYBLE_CSCS_DESCR_COUNT*** Total count of Descriptors

## Current Time Service (CTS)

### Description

The Current Time Service defines how a Bluetooth device can expose time information to other Bluetooth devices.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The CTS API names begin with CyBle_Cts. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- CTS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- CTS Server Functions
  *APIs unique to CTS designs configured as a GATT Server role.*
- CTS Client Functions
  *APIs unique to CTS designs configured as a GATT Client role.*
- CTS Definitions and Data Structures
  *Contains the CTS specific definitions and data structures used in the CTS APIs.*

## CTS Server and Client Function

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Cts

### Functions

- void CyBle_CtsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_CtsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for Current Time Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_CTSS_NOTIFICATION_ENABLED)<br>• eventParam contains the parameters corresponding to the current event (e.g. Pointer to CYBLE_CTS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

## CTS Server Functions

### Description

APIs unique to CTS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Ctss

## Functions

- CYBLE_API_RESULT_T CyBle_CtssSetCharacteristicValue (CYBLE_CTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CtssGetCharacteristicValue (CYBLE_CTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CtssGetCharacteristicDescriptor (CYBLE_CTS_CHAR_INDEX_T charIndex, CYBLE_CTS_CHAR_DESCRIPTORS_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CtssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CtssSetCharacteristicValue (CYBLE_CTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a value for one of three characteristic values of the Current Time Service. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the Current Time Service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was written successfully.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_CtssGetCharacteristicValue (CYBLE_CTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Current Time Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a Current Time Service characteristic. |
| *attrSize* | The size of the Current Time Service characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_CtssGetCharacteristicDescriptor (CYBLE_CTS_CHAR_INDEX_T *charIndex*, CYBLE_CTS_CHAR_DESCRIPTORS_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Current Time Service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value. |

| | |
|---|---|
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_CtssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification to the client's device. A characteristic value also gets written to the GATT database.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_CTSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic to be send as a notification to the Client device. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic notification was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# CTS Client Functions

## Description

APIs unique to CTS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Ctsc

## Functions

- CYBLE_API_RESULT_T CyBle_CtscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CtscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CTS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_CtscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CTS_CHAR_INDEX_T charIndex, CYBLE_CTS_CHAR_DESCRIPTORS_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_CtscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_CTS_CHAR_INDEX_T charIndex, uint8 descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_CtscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_CTSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CTS service-specific callback is registered (with CyBle_CtsRegisterAttrCallback):

- CYBLE_EVT_CTSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_CTS_CHAR_VALUE_T.

Otherwise (if the CTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CtscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CTS_CHAR_INDEX_T *charIndex*)**

Gets a characteristic value of the Current Time Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CTS service-specific callback is registered (with CyBle_CtsRegisterAttrCallback):

- CYBLE_EVT_CTSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_CTS_CHAR_VALUE_T.

Otherwise (if the CTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CtscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CTS_CHAR_INDEX_T *charIndex*, CYBLE_CTS_CHAR_DESCRIPTORS_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor of the Current Time Characteristic of the Current Time Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_CTSS_NOTIFICATION_ENABLED.
- CYBLE_EVT_CTSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the Current Time Service characteristic. |
| *descrIndex* | The index of the Current Time Service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on specified attribute.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CTS service-specific callback is registered (with CyBle_CtsRegisterAttrCallback):

- CYBLE_EVT_CTSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_CTS_DESCR_VALUE_T.

Otherwise (if the CTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.

- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_CtscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_CTS_CHAR_INDEX_T *charIndex*, uint8 *descrIndex*)**

Gets a characteristic descriptor of the Current Time Characteristic of the Current Time Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - State is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the CTS service-specific callback is registered (with CyBle_CtsRegisterAttrCallback):

- CYBLE_EVT_CTSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_CTS_DESCR_VALUE_T.

Otherwise (if the CTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# CTS Definitions and Data Structures

## Description

Contains the CTS specific definitions and data structures used in the CTS APIs.

## Data Structures

- struct CYBLE_CTS_CURRENT_TIME_T
- struct CYBLE_CTS_LOCAL_TIME_INFO_T
- struct CYBLE_CTS_REFERENCE_TIME_INFO_T
- struct CYBLE_CTS_CHAR_VALUE_T
- struct CYBLE_CTS_DESCR_VALUE_T
- struct CYBLE_CTSS_T

- struct CYBLE_CTSC_T

## Enumerations

- enum CYBLE_CTS_CHAR_INDEX_T
- enum CYBLE_CTS_CHAR_DESCRIPTORS_T

## Data Structure Documentation

### struct CYBLE_CTS_CURRENT_TIME_T

**Data Fields**

- uint8 yearLow
- uint8 yearHigh
- uint8 month
- uint8 day
- uint8 hours
- uint8 minutes
- uint8 seconds
- uint8 dayOfWeek
- uint8 fractions256
- uint8 adjustReason

**Field Documentation**

**uint8 CYBLE_CTS_CURRENT_TIME_T::yearLow**
LSB of current year

**uint8 CYBLE_CTS_CURRENT_TIME_T::yearHigh**
MSB of current year

**uint8 CYBLE_CTS_CURRENT_TIME_T::month**
Current month

**uint8 CYBLE_CTS_CURRENT_TIME_T::day**
Current day

**uint8 CYBLE_CTS_CURRENT_TIME_T::hours**
Current time - hours

**uint8 CYBLE_CTS_CURRENT_TIME_T::minutes**
Current time - minutes

**uint8 CYBLE_CTS_CURRENT_TIME_T::seconds**
Current time - seconds

**uint8 CYBLE_CTS_CURRENT_TIME_T::dayOfWeek**
Current day of week

**uint8 CYBLE_CTS_CURRENT_TIME_T::fractions256**
The value of 1/256th of second

**uint8 CYBLE_CTS_CURRENT_TIME_T::adjustReason**
Reason of Current Time service characteristics change

### struct CYBLE_CTS_LOCAL_TIME_INFO_T

**Data Fields**

- int8 timeZone
- uint8 dst

**Field Documentation**

**int8 CYBLE_CTS_LOCAL_TIME_INFO_T::timeZone**
Current Time Zone

**uint8 CYBLE_CTS_LOCAL_TIME_INFO_T::dst**
Daylight Saving Time value

**struct CYBLE_CTS_REFERENCE_TIME_INFO_T**

**Data Fields**

- uint8 timeSource
- uint8 timeAccuracy
- uint8 daysSinceUpdate
- uint8 hoursSinseUpdate

**Field Documentation**

**uint8 CYBLE_CTS_REFERENCE_TIME_INFO_T::timeSource**
Time update source

**uint8 CYBLE_CTS_REFERENCE_TIME_INFO_T::timeAccuracy**
Time accuracy

**uint8 CYBLE_CTS_REFERENCE_TIME_INFO_T::daysSinceUpdate**
Days since last time update

**uint8 CYBLE_CTS_REFERENCE_TIME_INFO_T::hoursSinseUpdate**
Hours since last time update

**struct CYBLE_CTS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CTS_CHAR_INDEX_T charIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CTS_CHAR_VALUE_T::connHandle**
Connection handle

**CYBLE_CTS_CHAR_INDEX_T CYBLE_CTS_CHAR_VALUE_T::charIndex**
Characteristic index of Current Time Service

**CYBLE_GATT_ERR_CODE_T CYBLE_CTS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**CYBLE_GATT_VALUE_T* CYBLE_CTS_CHAR_VALUE_T::value**
Pointer to value of Current Time Service characteristic

**struct CYBLE_CTS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_CTS_CHAR_INDEX_T charIndex
- CYBLE_CTS_CHAR_DESCRIPTORS_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_CTS_DESCR_VALUE_T::connHandle**
Connection handle

**CYBLE_CTS_CHAR_INDEX_T CYBLE_CTS_DESCR_VALUE_T::charIndex**
Characteristic index of Current Time Service

**CYBLE_CTS_CHAR_DESCRIPTORS_T CYBLE_CTS_DESCR_VALUE_T::descrIndex**
Characteristic index Descriptor of Current Time Service

**CYBLE_GATT_VALUE_T* CYBLE_CTS_DESCR_VALUE_T::value**
Pointer to value of Current Time Service characteristic

**struct CYBLE_CTSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T currTimeCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T currTimeCccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T localTimeInfCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T refTimeInfCharHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSS_T::serviceHandle**
Current Time Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSS_T::currTimeCharHandle**
Current Time Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSS_T::currTimeCccdHandle**
Current Time Client Characteristic Configuration Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSS_T::localTimeInfCharHandle**
Local Time Information Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSS_T::refTimeInfCharHandle**
Reference Time Information Characteristic handle

**struct CYBLE_CTSC_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T currTimeCharacteristics [CYBLE_CTS_CHAR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T currTimeCccdHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_CTSC_T::currTimeCharacteristics[CYBLE_CTS_CHAR_COUNT]**
Structure with Characteristic handles + properties of Current Time Service

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CTSC_T::currTimeCccdHandle**
Current Time Client Characteristic Configuration handle of Current Time Service

## Enumeration Type Documentation

**enum CYBLE_CTS_CHAR_INDEX_T**
Service Characteristics indexes

**Enumerator**

***CYBLE_CTS_CURRENT_TIME*** Current Time characteristic index

***CYBLE_CTS_LOCAL_TIME_INFO*** Local Time Information characteristic index
***CYBLE_CTS_REFERENCE_TIME_INFO*** Reference Time Information characteristic index
***CYBLE_CTS_CHAR_COUNT*** Total count of Current Time Service characteristics

**enum CYBLE_CTS_CHAR_DESCRIPTORS_T**
Service Characteristic Descriptors indexes
**Enumerator**
***CYBLE_CTS_CURRENT_TIME_CCCD*** Current Time Client Characteristic configuration descriptor index
***CYBLE_CTS_COUNT*** Total count of Current Time Service characteristic descriptors

# Device Information Service (DIS)

## Description

The Device Information Service exposes manufacturer and/or vendor information about a device.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The DIS API names begin with CyBle_Dis. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- DIS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- DIS Server Functions
  *APIs unique to DIS designs configured as a GATT Server role.*
- DIS Client Functions
  *APIs unique to DIS designs configured as a GATT Client role.*
- DIS Definitions and Data Structures
  *Contains the DIS specific definitions and data structures used in the DIS APIs.*

# DIS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Dis

## Functions

- void CyBle_DisRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

**void CyBle_DisRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**
Registers a callback function for service specific attribute operations. Callback doesn't have events in server role.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for Device Information Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

# DIS Server Functions

## Description

APIs unique to DIS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Diss

## Functions

- CYBLE_API_RESULT_T CyBle_DissSetCharacteristicValue (CYBLE_DIS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_DissGetCharacteristicValue (CYBLE_DIS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_DissSetCharacteristicValue (CYBLE_DIS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service, which is identified by charIndex, to the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_DissGetCharacteristicValue (CYBLE_DIS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service, which is identified by charIndex, from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T. Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

# DIS Client Functions

## Description

APIs unique to DIS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Disc

## Functions

- CYBLE_API_RESULT_T CyBle_DiscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_DIS_CHAR_INDEX_T charIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_DiscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_DIS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server which is identified by charIndex.

The Read Response returns the characteristic value in the Attribute Value parameter. The Read Response only contains the characteristic value that is less than or equal to (MTU - 1) octets in length. If the characteristic value is greater than (MTU - 1) octets in length, a Read Long Characteristic Value procedure may be used if the rest of the characteristic value is required.

This function call can result in generation of the following events based on the response from the server device.

- CYBLE_EVT_DISC_READ_CHAR_RESPONSE
- CYBLE_EVT_GATTC_ERROR_RSP

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the DIS service-specific callback is registered (with CyBle_DisRegisterAttrCallback):

- CYBLE_EVT_DISC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_DIS_CHAR_VALUE_T.

Otherwise (if the DIS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# DIS Definitions and Data Structures

## Description

Contains the DIS specific definitions and data structures used in the DIS APIs.

## Data Structures

- struct CYBLE_DISS_T
- struct CYBLE_DISC_T
- struct CYBLE_DIS_CHAR_VALUE_T

## Enumerations

- enum CYBLE_DIS_CHAR_INDEX_T

## Data Structure Documentation

### struct CYBLE_DISS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle [CYBLE_DIS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISS_T::serviceHandle**

Device Information Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_DISS_T::charHandle[CYBLE_DIS_CHAR_COUNT]**

Device Information Service Characteristic handles

### struct CYBLE_DISC_T

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo [CYBLE_DIS_CHAR_COUNT]

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_DISC_T::charInfo[CYBLE_DIS_CHAR_COUNT]**

Characteristics handle + properties array

### struct CYBLE_DIS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_DIS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_DIS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_DIS_CHAR_INDEX_T CYBLE_DIS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_DIS_CHAR_VALUE_T::value**
Characteristic value

### Enumeration Type Documentation

**enum CYBLE_DIS_CHAR_INDEX_T**
DIS characteristic index

**Enumerator**

***CYBLE_DIS_MANUFACTURER_NAME*** Manufacturer Name String characteristic index
***CYBLE_DIS_MODEL_NUMBER*** Model Number String characteristic index
***CYBLE_DIS_SERIAL_NUMBER*** Serial Number String characteristic index
***CYBLE_DIS_HARDWARE_REV*** Hardware Revision String characteristic index
***CYBLE_DIS_FIRMWARE_REV*** Firmware Revision String characteristic index
***CYBLE_DIS_SOFTWARE_REV*** Software Revision String characteristic index
***CYBLE_DIS_SYSTEM_ID*** System ID characteristic index
***CYBLE_DIS_REG_CERT_DATA*** IEEE 11073-20601 characteristic index
***CYBLE_DIS_PNP_ID*** PnP ID characteristic index
***CYBLE_DIS_CHAR_COUNT*** Total count of DIS characteristics

## Environmental Sensing Service (ESS)

### Description

The Environmental Sensing Service exposes measurement data from an environmental sensor intended for sports and fitness applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The ESS API names begin with CyBle_Ess. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- ESS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- ESS Server Functions
  *APIs unique to ESS designs configured as a GATT Server role.*
- ESS Client Functions
  *APIs unique to ESS designs configured as a GATT Client role.*
- ESS Definitions and Data Structures
  *Contains the ESS specific definitions and data structures used in the ESS APIs.*

# ESS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Ess

## Functions

- void CyBle_EssRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_EssRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for ESS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode: Indicates the event that triggered this callback (e.g. CYBLE_EVT_ESSS_NOTIFICATION_ENABLED).<br>• eventParam: Contains the parameters corresponding to the current event. (e.g. Pointer to CYBLE_ESS_CHAR_VALUE_T structure that contains details of the characteristic for which the notification enabled event was triggered). |

# ESS Server Functions

## Description

APIs unique to ESS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Esss

## Functions

- CYBLE_API_RESULT_T CyBle_EsssSetChangeIndex (uint16 essIndex)
- CYBLE_API_RESULT_T CyBle_EsssSetCharacteristicValue (CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsssGetCharacteristicValue (CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsssSetCharacteristicDescriptor (CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsssGetCharacteristicDescriptor (CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_EsssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_EsssSetChangeIndex (uint16 *essIndex*)**

Performs write operation of two-byte pseudo-random change index to the advertisement packet. The "Service Data" field should be selected in the component customizer GUI and contain a two-byte initial change index value and in opposite case the function will always return "CYBLE_ERROR_INVALID_OPERATION".

This function must be called when CyBle_GetBleSsState() returns CYBLE_BLESS_STATE_EVENT_CLOSE state.

**Parameters:**

| | |
|---|---|
| *essIndex* | A two-byte pseudo-random change index to be written to the advertisement data. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - On NULL pointer, Data length in input parameter exceeds 31 bytes.
- CYBLE_ERROR_INVALID_OPERATION - The change index is not present in the advertisement data or its length is not equal to two bytes or ADV Event is not closed, BLESS is active or ADV is not enabled.

**CYBLE_API_RESULT_T CyBle_EsssSetCharacteristicValue (CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size (in Bytes) of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that Event is not stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_EsssGetCharacteristicValue (CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic value of the service, which is a value identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_EsssSetCharacteristicDescriptor (CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the service characteristic descriptor of type CYBLE_ESS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_EsssGetCharacteristicDescriptor (CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the service characteristic descriptor of type CYBLE_ESS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_EsssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification with a characteristic value of the Environmental Sensing Service, which is a value specified by charIndex, to the client's device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_ESSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - A notification is not enabled by the client.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_EsssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value of the Environmental Sensing Service, which is a value specified by charIndex, to the client's device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_ESSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# ESS Client Functions

## Description

APIs unique to ESS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Essc

## Functions

- CYBLE_API_RESULT_T CyBle_EsscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance)
- CYBLE_API_RESULT_T CyBle_EsscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex)
- CYBLE_API_RESULT_T CyBle_EsscSetLongCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_EsscGetLongCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_ESS_CHAR_INDEX_T charIndex, uint8 charInstance, CYBLE_ESS_DESCR_INDEX_T descrIndex, uint16 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_EsscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_ESSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_ESS_CHAR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_EsscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_ESS_CHAR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_EsscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_ESSS_DESCR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_ESSS_NOTIFICATION_ENABLED.
- CYBLE_EVT_ESSS_NOTIFICATION_DISABLED.
- CYBLE_EVT_ESSS_INDICATION_ENABLED.
- CYBLE_EVT_ESSS_INDICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional Characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_ESS_DESCR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_EsscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.

- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional Characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_ESS_DESCR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_EsscSetLongCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*, uint16 *attrSize*, uint8 **attrValue*)**

This function is used to write a long characteristic descriptor to the server, which is identified by charIndex and descrIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_ESS_DESCR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.

- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_EsscGetLongCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_ESS_CHAR_INDEX_T *charIndex*, uint8 *charInstance*, CYBLE_ESS_DESCR_INDEX_T *descrIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Sends a request to read long characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *charInstance* | The instance number of the characteristic specified by "charIndex". |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the buffer where the read long characteristic descriptor value should be stored. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The optional Characteristic Descriptor is absent.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the ESS service-specific callback is registered (with CyBle_EssRegisterAttrCallback):

- CYBLE_EVT_ESSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_ESS_DESCR_VALUE_T.

Otherwise (if the ESS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_BLOB_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# ESS Definitions and Data Structures

## Description

Contains the ESS specific definitions and data structures used in the ESS APIs.

## Data Structures

- struct CYBLE_ESSS_CHAR_T
- struct CYBLE_ESSS_CHAR_INFO_PTR_T

- struct CYBLE_ESSS_T
- struct CYBLE_ESSC_CHAR_T
- struct CYBLE_ESSC_CHAR_INFO_PTR_T
- struct CYBLE_ESSC_T
- struct CYBLE_ESS_CHAR_VALUE_T
- struct CYBLE_ESS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_ESS_CHAR_INDEX_T
- enum CYBLE_ESS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_ESSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_ESS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSS_CHAR_T::charHandle**

Handles of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSS_CHAR_T::descrHandle[CYBLE_ESS_DESCR_COUNT]**

Array of Descriptor handles

### struct CYBLE_ESSS_CHAR_INFO_PTR_T

**Data Fields**

- CYBLE_ESSS_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_ESSS_CHAR_T* CYBLE_ESSS_CHAR_INFO_PTR_T::charInfoPtr**

Pointer to CYBLE_ESSS_CHAR_T which holds information about specific ES Characteristic

### struct CYBLE_ESSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_ESSS_CHAR_INFO_PTR_T charInfoAddr [CYBLE_ESS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSS_T::serviceHandle**

Environmental Sensing Service handle

**CYBLE_ESSS_CHAR_INFO_PTR_T CYBLE_ESSS_T::charInfoAddr[CYBLE_ESS_CHAR_COUNT]**

Environmental Sensing Service Array with pointers to Characteristic handles.

### struct CYBLE_ESSC_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_ESS_DESCR_COUNT]
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSC_CHAR_T::valueHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSC_CHAR_T::endHandle**
End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSC_CHAR_T::descrHandle[CYBLE_ESS_DESCR_COUNT]**
Array of Descriptor handles

**uint8 CYBLE_ESSC_CHAR_T::properties**
Properties for value field

**struct CYBLE_ESSC_CHAR_INFO_PTR_T**

**Data Fields**

- CYBLE_ESSC_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_ESSC_CHAR_T* CYBLE_ESSC_CHAR_INFO_PTR_T::charInfoPtr**
Pointer to CYBLE_ESSC_CHAR_T which holds information about specific ES Characteristic.

**struct CYBLE_ESSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_ESSC_CHAR_INFO_PTR_T charInfoAddr [CYBLE_ESS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_ESSC_T::serviceHandle**
Environmental Sensing Service handle

**CYBLE_ESSC_CHAR_INFO_PTR_T CYBLE_ESSC_T::charInfoAddr[CYBLE_ESS_CHAR_COUNT]**
Environmental Sensing Service Array with pointers to characteristic information.

**struct CYBLE_ESS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ESS_CHAR_INDEX_T charIndex
- uint8 charInstance
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ESS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_ESS_CHAR_INDEX_T CYBLE_ESS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**uint8 CYBLE_ESS_CHAR_VALUE_T::charInstance**
Instance of specific service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_ESS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_ESS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_ESS_CHAR_INDEX_T charIndex
- uint8 charInstance
- CYBLE_ESS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_ESS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_ESS_CHAR_INDEX_T CYBLE_ESS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**uint8 CYBLE_ESS_DESCR_VALUE_T::charInstance**
Instance of specific service characteristic

**CYBLE_ESS_DESCR_INDEX_T CYBLE_ESS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_ERR_CODE_T CYBLE_ESS_DESCR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_ESS_DESCR_VALUE_T::value**
Characteristic value

## Enumeration Type Documentation

**enum CYBLE_ESS_CHAR_INDEX_T**
ESS Characteristic indexes

**Enumerator**

***CYBLE_ESS_DESCRIPTOR_VALUE_CHANGED*** Descriptor Value Changed Characteristic index
***CYBLE_ESS_APPARENT_WIND_DIR*** Apparent Wind Direction Characteristic index
***CYBLE_ESS_APPARENT_WIND_SPEED*** Apparent Wind Speed Characteristic index
***CYBLE_ESS_DEW_POINT*** Dew Point Characteristic index
***CYBLE_ESS_ELEVATION*** Elevation Characteristic index
***CYBLE_ESS_GUST_FACTOR*** Gust Factor Characteristic index
***CYBLE_ESS_HEAT_INDEX*** Heat Index Characteristic index
***CYBLE_ESS_HUMIDITY*** Humidity Characteristic index
***CYBLE_ESS_IRRADIANCE*** Irradiance Characteristic index
***CYBLE_ESS_POLLEN_CONCENTRATION*** Pollen Concentration Characteristic index
***CYBLE_ESS_RAINFALL*** Rainfall Characteristic index
***CYBLE_ESS_PRESSURE*** Pressure Characteristic index
***CYBLE_ESS_TEMPERATURE*** Temperature Characteristic index
***CYBLE_ESS_TRUE_WIND_DIR*** True Wind Direction Characteristic index
***CYBLE_ESS_TRUE_WIND_SPEED*** True Wind Speed Characteristic index
***CYBLE_ESS_UV_INDEX*** UV Index Characteristic index
***CYBLE_ESS_WIND_CHILL*** Wind Chill Characteristic index

***CYBLE_ESS_BAROMETRIC_PRESSURE_TREND*** Barometric Pressure trend Characteristic index
***CYBLE_ESS_MAGNETIC_DECLINATION*** Magnetic Declination Characteristic index
***CYBLE_ESS_MAGNETIC_FLUX_DENSITY_2D*** Magnetic Flux Density 2D Characteristic index
***CYBLE_ESS_MAGNETIC_FLUX_DENSITY_3D*** Magnetic Flux Density 3D Characteristic index
***CYBLE_ESS_CHAR_COUNT*** Total count of ESS characteristics

**enum CYBLE_ESS_DESCR_INDEX_T**

ESS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_ESS_CCCD*** Client Characteristic Configuration Descriptor index
***CYBLE_ESS_CHAR_EXTENDED_PROPERTIES*** Characteristic Extended Properties Descriptor index
***CYBLE_ESS_ES_MEASUREMENT_DESCR*** ES Measurement Descriptor index
***CYBLE_ESS_ES_TRIGGER_SETTINGS_DESCR1*** ES Trigger Settings Descriptor #1 index
***CYBLE_ESS_ES_TRIGGER_SETTINGS_DESCR2*** ES Trigger Settings Descriptor #2 index
***CYBLE_ESS_ES_TRIGGER_SETTINGS_DESCR3*** ES Trigger Settings Descriptor #3 index
***CYBLE_ESS_ES_CONFIG_DESCR*** ES Configuration Descriptor index
***CYBLE_ESS_CHAR_USER_DESCRIPTION_DESCR*** Characteristic User Description Descriptor index
***CYBLE_ESS_VRD*** Valid Range Descriptor index
***CYBLE_ESS_DESCR_COUNT*** Total count of descriptors

# Glucose Service (GLS)

## Description

The Glucose Service exposes glucose and other data related to a personal glucose sensor for consumer healthcare applications and is not designed for clinical use.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The GLS API names begin with CyBle_Gls. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- GLS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- GLS Server Functions
  *APIs unique to GLS designs configured as a GATT Server role.*
- GLS Client Functions
  *APIs unique to GLS designs configured as a GATT Client role.*
- GLS Definitions and Data Structures
  *Contains the GLS specific definitions and data structures used in the GLS APIs.*

# GLS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Gls

## Functions

- void CyBle_GlsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_GlsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for Glucose Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# GLS Server Functions

## Description

APIs unique to GLS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Glss

## Functions

- CYBLE_API_RESULT_T CyBle_GlssSetCharacteristicValue (CYBLE_GLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlssGetCharacteristicValue (CYBLE_GLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlssGetCharacteristicDescriptor (CYBLE_GLS_CHAR_INDEX_T charIndex, CYBLE_GLS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GlssSetCharacteristicValue (CYBLE_GLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_GlssGetCharacteristicValue (CYBLE_GLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | Pointer to the location where Characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_GlssGetCharacteristicDescriptor (CYBLE_GLS_CHAR_INDEX_T *charIndex*, CYBLE_GLS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| **attrValue* | Pointer to the location where the descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_GlssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic to the client device, as defined by the charIndex value.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_GLSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | Pointer to the Characteristic value data that should be sent to Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

**CYBLE_API_RESULT_T CyBle_GlssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a indication of the specified characteristic to the client device, as defined by the charIndex value.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_GLSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | Pointer to the Characteristic value data that should be sent to Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the GLS service-specific callback is registered (with CyBle_GlsRegisterAttrCallback):

- CYBLE_EVT_GLSS_INDICATION_CONFIRMED - In case if the indication is successfully delivered to the peer device.

Otherwise (if the GLS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - In case if the indication is successfully delivered to the peer device.

# GLS Client Functions

## Description

APIs unique to GLS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Glsc

## Functions

- CYBLE_API_RESULT_T CyBle_GlscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_GlscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex, CYBLE_GLS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_GlscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_GLS_CHAR_INDEX_T charIndex, CYBLE_GLS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_GlscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_GLSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| **attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the GLS service-specific callback is registered (with CyBle_GlsRegisterAttrCallback):

- CYBLE_EVT_GLSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_GLS_CHAR_VALUE_T.

Otherwise (if the GLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_GlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the GLS service-specific callback is registered (with CyBle_GlsRegisterAttrCallback):

- CYBLE_EVT_GLSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_GLS_CHAR_VALUE_T.

Otherwise (if the GLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_GlscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*, CYBLE_GLS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the Characteristic Descriptor of the specified Characteristic.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_GLSS_INDICATION_ENABLED.
- CYBLE_EVT_GLSS_INDICATION_DISABLED.
- CYBLE_EVT_GLSS_NOTIFICATION_ENABLED.
- CYBLE_EVT_GLSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |

| | |
|---|---|
| *attrSize* | The size of the characteristic descriptor value attribute. |
| **attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the GLS service-specific callback is registered (with CyBle_GlsRegisterAttrCallback):

- CYBLE_EVT_GLSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_GLS_DESCR_VALUE_T.

Otherwise (if the GLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_GlscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_GLS_CHAR_INDEX_T *charIndex*, CYBLE_GLS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the GLS service-specific callback is registered (with CyBle_GlsRegisterAttrCallback):

- CYBLE_EVT_GLSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_GLS_DESCR_VALUE_T.

Otherwise (if the GLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# GLS Definitions and Data Structures

## Description

Contains the GLS specific definitions and data structures used in the GLS APIs.

## Data Structures

- struct CYBLE_GLSS_CHAR_T
- struct CYBLE_GLSS_T
- struct CYBLE_GLSC_CHAR_T
- struct CYBLE_GLSC_T
- struct CYBLE_GLS_CHAR_VALUE_T
- struct CYBLE_GLS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_GLS_CHAR_INDEX_T
- enum CYBLE_GLS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_GLSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSS_CHAR_T::charHandle**

Glucose Service char handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSS_CHAR_T::cccdHandle**

Glucose Service CCCD handle

### struct CYBLE_GLSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GLSS_CHAR_T charInfo [CYBLE_GLS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSS_T::serviceHandle**

Glucose Service handle

**CYBLE_GLSS_CHAR_T CYBLE_GLSS_T::charInfo[CYBLE_GLS_CHAR_COUNT]**
Glucose Service characteristics info array

**struct CYBLE_GLSC_CHAR_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_GLSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSC_CHAR_T::valueHandle**
Handle of server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSC_CHAR_T::cccdHandle**
Glucose client char. descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_GLSC_CHAR_T::endHandle**
Characteristic End Handle

**struct CYBLE_GLSC_T**

**Data Fields**

- CYBLE_GLSC_CHAR_T charInfo [CYBLE_GLS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GLSC_CHAR_T CYBLE_GLSC_T::charInfo[CYBLE_GLS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_GLS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GLS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GLS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_GLS_CHAR_INDEX_T CYBLE_GLS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_GLS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_GLS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_GLS_CHAR_INDEX_T charIndex
- CYBLE_GLS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_GLS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_GLS_CHAR_INDEX_T CYBLE_GLS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GLS_DESCR_INDEX_T CYBLE_GLS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_GLS_DESCR_VALUE_T::value**
Descriptor value

### Enumeration Type Documentation

**enum CYBLE_GLS_CHAR_INDEX_T**
Service Characteristics indexes
**Enumerator**
***CYBLE_GLS_GLMT*** Glucose Measurement characteristic index
***CYBLE_GLS_GLMC*** Glucose Measurement Context characteristic index
***CYBLE_GLS_GLFT*** Glucose Feature characteristic index
***CYBLE_GLS_RACP*** Record Access Control Point characteristic index
***CYBLE_GLS_CHAR_COUNT*** Total count of GLS characteristics

**enum CYBLE_GLS_DESCR_INDEX_T**
Service Characteristic Descriptors indexes
**Enumerator**
***CYBLE_GLS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_GLS_DESCR_COUNT*** Total count of GLS descriptors

## HID Service (HIDS)

### Description

The HID Service exposes data and associated formatting for HID Devices and HID Hosts.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The HID API names begin with CyBle_Hid. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- HIDS Server and Client Functions
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- HIDS Server Functions
  *APIs unique to HID designs configured as a GATT Server role.*
- HIDS Client Functions
  *APIs unique to HID designs configured as a GATT Client role.*
- HIDS Definitions and Data Structures

*Contains the HID specific definitions and data structures used in the HID APIs.*

# HIDS Server and Client Functions

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Hids

## Functions

- void CyBle_HidsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_HidsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for HID Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_HIDS_NOTIFICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_HIDS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# HIDS Server Functions

## Description

APIs unique to HID designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Hidss

## Functions

- CYBLE_API_RESULT_T CyBle_HidssSetCharacteristicValue (uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HidssGetCharacteristicValue (uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_HidssGetCharacteristicDescriptor (uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, CYBLE_HIDS_DESCR_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HidssSendNotification (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HidssSetCharacteristicValue (uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets local characteristic value of the specified HID Service characteristics.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic.<br>• CYBLE_HIDS_PROTOCOL_MODE - Protocol Mode characteristic<br>• CYBLE_HIDS_REPORT_MAP - Report Map characteristic<br>• CYBLE_HIDS_INFORMATION - HID Information characteristic<br>• CYBLE_HIDS_CONTROL_POINT - HID Control Point characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_IN_REP - Boot Keyboard Input Report Characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_OUT_REP - Boot Keyboard Output Report Characteristic<br>• CYBLE_HIDS_BOOT_MOUSE_IN_REP - Boot Mouse Input Report Characteristic<br>• CYBLE_HIDS_REPORT - Report Characteristic |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_HidssGetCharacteristicValue (uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets local characteristic value of the specified HID Service characteristics.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of the service characteristic.<br>• CYBLE_HIDS_PROTOCOL_MODE - Protocol Mode characteristic<br>• CYBLE_HIDS_REPORT_MAP - Report Map characteristic<br>• CYBLE_HIDS_INFORMATION - HID Information characteristic |

| | |
|---|---|
| | • CYBLE_HIDS_CONTROL_POINT - HID Control Point characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_IN_REP - Boot Keyboard Input Report Characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_OUT_REP - Boot Keyboard Output Report Characteristic<br>• CYBLE_HIDS_BOOT_MOUSE_IN_REP - Boot Mouse Input Report Characteristic<br>• CYBLE_HIDS_REPORT - Report Characteristic |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_HidssGetCharacteristicDescriptor (uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, CYBLE_HIDS_DESCR_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets local characteristic descriptor of the specified HID Service characteristic.

**Parameters:**

| | |
|---|---|
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of the characteristic.<br>• CYBLE_HIDS_REPORT_MAP - Report Map Characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_IN_REP - Boot Keyboard Input Report Characteristic<br>• CYBLE_HIDS_BOOT_KYBRD_OUT_REP - Boot Keyboard Output Report Characteristic<br>• CYBLE_HIDS_BOOT_MOUSE_IN_REP - Boot Mouse Input Report Characteristic<br>• CYBLE_HIDS_REPORT - Report Characteristic |
| *descrIndex* | The index of the descriptor.<br>• CYBLE_HIDS_REPORT_CCCD - Client Characteristic Configuration descriptor<br>• CYBLE_HIDS_REPORT_RRD - Report Reference descriptor<br>• CYBLE_HIDS_REPORT_MAP_ERRD - Report Map External Report Reference descriptor |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent

**CYBLE_API_RESULT_T CyBle_HidssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends specified HID Service characteristic notification to the Client device.

CYBLE_EVT_HIDSC_NOTIFICATION event is received by the peer device, on invoking this function.

On enabling notification successfully for a service characteristic, it sends out a 'Handle Value Notification' which results in CYBLE_EVT_HIDSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | BLE peer device connection handle. |
| *serviceIndex* | The index of the HID service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

# HIDS Client Functions

## Description

APIs unique to HID designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Hidsc

## Functions

- CYBLE_API_RESULT_T CyBle_HidscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HIDSC_CHAR_WRITE_T subProcedure, uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HidscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HIDSC_CHAR_READ_T subProcedure, uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_HidscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, CYBLE_HIDS_DESCR_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HidscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, uint8 serviceIndex, CYBLE_HIDS_CHAR_INDEX_T charIndex, CYBLE_HIDS_DESCR_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HidscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HIDSC_CHAR_WRITE_T *subProcedure*, uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a request to set characteristic value of the specified HID Service, which is identified by serviceIndex and reportIndex, on the server device. This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_HIDSC_WRITE_CHAR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *subProcedure* | Characteristic value write sub-procedure.<br>• CYBLE_HIDSC_WRITE_WITHOUT_RESPONSE;<br>• CYBLE_HIDSC_WRITE_CHAR_VALUE. |
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HIDS service-specific callback is registered (with CyBle_HidsRegisterAttrCallback):

- CYBLE_EVT_HIDSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HIDS_CHAR_VALUE_T.

Otherwise (if the HIDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HidscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HIDSC_CHAR_READ_T *subProcedure*, uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic value from a server which is identified by charIndex.

The Read Response returns the characteristic value in the Attribute Value parameter.

The Read Response only contains the characteristic value that is less than or equal to (MTU - 1) octets in length. If the characteristic value is greater than (MTU - 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the characteristic Value is required.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *subProcedure* | The characteristic value read sub-procedure.<br>• CYBLE_HIDSC_READ_CHAR_VALUE;<br>• CYBLE_HIDSC_READ_LONG_CHAR_VALUE. |
| *serviceIndex* | The index of the service instance. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HIDS service-specific callback is registered (with CyBle_HidsRegisterAttrCallback):

- CYBLE_EVT_HIDSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_HIDS_CHAR_VALUE_T.

Otherwise (if the HIDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_READ_BLOB_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HidscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, CYBLE_HIDS_DESCR_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic descriptor to the server, which is identified by charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_HIDSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_HIDSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The BLE peer device connection handle. |
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |

| | |
|---|---|
| *charIndex* | The index of the HID service characteristic. |
| *descrIndex* | The index of the HID service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HIDS service-specific callback is registered (with CyBle_HidsRegisterAttrCallback):

- CYBLE_EVT_HIDSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_HIDS_DESCR_VALUE_T.

Otherwise (if the HIDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HidscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, uint8 *serviceIndex*, CYBLE_HIDS_CHAR_INDEX_T *charIndex*, CYBLE_HIDS_DESCR_T *descrIndex*)**

Gets a characteristic descriptor of the specified characteristic of the HID Service from the server device.

This function call can result in generation of the following events based on the response from the server device.

- CYBLE_EVT_HIDSC_READ_DESCR_RESPONSE;
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *serviceIndex* | The index of the service instance. e.g. If two HID Services are supported in your design, then first service will be identified by serviceIndex of 0 and the second by serviceIndex of 1. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the HID Service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HIDS service-specific callback is registered (with CyBle_HidsRegisterAttrCallback):

- CYBLE_EVT_HIDSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_HIDS_DESCR_VALUE_T.

Otherwise (if the HIDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# HIDS Definitions and Data Structures

## Description

Contains the HID specific definitions and data structures used in the HID APIs.

## Data Structures

- struct CYBLE_HIDSS_REPORT_REF_T
- struct CYBLE_HIDSS_INFORMATION_T
- struct CYBLE_HIDSS_REPORT_T
- struct CYBLE_HIDSS_T
- struct CYBLE_HIDSC_REPORT_T
- struct CYBLE_HIDSC_REPORT_MAP_T
- struct CYBLE_HIDSC_T
- struct CYBLE_HIDS_CHAR_VALUE_T
- struct CYBLE_HIDS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_HIDS_CHAR_INDEX_T
- enum CYBLE_HIDS_DESCR_T
- enum CYBLE_HIDSC_CHAR_WRITE_T
- enum CYBLE_HIDSC_CHAR_READ_T

## Data Structure Documentation

### struct CYBLE_HIDSS_REPORT_REF_T

**Data Fields**

- uint8 reportId
- uint8 reportType

**Field Documentation**

**uint8 CYBLE_HIDSS_REPORT_REF_T::reportId**

Non-zero value if there are more than one instance of the same Report Type

**uint8 CYBLE_HIDSS_REPORT_REF_T::reportType**
Type of Report characteristic

**struct CYBLE_HIDSS_INFORMATION_T**

**Data Fields**
- uint16 bcdHID
- uint8 bCountryCode
- uint8 flags

**Field Documentation**

**uint16 CYBLE_HIDSS_INFORMATION_T::bcdHID**
Version number of HIDSe USB HID Specification implemented by HID Device

**uint8 CYBLE_HIDSS_INFORMATION_T::bCountryCode**
Identifies which country hardware is localized for

**uint8 CYBLE_HIDSS_INFORMATION_T::flags**
Bit 0: RemoteWake - Indicates whether HID Device is capable of sending wake-signal to HID Host. Bit 1: NormallyConnectable - Indicates whether HID Device will be advertising when bonded but not connected.

**struct CYBLE_HIDSS_REPORT_T**

**Data Fields**
- CYBLE_GATT_DB_ATTR_HANDLE_T reportHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T rrdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_REPORT_T::reportHandle**
Handle of Report characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_REPORT_T::cccdHandle**
Handle of Client Characteristic Configuration descriptor

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_REPORT_T::rrdHandle**
Handle of Report Reference descriptor

**struct CYBLE_HIDSS_T**

**Data Fields**
- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T protocolModeHandle
- uint8 reportCount
- const CYBLE_HIDSS_REPORT_T * reportArray
- CYBLE_HIDSS_REPORT_T bootReportArray [(0x03u)]
- CYBLE_GATT_DB_ATTR_HANDLE_T reportMapHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T reportMapErrdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T informationHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T controlPointHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::serviceHandle**
Handle of HID service

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::protocolModeHandle**
Handle of Protocol Mode Characteristic

**uint8 CYBLE_HIDSS_T::reportCount**
Number of report Characteristics

**const CYBLE_HIDSS_REPORT_T* CYBLE_HIDSS_T::reportArray**
Info about report Characteristics

**CYBLE_HIDSS_REPORT_T CYBLE_HIDSS_T::bootReportArray[(0x03u)]**
Info about Boot Report Characteristics

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::reportMapHandle**
Handle of Report Map Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::reportMapErrdHandle**
Handle of Report Map External Report Reference descr.

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::informationHandle**
Handle of HID Information Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSS_T::controlPointHandle**
Handle of HID Control Point Characteristic

### struct CYBLE_HIDSC_REPORT_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T cccdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T rrdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_T::cccdHandle**
Handle of Client Characteristic Configuration Descriptor

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_T::rrdHandle**
Handle of Report Reference Descriptor

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_T::valueHandle**
Handle of Report Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_T::endHandle**
End handle of Characteristic

**uint8 CYBLE_HIDSC_REPORT_T::properties**
Properties for value field

### struct CYBLE_HIDSC_REPORT_MAP_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T errdHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_MAP_T::errdHandle**
Handle of Report Map External Report Reference descriptor

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_MAP_T::valueHandle**
Handle of Report characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_REPORT_MAP_T::endHandle**
End handle of characteristic

**uint8 CYBLE_HIDSC_REPORT_MAP_T::properties**
Properties for value field

**struct CYBLE_HIDSC_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_SRVR_CHAR_INFO_T protocolMode
- CYBLE_HIDSC_REPORT_T bootReport [(0x03u)]
- CYBLE_HIDSC_REPORT_MAP_T reportMap
- CYBLE_SRVR_CHAR_INFO_T information
- CYBLE_SRVR_CHAR_INFO_T controlPoint
- CYBLE_HIDSC_REPORT_T report [(`$HidsCReportCount`)]
- uint8 reportCount
- CYBLE_GATT_DB_ATTR_HANDLE_T includeHandle

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HIDSC_T::connHandle**
Peer device handle

**CYBLE_SRVR_CHAR_INFO_T CYBLE_HIDSC_T::protocolMode**
Protocol Mode Characteristic handle and properties

**CYBLE_HIDSC_REPORT_T CYBLE_HIDSC_T::bootReport[(0x03u)]**
Boot Report Characteristic info

**CYBLE_HIDSC_REPORT_MAP_T CYBLE_HIDSC_T::reportMap**
Report Map Characteristic handle and descriptors

**CYBLE_SRVR_CHAR_INFO_T CYBLE_HIDSC_T::information**
Information Characteristic handle and properties

**CYBLE_SRVR_CHAR_INFO_T CYBLE_HIDSC_T::controlPoint**
Control Point Characteristic handle and properties

**CYBLE_HIDSC_REPORT_T CYBLE_HIDSC_T::report[(`$HidsCReportCount`)]**
Report Characteristic info

**uint8 CYBLE_HIDSC_T::reportCount**
Number of report Characteristics

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HIDSC_T::includeHandle**
Included declaration handle

**struct CYBLE_HIDS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint8 serviceIndex
- CYBLE_HIDS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HIDS_CHAR_VALUE_T::connHandle**
Peer device handle

**uint8 CYBLE_HIDS_CHAR_VALUE_T::serviceIndex**
Index of HID Service

**CYBLE_HIDS_CHAR_INDEX_T CYBLE_HIDS_CHAR_VALUE_T::charIndex**
Index of HID Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_HIDS_CHAR_VALUE_T::value**
Pointer to Characteristic value

**struct CYBLE_HIDS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- uint8 serviceIndex
- CYBLE_HIDS_CHAR_INDEX_T charIndex
- CYBLE_HIDS_DESCR_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HIDS_DESCR_VALUE_T::connHandle**
Peer device handle

**uint8 CYBLE_HIDS_DESCR_VALUE_T::serviceIndex**
Index of HID Service

**CYBLE_HIDS_CHAR_INDEX_T CYBLE_HIDS_DESCR_VALUE_T::charIndex**
Index of HID Service Characteristic

**CYBLE_HIDS_DESCR_T CYBLE_HIDS_DESCR_VALUE_T::descrIndex**
Service Characteristic Descriptor index

**CYBLE_GATT_VALUE_T* CYBLE_HIDS_DESCR_VALUE_T::value**
Pointer to value of Service Characteristic Descriptor value

## Enumeration Type Documentation

**enum CYBLE_HIDS_CHAR_INDEX_T**
HIDS characteristic indexes

**Enumerator**

***CYBLE_HIDS_PROTOCOL_MODE*** Protocol Mode Characteristic index
***CYBLE_HIDS_INFORMATION*** HID Information Characteristic index
***CYBLE_HIDS_CONTROL_POINT*** HID Control Point Characteristic index
***CYBLE_HIDS_REPORT_MAP*** Report Map Characteristic index
***CYBLE_HIDS_BOOT_KYBRD_IN_REP*** Boot Keyboard Input Report Characteristic index
***CYBLE_HIDS_BOOT_KYBRD_OUT_REP*** Boot Keyboard Output Report Characteristic index
***CYBLE_HIDS_BOOT_MOUSE_IN_REP*** Boot Mouse Input Report Characteristic index
***CYBLE_HIDS_REPORT*** Report Characteristic index
***CYBLE_HIDS_REPORT_END*** Index of last Report Char
***CYBLE_HIDS_CHAR_COUNT*** Total count of characteristics

**enum CYBLE_HIDS_DESCR_T**
HID Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_HIDS_REPORT_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_HIDS_REPORT_RRD*** Report Reference descriptor index
***CYBLE_HIDS_REPORT_MAP_ERRD*** Report Map External Report Reference descriptor index
***CYBLE_HIDS_DESCR_COUNT*** Total count of descriptors

**enum CYBLE_HIDSC_CHAR_WRITE_T**

Characteristic Value Write Sub-Procedure supported by HID Service

**Enumerator**

***CYBLE_HIDSC_WRITE_WITHOUT_RESPONSE*** Write Without Response
***CYBLE_HIDSC_WRITE_CHAR_VALUE*** Write Characteristic Value

**enum CYBLE_HIDSC_CHAR_READ_T**

Characteristic Value Read Sub-Procedure supported by HID Service

**Enumerator**

***CYBLE_HIDSC_READ_CHAR_VALUE*** Read Characteristic Value
***CYBLE_HIDSC_READ_LONG_CHAR_VALUE*** Read Long Characteristic Values

# Heart Rate Service (HRS)

## Description

The Heart Rate Service exposes heart rate and other data related to a heart rate sensor intended for fitness applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The HRS API names begin with CyBle_Hrs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- HRS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- HRS Server Functions
  *APIs unique to HRS designs configured as a GATT Server role.*
- HRS Client Functions
  *APIs unique to HRS designs configured as a GATT Client role.*
- HRS Definitions and Data Structures
  *Contains the HRS specific definitions and data structures used in the HRS APIs.*

# HRS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Hrs

### Functions

- void CyBle_HrsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_HrsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for HRS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_HRSS_NOTIFICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_HRS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

## HRS Server Functions

### Description

APIs unique to HRS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Hrss

### Functions

- CYBLE_API_RESULT_T CyBle_HrssSetCharacteristicValue (CYBLE_HRS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HrssGetCharacteristicValue (CYBLE_HRS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HrssGetCharacteristicDescriptor (CYBLE_HRS_CHAR_INDEX_T charIndex, CYBLE_HRS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HrssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HRS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_HrssSetCharacteristicValue (CYBLE_HRS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets local characteristic value of the specified Heart Rate Service characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_HrssGetCharacteristicValue (CYBLE_HRS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the local characteristic value of specified Heart Rate Service characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_HrssGetCharacteristicDescriptor (CYBLE_HRS_CHAR_INDEX_T *charIndex*, CYBLE_HRS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the local characteristic descriptor of the specified Heart Rate Service characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. The Heart Rate Measurement characteristic client configuration descriptor has 2 bytes length. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_HrssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HRS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends notification of a specified Heart Rate Service characteristic value to the Client device. No response is expected.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_HRSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of a service characteristic. |

| | |
|---|---|
| *attrSize* | The size of the characteristic value attribute. The Heart Rate Measurement characteristic has 2 bytes length (by default). The Body Sensor Location and Control Point characteristic both have 1 byte length. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# HRS Client Functions

## Description

APIs unique to HRS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Hrsc

## Functions

- CYBLE_API_RESULT_T CyBle_HrscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HRS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HrscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HRS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_HrscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HRS_CHAR_INDEX_T charIndex, CYBLE_HRS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HrscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HRS_CHAR_INDEX_T charIndex, CYBLE_HRS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HrscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HRS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic value attribute (identified by charIndex) to the server. The Write Response just confirms the operation success.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_HRSC_WRITE_CHAR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HRS service-specific callback is registered (with CyBle_HrsRegisterAttrCallback):

- CYBLE_EVT_HRSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HRS_CHAR_VALUE_T.

Otherwise (if the HRS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_HrscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HRS_CHAR_INDEX_T *charIndex*)

This function is used to read the characteristic Value from a server which is identified by charIndex.

The Read Response returns the characteristic Value in the Attribute Value parameter.

The Read Response only contains the characteristic Value that is less than or equal to (MTU - 1) octets in length. If the characteristic Value is greater than (MTU - 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the characteristic Value is required.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HRS service-specific callback is registered (with CyBle_HrsRegisterAttrCallback):

- CYBLE_EVT_HRSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_HRS_CHAR_VALUE_T.

Otherwise (if the HRS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HrscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HRS_CHAR_INDEX_T *charIndex*, CYBLE_HRS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server, which is identified by charIndex.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_HRSC_WRITE_DESCR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

One of the following events is received by the peer device, on invoking this function:

- CYBLE_EVT_HRSS_NOTIFICATION_ENABLED.
- CYBLE_EVT_HRSS_NOTIFICATION_DISABLED.
- CYBLE_EVT_HRSS_ENERGY_EXPENDED_RESET.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HRS service-specific callback is registered (with CyBle_HrsRegisterAttrCallback):

- CYBLE_EVT_HRSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_HRS_DESCR_VALUE_T.

Otherwise (if the HRS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HrscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HRS_CHAR_INDEX_T *charIndex*, CYBLE_HRS_DESCR_INDEX_T *descrIndex*)**

Gets a characteristic descriptor of a specified characteristic of the service.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_HRSC_READ_DESCR_RESPONSE
- CYBLE_EVT_GATTC_ERROR_RSP

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HRS service-specific callback is registered (with CyBle_HrsRegisterAttrCallback):

- CYBLE_EVT_HRSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_HRS_DESCR_VALUE_T.

Otherwise (if the HRS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# HRS Definitions and Data Structures

## Description

Contains the HRS specific definitions and data structures used in the HRS APIs.

## Data Structures

- struct CYBLE_HRSS_T
- struct CYBLE_HRSC_T
- struct CYBLE_HRS_CHAR_VALUE_T
- struct CYBLE_HRS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_HRS_CHAR_INDEX_T
- enum CYBLE_HRS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_HRSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle [CYBLE_HRS_CHAR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T hrmCccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HRSS_T::serviceHandle**
Heart Rate Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HRSS_T::charHandle[CYBLE_HRS_CHAR_COUNT]**
Heart Rate Service characteristics handles and properties array

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HRSS_T::hrmCccdHandle**
Heart Rate Measurement client char. config. descriptor Handle

### struct CYBLE_HRSC_T

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo [CYBLE_HRS_CHAR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T hrmCccdHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_HRSC_T::charInfo[CYBLE_HRS_CHAR_COUNT]**
Heart Rate Service characteristics handles and properties array

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HRSC_T::hrmCccdHandle**
Heart Rate Measurement client char. config. descriptor Handle

### struct CYBLE_HRS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_HRS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HRS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_HRS_CHAR_INDEX_T CYBLE_HRS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_HRS_CHAR_VALUE_T::value**
Characteristic value

### struct CYBLE_HRS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle

- CYBLE_HRS_CHAR_INDEX_T charIndex
- CYBLE_HRS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HRS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_HRS_CHAR_INDEX_T CYBLE_HRS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_HRS_DESCR_INDEX_T CYBLE_HRS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_HRS_DESCR_VALUE_T::value**
Descriptor value

### Enumeration Type Documentation

**enum CYBLE_HRS_CHAR_INDEX_T**
HRS Characteristics indexes
**Enumerator**
***CYBLE_HRS_HRM*** Heart Rate Measurement characteristic index
***CYBLE_HRS_BSL*** Body Sensor Location characteristic index
***CYBLE_HRS_CPT*** Control Point characteristic index
***CYBLE_HRS_CHAR_COUNT*** Total count of HRS characteristics

**enum CYBLE_HRS_DESCR_INDEX_T**
HRS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_HRS_HRM_CCCD*** Heart Rate Measurement client char. config. descriptor index
***CYBLE_HRS_DESCR_COUNT*** Total count of HRS HRM descriptors

## HTTP Proxy Service (HPS)

### Description

The HTTP Proxy Service allows a Client device, typically a sensor, to communicate with a Web Server through a gateway device.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The HPS API names begin with CyBle_Hps. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- HPS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- HPS Server Functions
  *APIs unique to HPS designs configured as a GATT Server role.*
- HPS Client Functions

*APIs unique to HPS designs configured as a GATT Client role.*

- HPS Definitions and Data Structures
  *Contains the HPS specific definitions and data structures used in the HPS APIs.*

# HPS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Hps

## Functions

- void CyBle_HpsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_HpsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br><br>• eventCode - Indicates the event that triggered this callback (e.g. CYBLE_EVT_HPSS_NOTIFICATION_ENABLED).<br>• eventParam - Contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_HPS_CHAR_VALUE_T structure that contains details of the characteristic for which an indication enabled event was triggered). |

# HPS Server Functions

## Description

APIs unique to HPS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Hpss

## Functions

- CYBLE_API_RESULT_T CyBle_HpssSetCharacteristicValue (CYBLE_HPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpssGetCharacteristicValue (CYBLE_HPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpssSetCharacteristicDescriptor (CYBLE_HPS_CHAR_INDEX_T charIndex, CYBLE_HPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_HpssGetCharacteristicDescriptor (CYBLE_HPS_CHAR_INDEX_T charIndex, CYBLE_HPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HpssSetCharacteristicValue (CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Sets a value for one of characteristic values of the HTTP Proxy Service. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a HTTP Proxy Service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was written successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_HpssGetCharacteristicValue (CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Reads a characteristic value of the HTTP Proxy Service, which is identified by charIndex from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the HTTP Proxy Service characteristic. |
| *attrSize* | The size of the HTTP Proxy Service characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_HpssSetCharacteristicDescriptor (CYBLE_HPS_CHAR_INDEX_T *charIndex*, CYBLE_HPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor value of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HpssGetCharacteristicDescriptor (CYBLE_HPS_CHAR_INDEX_T *charIndex*, CYBLE_HPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a a characteristic descriptor of a specified characteristic of the HTTP Proxy Service from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HpssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification with a characteristic value of the HTTP Proxy Service, which is a value specified by charIndex, to the client's device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_HPSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - A notification is not enabled by the client.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

# HPS Client Functions

## Description

APIs unique to HPS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Hpsc

## Functions

- CYBLE_API_RESULT_T CyBle_HpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_HpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_HpscSetLongCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, CYBLE_HPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HPS_CHAR_INDEX_T charIndex, CYBLE_HPS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_HPSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HPS_CHAR_VALUE_T.
- Otherwise (if the HPS service-specific callback is not registered):
- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there were some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_HPS_CHAR_VALUE_T.

Otherwise (if the HPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HpscSetLongCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Sends a request to set a long characteristic value of the service, which is a value identified by charIndex, to the server's device.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HPS_CHAR_VALUE_T.
- Otherwise (if the HPS service-specific callback is not registered):
- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HpscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

This function is used to read a long characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size of the buffer to store long characteristic value. |
| *attrValue* | The pointer to the buffer where the read long characteristic value should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_HPS_CHAR_VALUE_T.
- Otherwise (if the HPS service-specific callback is not registered):
- CYBLE_EVT_GATTC_READ_BLOB_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, CYBLE_HPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic descriptor to the server, which is identified by charIndex and descrIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_HPSS_NOTIFICATION_ENABLED
- CYBLE_EVT_HPSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HPS_CHAR_VALUE_T. Otherwise (if the HPS service-specific callback is not registered):
- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HPS_CHAR_INDEX_T *charIndex*, CYBLE_HPS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HPS service-specific callback is registered (with CyBle_HpsRegisterAttrCallback):

- CYBLE_EVT_HPSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully read on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_HPS_DESCR_VALUE_T.

Otherwise (if the HPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# HPS Definitions and Data Structures

## Description

Contains the HPS specific definitions and data structures used in the HPS APIs.

## Data Structures

- struct CYBLE_HPS_CHAR_VALUE_T
- struct CYBLE_HPS_DESCR_VALUE_T
- struct CYBLE_HPSS_CHAR_T
- struct CYBLE_HPSS_T
- struct CYBLE_HPSC_CHAR_T
- struct CYBLE_HPSC_T

## Enumerations

- enum CYBLE_HPS_CHAR_INDEX_T
- enum CYBLE_HPS_DESCR_INDEX_T
- enum CYBLE_HPS_HTTP_REQUEST_T

## Data Structure Documentation

### struct CYBLE_HPS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_HPS_CHAR_INDEX_T charIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HPS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_HPS_CHAR_INDEX_T CYBLE_HPS_CHAR_VALUE_T::charIndex**

Index of service characteristic

**CYBLE_GATT_ERR_CODE_T CYBLE_HPS_CHAR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_HPS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_HPS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_HPS_CHAR_INDEX_T charIndex
- CYBLE_HPS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HPS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_HPS_CHAR_INDEX_T CYBLE_HPS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_HPS_DESCR_INDEX_T CYBLE_HPS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_ERR_CODE_T CYBLE_HPS_DESCR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_HPS_DESCR_VALUE_T::value**
Characteristic value

**struct CYBLE_HPSS_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_HPS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSS_CHAR_T::charHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSS_CHAR_T::descrHandle[CYBLE_HPS_DESCR_COUNT]**
Array of descriptor handles

**struct CYBLE_HPSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_HPSS_CHAR_T charInfo [CYBLE_HPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSS_T::serviceHandle**
HTTP Proxy Service handle

**CYBLE_HPSS_CHAR_T CYBLE_HPSS_T::charInfo[CYBLE_HPS_CHAR_COUNT]**
Array of characteristics and descriptors handles

**struct CYBLE_HPSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_HPS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSC_CHAR_T::valueHandle**

Handle of characteristic value

**uint8 CYBLE_HPSC_CHAR_T::properties**

Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSC_CHAR_T::endHandle**

End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSC_CHAR_T::descrHandle[CYBLE_HPS_DESCR_COUNT]**

Array of descriptor handles

**struct CYBLE_HPSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_HPSC_CHAR_T charInfo [CYBLE_HPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HPSC_T::serviceHandle**

HTTP Proxy Service handle

**CYBLE_HPSC_CHAR_T CYBLE_HPSC_T::charInfo[CYBLE_HPS_CHAR_COUNT]**

HTTP Proxy Service characteristics info structure

## Enumeration Type Documentation

**enum CYBLE_HPS_CHAR_INDEX_T**

HPS Characteristic indexes

**Enumerator**

***CYBLE_HPS_URI*** Universal Resource Identifier Characteristics index

***CYBLE_HPS_HTTP_HEADERS*** HTTP Headers Characteristics index

***CYBLE_HPS_HTTP_ENTITY_BODY*** HTTP Entity Body Characteristics index

***CYBLE_HPS_HTTP_CP*** HTTP Control Point Characteristics index

***CYBLE_HPS_HTTP_STATUS_CODE*** HTTP Status Code Characteristics index

***CYBLE_HPS_HTTPS_SECURITY*** HTTPS Security Characteristics index

***CYBLE_HPS_CHAR_COUNT*** Total count of HPS Characteristics

**enum CYBLE_HPS_DESCR_INDEX_T**

HPS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_HPS_CCCD*** Client Characteristic Configuration Descriptor index

***CYBLE_HPS_DESCR_COUNT*** Total count of Descriptors

**enum CYBLE_HPS_HTTP_REQUEST_T**

HTTP Requests

**Enumerator**

***CYBLE_HPS_HTTP_GET*** HTTP GET Request

***CYBLE_HPS_HTTP_HEAD*** HTTP HEAD Request

***CYBLE_HPS_HTTP_POST*** HTTP POST Request

***CYBLE_HPS_HTTP_PUT*** HTTP PUT Request

***CYBLE_HPS_HTTP_DELETE*** HTTP DELETE Request

***CYBLE_HPS_HTTPS_GET*** HTTS GET Request

***CYBLE_HPS_HTTPS_HEAD*** HTTPS HEAD Request

***CYBLE_HPS_HTTPS_POST*** HTTPS POST Request

***CYBLE_HPS_HTTPS_PUT*** HTTPS PUT Request

***CYBLE_HPS_HTTPS_DELETE*** HTTPS DELETE Request

***CYBLE_HPS_HTTP_REQ_CANCEL*** HTTP CANCEL Request

# Health Thermometer Service (HTS)

## Description

The Health Thermometer Service exposes temperature and other data related to a thermometer used for healthcare applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The HTS API names begin with CyBle_Hts. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- HTS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- HTS Server Functions
  *APIs unique to HTS designs configured as a GATT Server role.*
- HTS Client Functions
  *APIs unique to HTS designs configured as a GATT Client role.*
- HTS Definitions and Data Structures
  *Contains the HTS specific definitions and data structures used in the HTS APIs.*

# HTS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Hts

## Functions

- void CyBle_HtsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_HtsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for HTS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_HTSS_NOTIFICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_HTS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

## HTS Server Functions

### Description

APIs unique to HTS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Htss

### Functions

- CYBLE_API_RESULT_T CyBle_HtssSetCharacteristicValue (CYBLE_HTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtssGetCharacteristicValue (CYBLE_HTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtssSetCharacteristicDescriptor (CYBLE_HTS_CHAR_INDEX_T charIndex, CYBLE_HTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtssGetCharacteristicDescriptor (CYBLE_HTS_CHAR_INDEX_T charIndex, CYBLE_HTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_HtssSetCharacteristicValue (CYBLE_HTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size (in Bytes) of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HtssGetCharacteristicValue (CYBLE_HTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic value of the service, which is a value identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HtssSetCharacteristicDescriptor (CYBLE_HTS_CHAR_INDEX_T *charIndex*, CYBLE_HTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HtssGetCharacteristicDescriptor (CYBLE_HTS_CHAR_INDEX_T *charIndex*, CYBLE_HTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_HtssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends notification with a characteristic value of the Health Thermometer Service, which is a value specified by charIndex, to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_HTSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

**CYBLE_API_RESULT_T CyBle_HtssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends indication with a characteristic value of the Health Thermometer Service, which is a value specified by charIndex, to the Client device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_HTSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HTS service-specific callback is registered (with CyBle_HtsRegisterAttrCallback):

- CYBLE_EVT_HTSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the HTS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# HTS Client Functions

## Description

APIs unique to HTS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Htsc

## Functions

- CYBLE_API_RESULT_T CyBle_HtscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_HtscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex, CYBLE_HTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_HtscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_HTS_CHAR_INDEX_T charIndex, CYBLE_HTS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_HtscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_HTSS_CHAR_WRITE events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HTS service-specific callback is registered (with CyBle_HtsRegisterAttrCallback):

- CYBLE_EVT_HTSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_HTS_CHAR_VALUE_T.

Otherwise (if the HTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HtscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HTS service-specific callback is registered (with CyBle_HtsRegisterAttrCallback):

- CYBLE_EVT_HTSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_HTS_CHAR_VALUE_T.

Otherwise (if the HTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HtscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*, CYBLE_HTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic descriptor to the server, which is identified by charIndex and descrIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_HTSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_HTSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_HTSS_INDICATION_ENABLED;
- CYBLE_EVT_HTSS_INDICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HTS service-specific callback is registered (with CyBle_HtsRegisterAttrCallback):

- CYBLE_EVT_HTSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_HTS_DESCR_VALUE_T.

Otherwise (if the HTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_HtscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_HTS_CHAR_INDEX_T *charIndex*, CYBLE_HTS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the HTS service-specific callback is registered (with CyBle_HtsRegisterAttrCallback):

- CYBLE_EVT_HTSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_HTS_DESCR_VALUE_T.

Otherwise (if the HTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# HTS Definitions and Data Structures

## Description

Contains the HTS specific definitions and data structures used in the HTS APIs.

## Data Structures

- struct CYBLE_HTSS_CHAR_T
- struct CYBLE_HTSS_T
- struct CYBLE_HTSC_CHAR_T
- struct CYBLE_HTSC_T
- struct CYBLE_HTS_CHAR_VALUE_T
- struct CYBLE_HTS_DESCR_VALUE_T
- struct CYBLE_HTS_FLOAT32

## Enumerations

- enum CYBLE_HTS_CHAR_INDEX_T
- enum CYBLE_HTS_DESCR_INDEX_T
- enum CYBLE_HTS_TEMP_TYPE_T

## Data Structure Documentation

### struct CYBLE_HTSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_HTS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSS_CHAR_T::descrHandle[CYBLE_HTS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_HTSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_HTSS_CHAR_T charInfo [CYBLE_HTS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSS_T::serviceHandle**
Health Thermometer Service handle

**CYBLE_HTSS_CHAR_T CYBLE_HTSS_T::charInfo[CYBLE_HTS_CHAR_COUNT]**
Health Thermometer Service Characteristic handles

**struct CYBLE_HTSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_HTS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSC_CHAR_T::descrHandle[CYBLE_HTS_DESCR_COUNT]**
Handle of descriptor

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSC_CHAR_T::valueHandle**
Handle of Report characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_HTSC_CHAR_T::endHandle**
End handle of characteristic

**uint8 CYBLE_HTSC_CHAR_T::properties**
Properties for value field

**struct CYBLE_HTSC_T**

**Data Fields**

- CYBLE_HTSC_CHAR_T charInfo [CYBLE_HTS_CHAR_COUNT]

**Field Documentation**

**CYBLE_HTSC_CHAR_T CYBLE_HTSC_T::charInfo[CYBLE_HTS_CHAR_COUNT]**
Characteristics handles array

**struct CYBLE_HTS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_HTS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HTS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_HTS_CHAR_INDEX_T CYBLE_HTS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_HTS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_HTS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle

- CYBLE_HTS_CHAR_INDEX_T charIndex
- CYBLE_HTS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_HTS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_HTS_CHAR_INDEX_T CYBLE_HTS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_HTS_DESCR_INDEX_T CYBLE_HTS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_VALUE_T* CYBLE_HTS_DESCR_VALUE_T::value**
Characteristic value

**struct CYBLE_HTS_FLOAT32**

**Data Fields**

- int8 exponent
- int32 mantissa

**Field Documentation**

**int8 CYBLE_HTS_FLOAT32::exponent**
Base 10 exponent

**int32 CYBLE_HTS_FLOAT32::mantissa**
Mantissa, should be using only 24 bits

## Enumeration Type Documentation

**enum CYBLE_HTS_CHAR_INDEX_T**
HTS Characteristic indexes

**Enumerator**

***CYBLE_HTS_TEMP_MEASURE*** Temperature Measurement characteristic index
***CYBLE_HTS_TEMP_TYPE*** Temperature Type characteristic index
***CYBLE_HTS_INTERM_TEMP*** Intermediate Temperature characteristic index
***CYBLE_HTS_MEASURE_INTERVAL*** Measurement Interval characteristic index
***CYBLE_HTS_CHAR_COUNT*** Total count of HTS characteristics

**enum CYBLE_HTS_DESCR_INDEX_T**
HTS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_HTS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_HTS_VRD*** Valid Range descriptor index
***CYBLE_HTS_DESCR_COUNT*** Total count of descriptors

**enum CYBLE_HTS_TEMP_TYPE_T**
Temperature Type measurement indicates where the temperature was measured

**Enumerator**

***CYBLE_HTS_TEMP_TYPE_ARMPIT*** Armpit

***CYBLE_HTS_TEMP_TYPE_BODY*** Body (general)
***CYBLE_HTS_TEMP_TYPE_EAR*** Ear (usually ear lobe)
***CYBLE_HTS_TEMP_TYPE_FINGER*** Finger
***CYBLE_HTS_TEMP_TYPE_GI_TRACT*** Gastro-intestinal Tract
***CYBLE_HTS_TEMP_TYPE_MOUTH*** Mouth
***CYBLE_HTS_TEMP_TYPE_RECTUM*** Rectum
***CYBLE_HTS_TEMP_TYPE_TOE*** Toe
***CYBLE_HTS_TEMP_TYPE_TYMPANUM*** Tympanum (ear drum)

# Immediate Alert Service (IAS)

## Description

The Immediate Alert Service exposes a control point to allow a peer device to cause the device to immediately alert.

The Immediate Alert Service uses the Alert Level Characteristic to cause an alert when it is written with a value other than "No Alert".

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The IAS API names begin with CyBle_Ias. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- IAS Server Functions
  *APIs unique to IAS designs configured as a GATT Server role.*
- IAS Client Functions
  *APIs unique to IAS designs configured as a GATT Client role.*
- IAS Definitions and Data Structures
  *Contains the IAS specific definitions and data structures used in the IAS APIs.*

# IAS Server Functions

## Description

APIs unique to IAS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Iass

## Functions

- void CyBle_IasRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)
- CYBLE_API_RESULT_T CyBle_IassGetCharacteristicValue (CYBLE_IAS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

### void CyBle_IasRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for IAS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_IASS_NOTIFICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_IAS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

**Note** : IAS only has events for the GATT server. There are no events for the GATT client since the client sends data without waiting for response. Therefore there is no need to register a callback through CyBle_IasRegisterAttrCallback for an IAS GATT client.

**CYBLE_API_RESULT_T CyBle_IassGetCharacteristicValue (CYBLE_IAS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the Alert Level characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the Alert Level characteristic. |
| *attrSize* | The size of the Alert Level characteristic value attribute. |
| *attrValue* | The pointer to the location where the Alert Level characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed

# IAS Client Functions

## Description

APIs unique to IAS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Iasc

## Functions

- CYBLE_API_RESULT_T CyBle_IascSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IAS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_IascSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IAS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_IASS_WRITE_CHAR_CMD event is generated.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the Alert Level service characteristic. |
| *attrSize* | The size of the Alert Level characteristic value attribute. |
| *attrValue* | The pointer to the Alert Level characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

## IAS Definitions and Data Structures

### Description

Contains the IAS specific definitions and data structures used in the IAS APIs.

### Data Structures

- struct CYBLE_IASS_T
- struct CYBLE_IAS_CHAR_VALUE_T
- struct CYBLE_IASC_T

### Enumerations

- enum CYBLE_IAS_CHAR_INDEX_T

### Data Structure Documentation

**struct CYBLE_IASS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T alertLevelCharHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IASS_T::serviceHandle**

Immediate Alert Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IASS_T::alertLevelCharHandle**
Handle of Alert Level Characteristic

**struct CYBLE_IAS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_IAS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_IAS_CHAR_VALUE_T::connHandle**
Connection handle

**CYBLE_IAS_CHAR_INDEX_T CYBLE_IAS_CHAR_VALUE_T::charIndex**
Characteristic index of Immediate Alert Service

**CYBLE_GATT_VALUE_T* CYBLE_IAS_CHAR_VALUE_T::value**
Pointer to value of Immediate Alert Service characteristic

**struct CYBLE_IASC_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T alertLevelChar

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_IASC_T::alertLevelChar**
Handle of Alert Level Characteristic of Immediate Alert Service

### Enumeration Type Documentation

**enum CYBLE_IAS_CHAR_INDEX_T**
Immediate Alert Service Characteristic indexes

**Enumerator**

***CYBLE_IAS_ALERT_LEVEL*** Alert Level Characteristic index
***CYBLE_IAS_CHAR_COUNT*** Total count of characteristics

## Indoor Positioning Service (IPS)

### Description

The Indoor Positioning exposes coordinates and other location related information via an advertisement or indicates that the device address can be used for location look-up, enabling mobile devices to find their position.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The IPS API names begin with CyBle_Ips. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- IPS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- IPS Server Functions

*APIs unique to IPS designs configured as a GATT Server role.*

- IPS Client Functions
  *APIs unique to IPS designs configured as a GATT Client role.*
- IPS Definitions and Data Structures
  *Contains the IPS specific definitions and data structures used in the IPS APIs.*

# IPS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Ips

## Functions

- void CyBle_IpsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_IpsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for IPS Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode: Indicates the event that triggered this callback (e.g. CYBLE_EVT_IPS_NOTIFICATION_ENABLED).<br>• eventParam: Contains the parameters corresponding to the current event. (e.g. Pointer to CYBLE_IPS_CHAR_VALUE_T structure that contains details of the characteristic for which the notification enabled event was triggered). |

# IPS Server Functions

## Description

APIs unique to IPS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Ipss

## Functions

- CYBLE_API_RESULT_T CyBle_IpssSetCharacteristicValue (CYBLE_IPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpssGetCharacteristicValue (CYBLE_IPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_IpssSetCharacteristicDescriptor (CYBLE_IPS_CHAR_INDEX_T charIndex, CYBLE_IPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpssGetCharacteristicDescriptor (CYBLE_IPS_CHAR_INDEX_T charIndex, CYBLE_IPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_IpssSetCharacteristicValue (CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic value of the service in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size (in bytes) of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_IpssGetCharacteristicValue (CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic value of the service, which is a value identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_IpssSetCharacteristicDescriptor (CYBLE_IPS_CHAR_INDEX_T *charIndex*, CYBLE_IPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Set a characteristic descriptor of a specified characteristic of the Indoor Positioning Service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_IpssGetCharacteristicDescriptor (CYBLE_IPS_CHAR_INDEX_T *charIndex*, CYBLE_IPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the Indoor Positioning Service from the local GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

# IPS Client Functions

## Description

APIs unique to IPS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Ipsc

## Functions

- CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicValueWithoutResponse (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpscReliableWriteCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_IpscGetMultipleCharacteristicValues (CYBLE_CONN_HANDLE_T connHandle, const CYBLE_IPS_CHAR_INDEX_T *charIndexesList, uint8 numberOfCharIndexes)
- CYBLE_API_RESULT_T CyBle_IpscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, CYBLE_IPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_IpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_IPS_CHAR_INDEX_T charIndex, CYBLE_IPS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicValueWithoutResponse (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server without response.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_IPSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

The Write Response just confirms the operation success.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_WRITE_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, etc.) are provided with an event parameter structure of type CYBLE_IPS_CHAR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_IpscReliableWriteCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to perform a reliable write command for the Indoor Positioning Service (identified by charIndex) value attribute to the server.

The Write response just confirms the operation success.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_IPS_CHAR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_IpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server, as identified by its charIndex

The Read Response returns the characteristic Value in the Attribute Value parameter.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_IPS_CHAR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_IpscGetMultipleCharacteristicValues (CYBLE_CONN_HANDLE_T *connHandle*, const CYBLE_IPS_CHAR_INDEX_T **charIndexesList*, uint8 *numberOfCharIndexes*)

This function reads multiple Characteristic Values from a GATT Server when the GATT Client knows the Characteristic value handles. This is a non-blocking function.

Internally, Read Multiple Request is sent to the peer device in response to which Read Multiple Response is received. This results in CYBLE_EVT_GATTC_READ_MULTI_RSP event, which is propagated to the application layer.

An Error Response event is sent by the server (CYBLE_EVT_GATTC_ERROR_RSP) in response to the Read Multiple Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on any of the Characteristic values. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to Bluetooth 4.1 core specification, Volume 3, Part G, section 4.8.4 for more details on the sequence of operations.

**Parameters:**

| | |
|---|---|
| *connHandle* | Connection handle to identify the peer GATT entity, of type CYBLE_CONN_HANDLE_T. |
| *charIndexes List* | Pointer to a list of Characteristic value handles |
| *numberOfCharIndexes* | Number of requested Characteristic handles |

**Returns:**

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

| Errors codes | Description |
|---|---|
| CYBLE_ERROR_OK | On successful operation |
| CYBLE_ERROR_INVALID_PARAMETER | 'connHandle' value does not represent any existing entry in the Stack |
| CYBLE_ERROR_INVALID_OPERATION | This operation is not permitted |
| CYBLE_ERROR_MEMORY_ALLOCATION_FAILED | Memory allocation failed |
| CYBLE_ERROR_INVALID_STATE | Connection with the Client is not established. |
| CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE | The peer device doesn't have the particular characteristic. |

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_READ_MULTIPLE_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_IPS_CHAR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_MULTI_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_IpscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)**

Sends a request to read a long characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the buffer where the read long characteristic descriptor value should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_IPS_CHAR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_BLOB_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_IpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, CYBLE_IPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server, as identified by its charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data type should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_WRITE_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index etc.) are provided with an event parameter structure of type CYBLE_IPS_DESCR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_IpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_IPS_CHAR_INDEX_T *charIndex*, CYBLE_IPS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the IPS service-specific callback is registered (with CyBle_IpsRegisterAttrCallback):

- CYBLE_EVT_IPSC_READ_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index, value, etc.) are provided with an event parameter structure of type CYBLE_IPS_DESCR_VALUE_T.

Otherwise (if the IPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# IPS Definitions and Data Structures

## Description

Contains the IPS specific definitions and data structures used in the IPS APIs.

## Data Structures

- struct CYBLE_IPSS_CHAR_T
- struct CYBLE_IPSS_CHAR_INFO_PTR_T
- struct CYBLE_IPSS_T
- struct CYBLE_IPSC_CHAR_T
- struct CYBLE_IPSC_CHAR_INFO_PTR_T
- struct CYBLE_IPSC_T
- struct CYBLE_IPS_CHAR_VALUE_T
- struct CYBLE_IPS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_IPS_CHAR_INDEX_T
- enum CYBLE_IPS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_IPSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_IPS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSS_CHAR_T::charHandle**
Handles of Characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSS_CHAR_T::descrHandle[CYBLE_IPS_DESCR_COUNT]**
Array of Descriptor handles

**struct CYBLE_IPSS_CHAR_INFO_PTR_T**

**Data Fields**

- CYBLE_IPSS_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_IPSS_CHAR_T* CYBLE_IPSS_CHAR_INFO_PTR_T::charInfoPtr**
Pointer to CYBLE_IPSS_CHAR_T which holds information about specific IP Characteristic

**struct CYBLE_IPSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_IPSS_CHAR_T charInfo [CYBLE_IPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSS_T::serviceHandle**
Indoor Positioning Service handle

**CYBLE_IPSS_CHAR_T CYBLE_IPSS_T::charInfo[CYBLE_IPS_CHAR_COUNT]**
Indoor Positioning Service Array with pointers to Characteristic handles.

**struct CYBLE_IPSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_IPS_DESCR_COUNT]
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSC_CHAR_T::valueHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSC_CHAR_T::endHandle**
End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSC_CHAR_T::descrHandle[CYBLE_IPS_DESCR_COUNT]**
Array of Descriptor handles

**uint8 CYBLE_IPSC_CHAR_T::properties**
Properties for value field

**struct CYBLE_IPSC_CHAR_INFO_PTR_T**

**Data Fields**

- CYBLE_IPSC_CHAR_T * charInfoPtr

**Field Documentation**

**CYBLE_IPSC_CHAR_T* CYBLE_IPSC_CHAR_INFO_PTR_T::charInfoPtr**
Pointer to CYBLE_IPSC_CHAR_T which holds information about specific IP Characteristic.

**struct CYBLE_IPSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_IPSC_CHAR_T charInfo [CYBLE_IPS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_IPSC_T::serviceHandle**
Indoor Positioning Service handle

**CYBLE_IPSC_CHAR_T CYBLE_IPSC_T::charInfo[CYBLE_IPS_CHAR_COUNT]**
Indoor Positioning Service characteristics info array

**struct CYBLE_IPS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_IPS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_IPS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_IPS_CHAR_INDEX_T CYBLE_IPS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_IPS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_IPS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**struct CYBLE_IPS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_IPS_CHAR_INDEX_T charIndex
- CYBLE_IPS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_IPS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_IPS_CHAR_INDEX_T CYBLE_IPS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_IPS_DESCR_INDEX_T CYBLE_IPS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_ERR_CODE_T CYBLE_IPS_DESCR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_IPS_DESCR_VALUE_T::value**
Characteristic value

### Enumeration Type Documentation

**enum CYBLE_IPS_CHAR_INDEX_T**

IPS Characteristic indexes

**Enumerator**

***CYBLE_IPS_INDOOR_POSITINING_CONFIG*** Set of characteristic values included in the Indoor Positioning Service AD type.

***CYBLE_IPS_LATITUDE*** WGS84 North coordinate of the device.

***CYBLE_IPS_LONGITUDE*** WGS84 East coordinate of the device.

***CYBLE_IPS_LOCAL_NORTH_COORDINATE*** North coordinate of the device using local coordinate system.

***CYBLE_IPS_LOCAL_EAST_COORDINATE*** East coordinate of the device using local coordinate system.

***CYBLE_IPS_FLOOR_NUMBER*** Describes in which floor the device is installed in.

***CYBLE_IPS_ALTITUDE*** Altitude of the device.

***CYBLE_IPS_UNCERTAINTY*** Uncertainty of the location information the device exposes.

***CYBLE_IPS_LOCATION_NAME*** Name of the location the device is installed in.

***CYBLE_IPS_CHAR_COUNT*** Total count of IPS characteristics

**enum CYBLE_IPS_DESCR_INDEX_T**

IPS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_IPS_CEPD*** Characteristic Extended Properties descriptor index

***CYBLE_IPS_SCCD*** Server Characteristic Configuration Descriptor index

***CYBLE_IPS_DESCR_COUNT*** Total count of descriptors

## Link Loss Service (LLS)

### Description

The Link Loss Service uses the Alert Level Characteristic to cause an alert in the device when the link is lost.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The LLS API names begin with CyBle_Lls. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- LLS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- LLS Server Functions
  *APIs unique to LLS designs configured as a GATT Server role.*
- LLS Client Functions
  *APIs unique to LLS designs configured as a GATT Client role.*
- LLS Definitions and Data Structures
  *Contains the LLS specific definitions and data structures used in the LLS APIs.*

# LLS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Lls

## Functions

- void CyBle_LlsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_LlsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for Link Loss Service is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_LLSS_NOTIFICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_LLS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# LLS Server Functions

## Description

APIs unique to LLS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Llss

## Functions

- CYBLE_API_RESULT_T CyBle_LlssGetCharacteristicValue (CYBLE_LLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

### CYBLE_API_RESULT_T CyBle_LlssGetCharacteristicValue (CYBLE_LLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)

Gets an Alert Level characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of an Alert Level characteristic. |
| *attrSize* | The size of the Alert Level characteristic value attribute. |
| *attrValue* | The pointer to the location where an Alert Level characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed

# LLS Client Functions

## Description

APIs unique to LLS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Llsc

## Functions

- CYBLE_API_RESULT_T CyBle_LlscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LLS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LLS_CHAR_INDEX_T charIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_LlscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LLS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the Alert Level characteristic value of the Link Loss Service, which is identified by charIndex. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_LLSS_WRITE_CHAR_REQ event is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the Alert Level service characteristic. |
| *attrSize* | The size of the Alert Level characteristic value attribute. |
| *attrValue* | The pointer to the Alert Level characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LLS service-specific callback is registered (with CyBle_LlsRegisterAttrCallback):

- CYBLE_EVT_LLSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_LLS_CHAR_VALUE_T.

Otherwise (if the LLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_LlscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LLS_CHAR_INDEX_T *charIndex*)**

Sends a request to get characteristic value of the Link Loss Service, which is identified by charIndex.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_LLSC_READ_CHAR_RESPONSE
- CYBLE_EVT_GATTC_ERROR_RSP

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the Link Loss Service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LLS service-specific callback is registered (with CyBle_LlsRegisterAttrCallback):

- CYBLE_EVT_LLSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_LLS_CHAR_VALUE_T.

Otherwise (if the LLS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# LLS Definitions and Data Structures

## Description

Contains the LLS specific definitions and data structures used in the LLS APIs.

## Data Structures

- struct CYBLE_LLS_CHAR_VALUE_T
- struct CYBLE_LLSS_T
- struct CYBLE_LLSC_T

## Enumerations

- enum CYBLE_LLS_CHAR_INDEX_T

## Data Structure Documentation

### struct CYBLE_LLS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_LLS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_LLS_CHAR_VALUE_T::connHandle**
Connection handle

**CYBLE_LLS_CHAR_INDEX_T CYBLE_LLS_CHAR_VALUE_T::charIndex**
Characteristic index of Link Loss Service

**CYBLE_GATT_VALUE_T* CYBLE_LLS_CHAR_VALUE_T::value**
Pointer to value of Link Loss Service characteristic

### struct CYBLE_LLSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T alertLevelCharHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LLSS_T::serviceHandle**
Link Loss Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LLSS_T::alertLevelCharHandle**
Handle of Alert Level Characteristic

### struct CYBLE_LLSC_T

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T alertLevelChar

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_LLSC_T::alertLevelChar**
Handle of Alert Level Characteristic of Link Loss Service

## Enumeration Type Documentation

### enum CYBLE_LLS_CHAR_INDEX_T

Link Loss Service Characteristic indexes

**Enumerator**

***CYBLE_LLS_ALERT_LEVEL*** Alert Level Characteristic index

***CYBLE_LLS_CHAR_COUNT*** Total count of characteristics

# Location and Navigation Service (LNS)

## Description

The Location and Navigation Service exposes location and navigation-related data from a Location and Navigation sensor (Server) intended for outdoor activity applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The LNS API names begin with CyBle_Lns. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- LNS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- LNS Server Functions
  *APIs unique to LNS designs configured as a GATT Server role.*
- LNS Client Functions
  *APIs unique to LNS designs configured as a GATT Client role.*
- LNS Definitions and Data Structures
  *Contains the LNS specific definitions and data structures used in the LNS APIs.*

# LNS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Lns

## Functions

- void CyBle_LnsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_LnsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for LNS is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# LNS Server Functions

## Description

APIs unique to LNS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Lnss

## Functions

- CYBLE_API_RESULT_T CyBle_LnssSetCharacteristicValue (CYBLE_LNS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnssGetCharacteristicValue (CYBLE_LNS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnssGetCharacteristicDescriptor (CYBLE_LNS_CHAR_INDEX_T charIndex, CYBLE_LNS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_LnssSetCharacteristicValue (CYBLE_LNS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_LnssGetCharacteristicValue (CYBLE_LNS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Characteristic value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Characteristic is absent.

**CYBLE_API_RESULT_T CyBle_LnssGetCharacteristicDescriptor (CYBLE_LNS_CHAR_INDEX_T *charIndex*, CYBLE_LNS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Characteristic Descriptor value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Characteristic is absent.

**CYBLE_API_RESULT_T CyBle_LnssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic value, as identified by the charIndex.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_LNSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client

**CYBLE_API_RESULT_T CyBle_LnssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication of the specified characteristic value, as identified by the charIndex.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_LNSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client
- CYBLE_ERROR_IND_DISABLED - Indication is disabled for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LNS service-specific callback is registered (with CyBle_LnsRegisterAttrCallback):

- CYBLE_EVT_LNSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the LNS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# LNS Client Functions

## Description

APIs unique to LNS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Lnsc

## Functions

- CYBLE_API_RESULT_T CyBle_LnscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_LnscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex, CYBLE_LNS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_LnscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_LNS_CHAR_INDEX_T charIndex, CYBLE_LNS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_LnscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_LNSS_WRITE_CHAR event is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LNS service-specific callback is registered (with CyBle_LnsRegisterAttrCallback):

- CYBLE_EVT_LNSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_LNS_CHAR_VALUE_T.

Otherwise (if the LNS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_LnscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server, as identified by its charIndex

The Read Response returns the characteristic Value in the Attribute Value parameter.

The Read Response only contains the characteristic Value that is less than or equal to (MTU - 1) octets in length. If the characteristic Value is greater than (MTU - 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the characteristic Value is required.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully

- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LNS service-specific callback is registered (with CyBle_LnsRegisterAttrCallback):

- CYBLE_EVT_LNSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_LNS_CHAR_VALUE_T.

Otherwise (if the LNS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_LnscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*, CYBLE_LNS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server, as identified by its charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_LNSS_INDICATION_ENABLED
- CYBLE_EVT_LNSS_INDICATION_DISABLED
- CYBLE_EVT_LNSS_NOTIFICATION_ENABLED
- CYBLE_EVT_LNSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LNS service-specific callback is registered (with CyBle_LnsRegisterAttrCallback):

- CYBLE_EVT_LNSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_LNS_DESCR_VALUE_T.

Otherwise (if the LNS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_LnscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_LNS_CHAR_INDEX_T *charIndex*, CYBLE_LNS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the LNS service-specific callback is registered (with CyBle_LnsRegisterAttrCallback):

- CYBLE_EVT_LNSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_LNS_DESCR_VALUE_T.

Otherwise (if the LNS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# LNS Definitions and Data Structures

## Description

Contains the LNS specific definitions and data structures used in the LNS APIs.

## Data Structures

- struct CYBLE_LNSS_CHAR_T

- struct CYBLE_LNSS_T
- struct CYBLE_LNSC_CHAR_T
- struct CYBLE_LNSC_T
- struct CYBLE_LNS_CHAR_VALUE_T
- struct CYBLE_LNS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_LNS_CHAR_INDEX_T
- enum CYBLE_LNS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_LNSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_LNS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSS_CHAR_T::descrHandle[CYBLE_LNS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_LNSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_LNSS_CHAR_T charInfo [CYBLE_LNS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSS_T::serviceHandle**

Location and Navigation Service handle

**CYBLE_LNSS_CHAR_T CYBLE_LNSS_T::charInfo[CYBLE_LNS_CHAR_COUNT]**

Location and Navigation Service characteristics info array

### struct CYBLE_LNSC_CHAR_T

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_LNS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_LNSC_CHAR_T::properties**

Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSC_CHAR_T::valueHandle**

Handle of server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSC_CHAR_T::descrHandle[CYBLE_LNS_DESCR_COUNT]**

Location and Navigation client char. descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_LNSC_CHAR_T::endHandle**
Characteristic End Handle

**struct CYBLE_LNSC_T**

**Data Fields**

- CYBLE_LNSC_CHAR_T charInfo [CYBLE_LNS_CHAR_COUNT]

**Field Documentation**

**CYBLE_LNSC_CHAR_T CYBLE_LNSC_T::charInfo[CYBLE_LNS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_LNS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_LNS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_LNS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_LNS_CHAR_INDEX_T CYBLE_LNS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_LNS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_LNS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_LNS_CHAR_INDEX_T charIndex
- CYBLE_LNS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_LNS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_LNS_CHAR_INDEX_T CYBLE_LNS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_LNS_DESCR_INDEX_T CYBLE_LNS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_LNS_DESCR_VALUE_T::value**
Descriptor value

## Enumeration Type Documentation

**enum CYBLE_LNS_CHAR_INDEX_T**
LNS Service Characteristics indexes

**Enumerator**

***CYBLE_LNS_FT*** Location and Navigation Feature characteristic index

***CYBLE_LNS_LS*** Location and Speed characteristic index
***CYBLE_LNS_PQ*** Position Quality characteristic index
***CYBLE_LNS_CP*** Location and Navigation Control Point characteristic index
***CYBLE_LNS_NV*** Navigation characteristic index
***CYBLE_LNS_CHAR_COUNT*** Total count of LNS characteristics

**enum CYBLE_LNS_DESCR_INDEX_T**

LNS Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_LNS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_LNS_DESCR_COUNT*** Total count of LNS descriptors

# Next DST Change Service (NDCS)

## Description

The Next DST Change Service enables a BLE device that has knowledge about the next occurrence of a DST change to expose this information to another Bluetooth device. The Service uses the "Time with DST" Characteristic and the functions exposed in this Service are used to interact with that Characteristic.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The NDCS API names begin with CyBle_Ndcs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- NDCS Server and Client Functions
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- NDCS Server Functions
  *APIs unique to NDCS designs configured as a GATT Server role.*
- NDCS Client Functions
  *APIs unique to NDCS designs configured as a GATT Client role.*
- NDCS Definitions and Data Structures
  *Contains the NDCS specific definitions and data structures used in the NDCS APIs.*

# NDCS Server and Client Functions

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Ndcs

## Functions

- void CyBle_NdcsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_NdcsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for Next DST Change Service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for NDCS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

## NDCS Server Functions

### Description

APIs unique to NDCS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Ndcss

### Functions

- CYBLE_API_RESULT_T CyBle_NdcssSetCharacteristicValue (CYBLE_NDCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_NdcssGetCharacteristicValue (CYBLE_NDCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_NdcssSetCharacteristicValue (CYBLE_NDCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets characteristic value of the Next DST Change Service, which is identified by charIndex in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_NDCS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request is handled successfully;
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_NdcssGetCharacteristicValue (CYBLE_NDCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Next DST Change Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_NDCS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request is handled successfully;
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameter failed.

# NDCS Client Functions

## Description

APIs unique to NDCS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Ndcsc

## Functions

- CYBLE_API_RESULT_T CyBle_NdcscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_NDCS_CHAR_INDEX_T charIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_NdcscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_NDCS_CHAR_INDEX_T *charIndex*)**

Sends a request to peer device to set characteristic value of the Next DST Change Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the NDCS service-specific callback is registered (with CyBle_NdcsRegisterAttrCallback):

- CYBLE_EVT_NDCSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_NDCS_CHAR_VALUE_T.

Otherwise (if the NDCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).

- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# NDCS Definitions and Data Structures

## Description

Contains the NDCS specific definitions and data structures used in the NDCS APIs.

## Data Structures

- struct CYBLE_NDCS_CHAR_VALUE_T
- struct CYBLE_NDCSS_T
- struct CYBLE_NDCSC_T

## Enumerations

- enum CYBLE_NDCS_CHAR_INDEX_T

## Data Structure Documentation

### struct CYBLE_NDCS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_NDCS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_NDCS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_NDCS_CHAR_INDEX_T CYBLE_NDCS_CHAR_VALUE_T::charIndex**

Index of Next DST Change Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_NDCS_CHAR_VALUE_T::value**

Characteristic value

### struct CYBLE_NDCSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T timeWithDst

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_NDCSS_T::serviceHandle**

Handle of the Next DST Change Service

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_NDCSS_T::timeWithDst**

Handle of the Time with DST Characteristic

**struct CYBLE_NDCSC_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo [CYBLE_NDCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_NDCSC_T::charInfo[CYBLE_NDCS_CHAR_COUNT]**

Characteristic handle and properties

### Enumeration Type Documentation

**enum CYBLE_NDCS_CHAR_INDEX_T**

Characteristic indexes

**Enumerator**

***CYBLE_NDCS_TIME_WITH_DST*** Time with DST Characteristic index

***CYBLE_NDCS_CHAR_COUNT*** Total count of NDCS Characteristics

## Phone Alert Status Service (PASS)

### Description

The Phone Alert Status Service uses the Alert Status Characteristic and Ringer Setting Characteristic to expose the phone alert status and uses the Ringer Control Point Characteristic to control the phone's ringer into mute or enable.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The PASS API names begin with CyBle_Pass. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- PASS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- PASS Server Functions
  *APIs unique to PASS designs configured as a GATT Server role.*
- PASS Client Functions
  *APIs unique to PASS designs configured as a GATT Client role.*
- PASS Definitions and Data Structures
  *Contains the PASS specific definitions and data structures used in the PASS APIs.*

## PASS Server and Client Function

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Pass

### Functions

- void CyBle_PassRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_PassRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for PASS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

## PASS Server Functions

### Description

APIs unique to PASS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Passs

### Functions

- CYBLE_API_RESULT_T CyBle_PasssSetCharacteristicValue (CYBLE_PASS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PasssGetCharacteristicValue (CYBLE_PASS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PasssGetCharacteristicDescriptor (CYBLE_PASS_CHAR_INDEX_T charIndex, CYBLE_PASS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PasssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PASS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_PasssSetCharacteristicValue (CYBLE_PASS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the value of a characteristic which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_PasssGetCharacteristicValue (CYBLE_PASS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the value of a characteristic which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_PasssGetCharacteristicDescriptor (CYBLE_PASS_CHAR_INDEX_T *charIndex*, CYBLE_PASS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T:

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_PasssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PASS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified by the charIndex characteristic value.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_PASSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consists of the device ID and ATT connection ID. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the Client.

# PASS Client Functions

## Description

APIs unique to PASS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Passc

## Functions

- CYBLE_API_RESULT_T CyBle_PasscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PASS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PasscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PASS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_PasscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PASS_CHAR_INDEX_T charIndex, CYBLE_PASS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PasscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PASS_CHAR_INDEX_T charIndex, CYBLE_PASS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_PasscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PASS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_PASSS_WRITE_CHAR event is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**CYBLE_API_RESULT_T CyBle_PasscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PASS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a Server which is identified by the charIndex.

The Read Response returns the characteristic Value in the Attribute Value parameter.

The Read Response only contains the characteristic Value that is less than or equal to (MTU - 1) octets in length. If the characteristic Value is greater than (MTU - 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the characteristic Value is required.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the Server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PASS service-specific callback is registered (with CyBle_PassRegisterAttrCallback):

- CYBLE_EVT_PASSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_PASS_CHAR_VALUE_T.

Otherwise (if the PASS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_PasscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PASS_CHAR_INDEX_T *charIndex*, CYBLE_PASS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server which is identified by the charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_PASSS_NOTIFICATION_ENABLED
- CYBLE_EVT_PASSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.

- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PASS service-specific callback is registered (with CyBle_PassRegisterAttrCallback):

- CYBLE_EVT_PASSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_PASS_DESCR_VALUE_T.

Otherwise (if the PASS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_PasscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PASS_CHAR_INDEX_T *charIndex*, CYBLE_PASS_DESCR_INDEX_T *descrIndex*)

Gets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PASS service-specific callback is registered (with CyBle_PassRegisterAttrCallback):

- CYBLE_EVT_PASSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_PASS_DESCR_VALUE_T.

Otherwise (if the PASS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# PASS Definitions and Data Structures

## Description

Contains the PASS specific definitions and data structures used in the PASS APIs.

## Data Structures

- struct CYBLE_PASSS_CHAR_T
- struct CYBLE_PASSS_T
- struct CYBLE_PASSC_CHAR_T
- struct CYBLE_PASSC_T
- struct CYBLE_PASS_CHAR_VALUE_T
- struct CYBLE_PASS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_PASS_CHAR_INDEX_T
- enum CYBLE_PASS_DESCR_INDEX_T
- enum CYBLE_PASS_RS_T
- enum CYBLE_PASS_CP_T

## Data Structure Documentation

### struct CYBLE_PASSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_PASS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PASSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_PASSS_CHAR_T::descrHandle[CYBLE_PASS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_PASSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_PASSS_CHAR_T charInfo [CYBLE_PASS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PASSS_T::serviceHandle**

Phone Alert Status Service handle

**CYBLE_PASSS_CHAR_T CYBLE_PASSS_T::charInfo[CYBLE_PASS_CHAR_COUNT]**

Phone Alert Status Service characteristics info array

**struct CYBLE_PASSC_CHAR_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_PASS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_PASSC_CHAR_T::properties**

Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PASSC_CHAR_T::valueHandle**

Handle of Server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_PASSC_CHAR_T::descrHandle[CYBLE_PASS_DESCR_COUNT]**

Phone Alert Status Client characteristics descriptors handles

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PASSC_CHAR_T::endHandle**

Characteristic End Handle

**struct CYBLE_PASSC_T**

**Data Fields**

- CYBLE_PASSC_CHAR_T charInfo [CYBLE_PASS_CHAR_COUNT]

**Field Documentation**

**CYBLE_PASSC_CHAR_T CYBLE_PASSC_T::charInfo[CYBLE_PASS_CHAR_COUNT]**

Characteristics handle and properties array

**struct CYBLE_PASS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_PASS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_PASS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_PASS_CHAR_INDEX_T CYBLE_PASS_CHAR_VALUE_T::charIndex**

Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_PASS_CHAR_VALUE_T::value**

Characteristic value

**struct CYBLE_PASS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_PASS_CHAR_INDEX_T charIndex
- CYBLE_PASS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_PASS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_PASS_CHAR_INDEX_T CYBLE_PASS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_PASS_DESCR_INDEX_T CYBLE_PASS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_PASS_DESCR_VALUE_T::value**
Descriptor value

### Enumeration Type Documentation

**enum CYBLE_PASS_CHAR_INDEX_T**
Service Characteristics indexes
**Enumerator**
***CYBLE_PASS_AS*** Alert Status characteristic index
***CYBLE_PASS_RS*** Ringer Setting characteristic index
***CYBLE_PASS_CP*** Ringer Control Point characteristic index
***CYBLE_PASS_CHAR_COUNT*** Total count of PASS characteristics

**enum CYBLE_PASS_DESCR_INDEX_T**
Service Characteristic Descriptors indexes
**Enumerator**
***CYBLE_PASS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_PASS_DESCR_COUNT*** Total count of PASS descriptors

**enum CYBLE_PASS_RS_T**
Ringer Setting values
**Enumerator**
***CYBLE_PASS_RS_SILENT*** Ringer Silent
***CYBLE_PASS_RS_NORMAL*** Ringer Normal

**enum CYBLE_PASS_CP_T**
Ringer Control Point values
**Enumerator**
***CYBLE_PASS_CP_SILENT*** Silent Mode
***CYBLE_PASS_CP_MUTE*** Mute Once
***CYBLE_PASS_CP_CANCEL*** Cancel Silent Mode

## Pulse Oximeter Service (PLXS)

### Description

The Pulse Oximeter (PLX) Service exposes pulse oximetry data related to a non-invasive pulse oximetry sensor for consumer and professional healthcare applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The PLXS API names begin with CyBle_Plxs. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- PLXS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- PLXS Server Functions
  *APIs unique to PLXS designs configured as a GATT Server role.*
- PLXS Client Functions
  *APIs unique to PLXS designs configured as a GATT Client role.*
- PLXS Definitions and Data Structures
  *Contains the PLXS specific definitions and data structures used in the PLXS APIs.*

## PLXS Server and Client Function

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Plxs

### Functions

- void CyBle_PlxsInit (void)
- void CyBle_PlxsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

#### void CyBle_PlxsInit (void )

This function initializes the Pulse Oximeter Service.

#### void CyBle_PlxsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service-specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of cyble_callback_t for PLX Service is:<br>typedef void (* cyble_callback_t) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# PLXS Server Functions

## Description

APIs unique to PLXS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Plxss

## Functions

- CYBLE_API_RESULT_T CyBle_PlxssSetCharacteristicValue (CYBLE_PLXS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxssGetCharacteristicValue (CYBLE_PLXS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxssSetCharacteristicDescriptor (CYBLE_PLXS_CHAR_INDEX_T charIndex, CYBLE_PLXS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxssGetCharacteristicDescriptor (CYBLE_PLXS_CHAR_INDEX_T charIndex, CYBLE_PLXS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_PlxssSetCharacteristicValue (CYBLE_PLXS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**Events**

None

**CYBLE_API_RESULT_T CyBle_PlxssGetCharacteristicValue (CYBLE_PLXS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | Pointer to the location where Characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_PlxssSetCharacteristicDescriptor (CYBLE_PLXS_CHAR_INDEX_T *charIndex*, CYBLE_PLXS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor of a specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_PLXS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_PLXS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

The return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request is handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent

**CYBLE_API_RESULT_T CyBle_PlxssGetCharacteristicDescriptor (CYBLE_PLXS_CHAR_INDEX_T *charIndex*, CYBLE_PLXS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | Pointer to the location where the descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent

**CYBLE_API_RESULT_T CyBle_PlxssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic to the client device, as defined by the charIndex value.

On enabling notification successfully for a service characteristic, it sends out a 'Handle Value Notification' which results in CYBLE_EVT_PLXSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | Pointer to the Characteristic value data that should be sent to Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client

**CYBLE_API_RESULT_T CyBle_PlxssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication of the specified characteristic to the client device, as defined by the charIndex value.

On enabling indication successfully, if the GATT server has an updated value to be indicated to the GATT Client, it sends out a 'Handle Value Indication' which results in CYBLE_EVT_PLXS_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service-specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | Pointer to the Characteristic value data that should be sent to Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PLXS service-specific callback is registered (with CyBle_PlxsRegisterAttrCallback):

- CYBLE_EVT_PLXSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the PLXS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# PLXS Client Functions

## Description

APIs unique to PLXS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Plxsc

## Functions

- CYBLE_API_RESULT_T CyBle_PlxscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_PlxscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex, CYBLE_PLXS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_PlxscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_PLXS_CHAR_INDEX_T charIndex, CYBLE_PLXS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_PlxscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_PLXSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PLXS service-specific callback is registered (with CyBle_PlxsRegisterAttrCallback):

- CYBLE_EVT_PLXSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_PLXS_CHAR_VALUE_T.

Otherwise (if the PLXS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (cy_stc_ble_gatt_err_param_t).

**CYBLE_API_RESULT_T CyBle_PlxscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PLXS service-specific callback is registered (with CyBle_PlxsRegisterAttrCallback):

- CYBLE_EVT_PLXSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_PLXS_CHAR_VALUE_T.

Otherwise (if the PLXS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (cy_stc_ble_gatt_err_param_t).

**CYBLE_API_RESULT_T CyBle_PlxscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*, CYBLE_PLXS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the Characteristic Descriptor of the specified Characteristic.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_PLXSS_INDICATION_ENABLED
- CYBLE_EVT_PLXSS_INDICATION_DISABLED
- CYBLE_EVT_PLXSS_NOTIFICATION_ENABLED
- CYBLE_EVT_PLXSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of a service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |

| | |
|---|---|
| *attrValue* | Pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PLXS service-specific callback is registered (with CyBle_PlxsRegisterAttrCallback):

- CYBLE_EVT_PLXSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_PLXS_DESCR_VALUE_T.

Otherwise (if the PLXS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (cy_stc_ble_gatt_err_param_t).

**CYBLE_API_RESULT_T CyBle_PlxscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_PLXS_CHAR_INDEX_T *charIndex*, CYBLE_PLXS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_INVALID_STATE - The state is not valid
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the PLXS service-specific callback is registered (with CyBle_PlxsRegisterAttrCallback):

- CYBLE_EVT_PLXSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_PLXS_DESCR_VALUE_T.

Otherwise (if the PLXS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).

- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (cy_stc_ble_gatt_err_param_t).

# PLXS Definitions and Data Structures

## Description

Contains the PLXS specific definitions and data structures used in the PLXS APIs.

## Data Structures

- struct CYBLE_PLXSS_CHAR_T
- struct CYBLE_PLXSS_T
- struct CYBLE_PLXSC_CHAR_T
- struct CYBLE_PLXSC_T
- struct CYBLE_PLXS_CHAR_VALUE_T
- struct CYBLE_PLXS_DESCR_VALUE_T

## Macros

- #define CYBLE_PLXS_DSS_EDU_BIT (0x01u << 0u)
- #define CYBLE_PLXS_DSS_EMD_BIT (0x01u << 1u)
- #define CYBLE_PLXS_DSS_SPID_BIT (0x01u << 2u)
- #define CYBLE_PLXS_DSS_ISD_BIT (0x01u << 3u)
- #define CYBLE_PLXS_DSS_PSD_BIT (0x01u << 4u)
- #define CYBLE_PLXS_DSS_LPD_BIT (0x01u << 5u)
- #define CYBLE_PLXS_DSS_ESD_BIT (0x01u << 6u)
- #define CYBLE_PLXS_DSS_NSD_BIT (0x01u << 7u)
- #define CYBLE_PLXS_DSS_QPD_BIT (0x01u << 8u)
- #define CYBLE_PLXS_DSS_SA_BIT (0x01u << 9u)
- #define CYBLE_PLXS_DSS_SID_BIT (0x01u << 10u)
- #define CYBLE_PLXS_DSS_SUTU_BIT (0x01u << 11u)
- #define CYBLE_PLXS_DSS_USC_BIT (0x01u << 12u)
- #define CYBLE_PLXS_DSS_SD_BIT (0x01u << 13u)
- #define CYBLE_PLXS_DSS_SM_BIT (0x01u << 14u)
- #define CYBLE_PLXS_DSS_SDISC_BIT (0x01u << 15u)
- #define CYBLE_PLXS_MS_MEAS_BIT (0x01u << 5u)
- #define CYBLE_PLXS_MS_EED_BIT (0x01u << 6u)
- #define CYBLE_PLXS_MS_VDATA_BIT (0x01u << 7u)
- #define CYBLE_PLXS_MS_FQDATA_BIT (0x01u << 8u)
- #define CYBLE_PLXS_MS_DFMS_BIT (0x01u << 9u)
- #define CYBLE_PLXS_MS_DFDEMO_BIT (0x01u << 10u)
- #define CYBLE_PLXS_MS_DFTEST_BIT (0x01u << 11u)
- #define CYBLE_PLXS_MS_CALIB_BIT (0x01u << 12u)
- #define CYBLE_PLXS_MS_MUN_BIT (0x01u << 13u)
- #define CYBLE_PLXS_MS_QMD_BIT (0x01u << 14u)
- #define CYBLE_PLXS_MS_IMD_BIT (0x01u << 15u)
- #define CYBLE_PLXS_SCMT_FLAG_TMSF_BIT (0x01u << 0u)
- #define CYBLE_PLXS_SCMT_FLAG_MSF_BIT (0x01u << 1u)
- #define CYBLE_PLXS_SCMT_FLAG_DSSF_BIT (0x01u << 2u)

- #define CYBLE_PLXS_SCMT_FLAG_PAIF_BIT (0x01u << 3u)
- #define CYBLE_PLXS_SCMT_FLAG_DEVCLK_BIT (0x01u << 4u)
- #define CYBLE_PLXS_CTMT_FLAG_FAST_BIT (0x01u << 0u)
- #define CYBLE_PLXS_CTMT_FLAG_SLOW_BIT (0x01u << 1u)
- #define CYBLE_PLXS_CTMT_FLAG_MSF_BIT (0x01u << 2u)
- #define CYBLE_PLXS_CTMT_FLAG_DSSF_BIT (0x01u << 3u)
- #define CYBLE_PLXS_CTMT_FLAG_PAIF_BIT (0x01u << 4u)
- #define CYBLE_PLXS_FEAT_SUPPORT_MEAS_BIT (0x01u << 0u)
- #define CYBLE_PLXS_FEAT_SUPPORT_DSS_BIT (0x01u << 1u)
- #define CYBLE_PLXS_FEAT_SUPPORT_MSSC_BIT (0x01u << 2u)
- #define CYBLE_PLXS_FEAT_SUPPORT_TMSF_BIT (0x01u << 3u)
- #define CYBLE_PLXS_FEAT_SUPPORT_FAST_BIT (0x01u << 4u)
- #define CYBLE_PLXS_FEAT_SUPPORT_SLOW_BIT (0x01u << 5u)
- #define CYBLE_PLXS_FEAT_SUPPORT_PAI_BIT (0x01u << 6u)
- #define CYBLE_PLXS_FEAT_SUPPORT_MBS_BIT (0x01u << 7u)

## Enumerations

- enum CYBLE_PLXS_CHAR_INDEX_T
- enum CYBLE_PLXS_DESCR_INDEX_T
- enum CYBLE_PLXS_RACP_OPC_T
- enum CYBLE_PLXS_RACP_OPR_T
- enum CYBLE_PLXS_RACP_OPD_T
- enum CYBLE_PLXS_RACP_RSP_T

## Data Structure Documentation

### struct CYBLE_PLXSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_PLXS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PLXSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_PLXSS_CHAR_T::descrHandle[CYBLE_PLXS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_PLXSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_PLXSS_CHAR_T charInfo [CYBLE_PLXS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PLXSS_T::serviceHandle**

PLXS handle

**CYBLE_PLXSS_CHAR_T CYBLE_PLXSS_T::charInfo[CYBLE_PLXS_CHAR_COUNT]**

PLXS Characteristic handles

**struct CYBLE_PLXSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_PLXS_DESCR_COUNT]
- uint8_t properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PLXSC_CHAR_T::valueHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PLXSC_CHAR_T::endHandle**

End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_PLXSC_CHAR_T::descrHandle[CYBLE_PLXS_DESCR_COUNT]**

Array of Descriptor handles

**uint8_t CYBLE_PLXSC_CHAR_T::properties**

Properties for value field

**struct CYBLE_PLXSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_PLXSC_CHAR_T charInfo [CYBLE_PLXS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_PLXSC_T::serviceHandle**

Pulse Oximeter Service handle

**CYBLE_PLXSC_CHAR_T CYBLE_PLXSC_T::charInfo[CYBLE_PLXS_CHAR_COUNT]**

PLXS characteristics info array

**struct CYBLE_PLXS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_PLXS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value
- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_PLXS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_PLXS_CHAR_INDEX_T CYBLE_PLXS_CHAR_VALUE_T::charIndex**

Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_PLXS_CHAR_VALUE_T::value**

Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_PLXS_CHAR_VALUE_T::gattErrorCode**

GATT error code for access control

**struct CYBLE_PLXS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_PLXS_CHAR_INDEX_T charIndex
- CYBLE_PLXS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_ERR_CODE_T gattErrorCode
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_PLXS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_PLXS_CHAR_INDEX_T CYBLE_PLXS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_PLXS_DESCR_INDEX_T CYBLE_PLXS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_ERR_CODE_T CYBLE_PLXS_DESCR_VALUE_T::gattErrorCode**
Error code received from application (optional)

**CYBLE_GATT_VALUE_T* CYBLE_PLXS_DESCR_VALUE_T::value**
Characteristic value

## Macro Definition Documentation

**#define CYBLE_PLXS_DSS_EDU_BIT (0x01u << 0u)**
"Device and Sensor Status" field bits Extended Display Update Ongoing bit

**#define CYBLE_PLXS_DSS_EMD_BIT (0x01u << 1u)**
Equipment Malfunction Detected bit

**#define CYBLE_PLXS_DSS_SPID_BIT (0x01u << 2u)**
Signal Processing Irregularity Detected bit

**#define CYBLE_PLXS_DSS_ISD_BIT (0x01u << 3u)**
Inadequite Signal Detected bit

**#define CYBLE_PLXS_DSS_PSD_BIT (0x01u << 4u)**
Poor Signal Detected bit

**#define CYBLE_PLXS_DSS_LPD_BIT (0x01u << 5u)**
Low Perfusion Detected bit

**#define CYBLE_PLXS_DSS_ESD_BIT (0x01u << 6u)**
Erratic Signal Detected bit

**#define CYBLE_PLXS_DSS_NSD_BIT (0x01u << 7u)**
Nonpulsatile Signal Detected bit

**#define CYBLE_PLXS_DSS_QPD_BIT (0x01u << 8u)**
Questionable Pulse Detected bit

**#define CYBLE_PLXS_DSS_SA_BIT (0x01u << 9u)**
Signal Analysis Ongoing bit

**#define CYBLE_PLXS_DSS_SID_BIT (0x01u << 10u)**
Sensor Interface Detected bit

**#define CYBLE_PLXS_DSS_SUTU_BIT (0x01u << 11u)**
Sensor Unconnected to User bit

**#define CYBLE_PLXS_DSS_USC_BIT (0x01u << 12u)**
Unknown Sensor Connected bit

**#define CYBLE_PLXS_DSS_SD_BIT (0x01u << 13u)**
Sensor Displaced bit

**#define CYBLE_PLXS_DSS_SM_BIT (0x01u << 14u)**
Sensor Malfunctioning bit

**#define CYBLE_PLXS_DSS_SDISC_BIT (0x01u << 15u)**
Sensor Disconnected bit

**#define CYBLE_PLXS_MS_MEAS_BIT (0x01u << 5u)**
"Measurement Status" field bits Measurement Ongoing bit

**#define CYBLE_PLXS_MS_EED_BIT (0x01u << 6u)**
Early Estimated Data bit

**#define CYBLE_PLXS_MS_VDATA_BIT (0x01u << 7u)**
Validated Data bit

**#define CYBLE_PLXS_MS_FQDATA_BIT (0x01u << 8u)**
Fully Qualified Data bit

**#define CYBLE_PLXS_MS_DFMS_BIT (0x01u << 9u)**
Data from Measurement Storage bit

**#define CYBLE_PLXS_MS_DFDEMO_BIT (0x01u << 10u)**
Data for Demonstration bit

**#define CYBLE_PLXS_MS_DFTEST_BIT (0x01u << 11u)**
Data for Testing bit

**#define CYBLE_PLXS_MS_CALIB_BIT (0x01u << 12u)**
Calibration Ongoing bit

**#define CYBLE_PLXS_MS_MUN_BIT (0x01u << 13u)**
Measurement Unavailable bit

**#define CYBLE_PLXS_MS_QMD_BIT (0x01u << 14u)**
Questionable Measurement Detected bit

**#define CYBLE_PLXS_MS_IMD_BIT (0x01u << 15u)**
Invalid Measurement Detected bit

**#define CYBLE_PLXS_SCMT_FLAG_TMSF_BIT (0x01u << 0u)**
"Flag" field bits of PLX Spot-Check Measurement characteristic Timestamp field bit

**#define CYBLE_PLXS_SCMT_FLAG_MSF_BIT (0x01u << 1u)**
Measurement Status Field bit

**#define CYBLE_PLXS_SCMT_FLAG_DSSF_BIT (0x01u << 2u)**
Device and Sensor Status Field bit

**#define CYBLE_PLXS_SCMT_FLAG_PAIF_BIT (0x01u << 3u)**
Pulse Amplitude Index field bit

**#define CYBLE_PLXS_SCMT_FLAG_DEVCLK_BIT (0x01u << 4u)**
Device Clock is Not Set bit

**#define CYBLE_PLXS_CTMT_FLAG_FAST_BIT (0x01u << 0u)**
"Flag" field bits of PLX Continuous Measurement characteristic SpO2PR-Fast field bit

**#define CYBLE_PLXS_CTMT_FLAG_SLOW_BIT (0x01u << 1u)**
SpO2PR-Slow field bit

**#define CYBLE_PLXS_CTMT_FLAG_MSF_BIT (0x01u << 2u)**
Measurement Status field bit

**#define CYBLE_PLXS_CTMT_FLAG_DSSF_BIT (0x01u << 3u)**
Device and Sensor Status field bit

**#define CYBLE_PLXS_CTMT_FLAG_PAIF_BIT (0x01u << 4u)**
Pulse Amplitude Index field bit

**#define CYBLE_PLXS_FEAT_SUPPORT_MEAS_BIT (0x01u << 0u)**
"Supported Features" bits of PLX Features characteristic Measurement Status support bit

**#define CYBLE_PLXS_FEAT_SUPPORT_DSS_BIT (0x01u << 1u)**
Device and Sensor Status support bit

**#define CYBLE_PLXS_FEAT_SUPPORT_MSSC_BIT (0x01u << 2u)**
Measurement Storage for Spot-check measurements bit

**#define CYBLE_PLXS_FEAT_SUPPORT_TMSF_BIT (0x01u << 3u)**
Timestamp for Spot-check measurements bit

**#define CYBLE_PLXS_FEAT_SUPPORT_FAST_BIT (0x01u << 4u)**

SpO2PR-Fast metric bit

**#define CYBLE_PLXS_FEAT_SUPPORT_SLOW_BIT (0x01u << 5u)**

SpO2PR-Slow metric bit

**#define CYBLE_PLXS_FEAT_SUPPORT_PAI_BIT (0x01u << 6u)**

Pulse Amplitude Index field bit

**#define CYBLE_PLXS_FEAT_SUPPORT_MBS_BIT (0x01u << 7u)**

Multiple Bonds Supported bit

## Enumeration Type Documentation

**enum CYBLE_PLXS_CHAR_INDEX_T**

PLXS Characteristic indexes

**Enumerator**

***CYBLE_PLXS_SCMT*** The PLX Spot-check Measurement characteristic, if supported, is used to send Spot-check measurements of SpO2 (Percent oxygen saturation of hemoglobin) and PR (pulse rate). This characteristic is a variable length structure containing the Flags field, the SpO2PR-Spot-Check field, and depending on the contents of the Flags field, the Timestamp field, the Measurement Status field, the Device and Sensor Status field, and/or the Pulse Amplitude Index field.

***CYBLE_PLXS_CTMT*** The PLX Continuous Measurement characteristic, if supported, is used to send periodic pulse oximetry measurements. This characteristic is a variable length structure containing the Flags field (to indicate presence of optional fields), the SpO2PR-Normal field, and depending on the contents of the Flags field, the SpO2PR-Fast field, the SpO2PR-Slow field, the Measurement Status field, the Device and Sensor Status field, and/or the Pulse Amplitude Index field.

***CYBLE_PLXS_FEAT*** The PLX Features characteristic is used to describe the supported features of the Server. Included in the characteristic is a PLX Features field, and, depending on the contents of the PLX Features field, the Measurement Status Support field, and the Device and Sensor Status Support field.

***CYBLE_PLXS_RACP*** This control point is used with a service to provide basic management functionality for the PLX Sensor patient record database. This enables functions including counting records, transmitting records and clearing records based on filter criterion. The filter criterion in the Operand field is defined by the service that references this characteristic as is the format of a record (that may be comprised of one or more characteristics) and the sequence of transferred records.

***CYBLE_PLXS_CHAR_COUNT*** Total count of PLXS characteristics

**enum CYBLE_PLXS_DESCR_INDEX_T**

PLXS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_PLXS_CCCD*** Client Characteristic Configuration Descriptor index

***CYBLE_PLXS_DESCR_COUNT*** Total count of descriptors

**enum CYBLE_PLXS_RACP_OPC_T**

Record Access Control Point characteristic fields defines Opcode of the Record Access Control Point characteristic value type

**Enumerator**

***CYBLE_PLXS_RACP_OPC_RESERVED*** Reserved for future use (Operator:N/A)

***CYBLE_PLXS_RACP_OPC_REPORT_REC*** Report stored records (Operator: Value from Operator Table)

***CYBLE_PLXS_RACP_OPC_DELETE_REC*** Delete stored records (Operator: Value from Operator Table)

***CYBLE_PLXS_RACP_OPC_ABORT_OPN*** Abort operation (Operator: Null 'value of 0x00 from Operator Table')

***CYBLE_PLXS_RACP_OPC_REPORT_NUM_REC*** Report number of stored records (Operator: Value from Operator Table)

***CYBLE_PLXS_RACP_OPC_NUM_REC_RSP*** Number of stored records response (Operator: Null 'value of 0x00 from Operator Table')

***CYBLE_PLXS_RACP_OPC_RSP_CODE*** Response Code (Operator: Null 'value of 0x00 from Operator Table')

**enum CYBLE_PLXS_RACP_OPR_T**

Operator of the Record Access Control Point characteristic value type

**Enumerator**

***CYBLE_PLXS_RACP_OPR_NULL*** Null

***CYBLE_PLXS_RACP_OPR_ALL*** All records

***CYBLE_PLXS_RACP_OPR_LESS*** Less than or equal to

***CYBLE_PLXS_RACP_OPR_GREAT*** Greater than or equal to

***CYBLE_PLXS_RACP_OPR_WITHIN*** Within range of (inclusive)

***CYBLE_PLXS_RACP_OPR_FIRST*** First record(i.e. oldest record)

***CYBLE_PLXS_RACP_OPR_LAST*** Last record (i.e. most recent record)

**enum CYBLE_PLXS_RACP_OPD_T**

Operand of the Record Access Control Point characteristic value type

**Enumerator**

***CYBLE_PLXS_RACP_OPD_NA*** N/A

***CYBLE_PLXS_RACP_OPD_1*** Filter parameters (as appropriate to Operator and Service)

***CYBLE_PLXS_RACP_OPD_2*** Filter parameters (as appropriate to Operator and Service)

***CYBLE_PLXS_RACP_OPD_NO_INCL*** Not included

***CYBLE_PLXS_RACP_OPD_4*** Filter parameters (as appropriate to Operator and Service)

***CYBLE_PLXS_RACP_OPD_NUM_REC*** Number of Records (Field size defined per service)

***CYBLE_PLXS_RACP_OPD_RSP*** Request Op Code, Response Code Value

**enum CYBLE_PLXS_RACP_RSP_T**

Operand Response Code Values of the Record Access Control Point characteristic value type

**Enumerator**

***CYBLE_PLXS_RACP_RSP_NA*** N/A

***CYBLE_PLXS_RACP_RSP_SUCCESS*** Normal response for successful operation

***CYBLE_PLXS_RACP_RSP_UNSPRT_OPC*** Normal response if unsupported Op Code is received

***CYBLE_PLXS_RACP_RSP_INV_OPR*** Normal response if Operator received does not meet the requirements of the service (e.g. Null was expected)

***CYBLE_PLXS_RACP_RSP_UNSPRT_OPR*** Normal response if unsupported Operator is received

***CYBLE_PLXS_RACP_RSP_INV_OPD*** Normal response if Operand received does not meet the requirements of the service

***CYBLE_PLXS_RACP_RSP_NO_REC*** Normal response if request to report stored records or request to delete stored records resulted in no records meeting criteria.

***CYBLE_PLXS_RACP_RSP_UNSUCCESS*** Normal response if request for Abort cannot be completed

***CYBLE_PLXS_RACP_RSP_NO_COMPL*** Normal response if request for Abort cannot be completed
***CYBLE_PLXS_RACP_RSP_UNSPRT_OPD*** Normal response if unsupported Operand is received

# Running Speed and Cadence Service (RSCS)

## Description

The Running Speed and Cadence (RSC) Service exposes speed, cadence and other data related to fitness applications such as the stride length and the total distance the user has travelled while using the Running Speed and Cadence sensor (Server).

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The RSCS API names begin with CyBle_Rscs. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- RSCS Server and Client Functions
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- RSCS Server Functions
  *APIs unique to RSCS designs configured as a GATT Server role.*
- RSCS Client Functions
  *APIs unique to RSCS designs configured as a GATT Client role.*
- RSCS Definitions and Data Structures
  *Contains the RSCS specific definitions and data structures used in the RSCS APIs.*

# RSCS Server and Client Functions

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Rscs

## Functions

- void CyBle_RscsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_RscsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for Running Speed and Cadence Service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for RSCS is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback. |

| | |
|---|---|
| | • eventParam contains the parameters corresponding to the current event. |

## RSCS Server Functions

### Description

APIs unique to RSCS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Rscss

### Functions

- CYBLE_API_RESULT_T CyBle_RscssSetCharacteristicValue (CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscssGetCharacteristicValue (CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscssGetCharacteristicDescriptor (CYBLE_RSCS_CHAR_INDEX_T charIndex, CYBLE_RSCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_RscssSetCharacteristicValue (CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic value of the Running Speed and Cadence Service in the local GATT database. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. Valid values are,<br>• CYBLE_RSCS_RSC_FEATURE<br>• CYBLE_RSCS_SENSOR_LOCATION. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_RscssGetCharacteristicValue (CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic value of the Running Speed and Cadence Service from the GATT database. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic

**CYBLE_API_RESULT_T CyBle_RscssGetCharacteristicDescriptor (CYBLE_RSCS_CHAR_INDEX_T *charIndex*, CYBLE_RSCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the characteristic descriptor of a specified characteristic of the Running Speed and Cadence Service from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic. Valid values are,<br>• CYBLE_RSCS_RSC_MEASUREMENT<br>• CYBLE_RSCS_SC_CONTROL_POINT |
| *descrIndex* | The index of a service characteristic descriptor. Valid value is,<br>• CYBLE_RSCS_CCCD |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor

**CYBLE_API_RESULT_T CyBle_RscssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification with the characteristic value to the Client device. This is specified by charIndex of the Running Speed and Cadence Service.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_RSCSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. Valid value is,<br>• CYBLE_RSCS_RSC_MEASUREMENT. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter is failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

**CYBLE_API_RESULT_T CyBle_RscssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value to the Client device. This is specified by charIndex of the Running Speed and Cadence Service.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_RSCSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter is failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RSCS service-specific callback is registered (with CyBle_RscsRegisterAttrCallback):

- CYBLE_EVT_RSCSS_INDICATION_CONFIRMED - In case if the indication is successfully delivered to the peer device.

Otherwise (if the RSCS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - In case if the indication is successfully delivered to the peer device.

# RSCS Client Functions

## Description

APIs unique to RSCS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Rscsc

## Functions

- CYBLE_API_RESULT_T CyBle_RscscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_RscscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex, CYBLE_RSCS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RscscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RSCS_CHAR_INDEX_T charIndex, uint8 descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_RscscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_RSCSS_CHAR_WRITE event is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | Size of the characteristic value attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RSCS service-specific callback is registered (with CyBle_RscsRegisterAttrCallback):

- CYBLE_EVT_RSCSC_WRITE_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_RSCS_CHAR_VALUE_T.

Otherwise (if the RSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_RscscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*)**

Sends a request to the peer device to set the characteristic value of the Running Speed and Cadence Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RSCS service-specific callback is registered (with CyBle_RscsRegisterAttrCallback):

- CYBLE_EVT_RSCSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_RSCS_CHAR_VALUE_T.

Otherwise (if the RSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_RscscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*, CYBLE_RSCS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)

Sends a request to the peer device to get the characteristic descriptor of the specified characteristic of the Running Speed and Cadence Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_RSCSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_RSCSS_NOTIFICATION_DISABLED;
- CYBLE_EVT_RSCSS_INDICATION_ENABLED;
- CYBLE_EVT_RSCSS_INDICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a RSCS characteristic. |
| *descrIndex* | The index of a RSCS characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RSCS service-specific callback is registered (with CyBle_RscsRegisterAttrCallback):

- CYBLE_EVT_RSCSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_RSCS_DESCR_VALUE_T.

Otherwise (if the RSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_RscscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RSCS_CHAR_INDEX_T *charIndex*, uint8 *descrIndex*)

Sends a request to the peer device to get characteristic descriptor of the specified characteristic of the Running Speed and Cadence Service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a Service Characteristic. |
| *descrIndex* | The index of a Service Characteristic Descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_INVALID_OPERATION - Cannot process a request to send PDU due to invalid operation performed by the application.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Peer device doesn't have a particular descriptor.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RSCS service-specific callback is registered (with CyBle_RscsRegisterAttrCallback):

- CYBLE_EVT_RSCSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_RSCS_DESCR_VALUE_T.

Otherwise (if the RSCS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# RSCS Definitions and Data Structures

## Description

Contains the RSCS specific definitions and data structures used in the RSCS APIs.

## Data Structures

- struct CYBLE_RSCS_CHAR_VALUE_T
- struct CYBLE_RSCS_DESCR_VALUE_T
- struct CYBLE_RSCSS_CHAR_T
- struct CYBLE_RSCSS_T
- struct CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T
- struct CYBLE_RSCSC_T

## Enumerations

- enum CYBLE_RSCS_CHAR_INDEX_T
- enum CYBLE_RSCS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_RSCS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_RSCS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_RSCS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_RSCS_CHAR_INDEX_T CYBLE_RSCS_CHAR_VALUE_T::charIndex**
Index of Running Speed and Cadence Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_RSCS_CHAR_VALUE_T::value**
Characteristic value

### struct CYBLE_RSCS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_RSCS_CHAR_INDEX_T charIndex
- CYBLE_RSCS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_RSCS_DESCR_VALUE_T::connHandle**
Connection handle

**CYBLE_RSCS_CHAR_INDEX_T CYBLE_RSCS_DESCR_VALUE_T::charIndex**
Characteristic index of the Service

**CYBLE_RSCS_DESCR_INDEX_T CYBLE_RSCS_DESCR_VALUE_T::descrIndex**
Characteristic index Descriptor the Service

**CYBLE_GATT_VALUE_T* CYBLE_RSCS_DESCR_VALUE_T::value**
Pointer to value of the Service Characteristic Descriptor

**struct CYBLE_RSCSS_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_RSCS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RSCSS_CHAR_T::charHandle**
Handle of the characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_RSCSS_CHAR_T::descrHandle[CYBLE_RSCS_DESCR_COUNT]**
Handle of the descriptor

**struct CYBLE_RSCSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_RSCSS_CHAR_T charInfo [CYBLE_RSCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RSCSS_T::serviceHandle**
Running Speed and Cadence Service handle

**CYBLE_RSCSS_CHAR_T CYBLE_RSCSS_T::charInfo[CYBLE_RSCS_CHAR_COUNT]**
Array of Running Speed and Cadence Service Characteristics + Descriptors handles

**struct CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo
- CYBLE_GATT_DB_ATTR_HANDLE_T descriptors [CYBLE_RSCS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T::charInfo**
Characteristic handle + properties

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T::descriptors[CYBLE_RSCS_DESCR_COUNT]**
Characteristic descriptors handles handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T::endHandle**
End handle of characteristic

**struct CYBLE_RSCSC_T**

**Data Fields**

- CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T characteristics [CYBLE_RSCS_CHAR_COUNT]

**Field Documentation**

**CYBLE_RSCSC_SRVR_FULL_CHAR_INFO_T**
**CYBLE_RSCSC_T::characteristics[CYBLE_RSCS_CHAR_COUNT]**
Characteristics handles array

### Enumeration Type Documentation

**enum CYBLE_RSCS_CHAR_INDEX_T**
RSCS Characteristic indexes
**Enumerator**
***CYBLE_RSCS_RSC_MEASUREMENT*** RSC Measurement Characteristic index
***CYBLE_RSCS_RSC_FEATURE*** RSC Feature Characteristic index
***CYBLE_RSCS_SENSOR_LOCATION*** Sensor Location Characteristic index
***CYBLE_RSCS_SC_CONTROL_POINT*** SC Control Point Characteristic index
***CYBLE_RSCS_CHAR_COUNT*** Total count of RSCS characteristics

**enum CYBLE_RSCS_DESCR_INDEX_T**
RSCS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_RSCS_CCCD*** Client Characteristic Configuration Descriptor index
***CYBLE_RSCS_DESCR_COUNT*** Total count of descriptors

## Reference Time Update Service (RTUS)

### Description

The Reference Time Update Service enables a Bluetooth device that can update the system time using the reference time such as a GPS receiver to expose a control point and expose the accuracy (drift) of the local system time compared to the reference time source.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The RTUS API names begin with CyBle_Rtus. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- RTUS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- RTUS Server Functions
  *APIs unique to RTUS designs configured as a GATT Server role.*
- RTUS Client Functions
  *APIs unique to RTUS designs configured as a GATT Client role.*
- RTUS Definitions and Data Structures
  *Contains the RTUS specific definitions and data structures used in the RTUS APIs.*

# RTUS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Rtus

## Functions

- void CyBle_RtusRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_RtusRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for Reference Time Update Service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for RTUS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

# RTUS Server Functions

## Description

APIs unique to RTUS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Rtuss

## Functions

- CYBLE_API_RESULT_T CyBle_RtussSetCharacteristicValue (CYBLE_RTUS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RtussGetCharacteristicValue (CYBLE_RTUS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

### CYBLE_API_RESULT_T CyBle_RtussSetCharacteristicValue (CYBLE_RTUS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)

Sets characteristic value of the Reference Time Update Service, which is identified by charIndex in the local database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_RTUS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request is handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameters failed

**CYBLE_API_RESULT_T CyBle_RtussGetCharacteristicValue (CYBLE_RTUS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Reference Time Update Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_RTUS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request is handled successfully;
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameter failed.

# RTUS Client Functions

## Description

APIs unique to RTUS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Rtusc

## Functions

- CYBLE_API_RESULT_T CyBle_RtuscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RTUS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_RtuscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_RTUS_CHAR_INDEX_T charIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_RtuscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RTUS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_RTUSS_WRITE_CHAR_CMD event is generated.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |
| *attrSize* | Size of the characteristic value attribute. |
| *attrValue* | Pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_STATE - Connection with the Client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**CYBLE_API_RESULT_T CyBle_RtuscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_RTUS_CHAR_INDEX_T *charIndex*)**

Sends a request to a peer device to set characteristic value of the Reference Time Update Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - the request was sent successfully;
- CYBLE_ERROR_INVALID_STATE - connection with the Client is not established.
- CYBLE_ERROR_INVALID_PARAMETER - validation of the input parameters failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the RTUS service-specific callback is registered (with CyBle_RtusRegisterAttrCallback):

- CYBLE_EVT_RTUSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_RTUS_CHAR_VALUE_T.

Otherwise (if the RTUS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# RTUS Definitions and Data Structures

## Description

Contains the RTUS specific definitions and data structures used in the RTUS APIs.

## Data Structures

- struct CYBLE_RTUS_CHAR_VALUE_T
- struct CYBLE_RTUS_TIME_UPDATE_STATE_T
- struct CYBLE_RTUSS_T
- struct CYBLE_RTUSC_T

## Enumerations

- enum CYBLE_RTUS_CHAR_INDEX_T

## Data Structure Documentation

### struct CYBLE_RTUS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_RTUS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_RTUS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_RTUS_CHAR_INDEX_T CYBLE_RTUS_CHAR_VALUE_T::charIndex**
Index of Reference Time Update Service Characteristic

**CYBLE_GATT_VALUE_T* CYBLE_RTUS_CHAR_VALUE_T::value**
Characteristic value

### struct CYBLE_RTUS_TIME_UPDATE_STATE_T

**Data Fields**

- uint8 currentState
- uint8 result

**Field Documentation**

**uint8 CYBLE_RTUS_TIME_UPDATE_STATE_T::currentState**
Current state

**uint8 CYBLE_RTUS_TIME_UPDATE_STATE_T::result**
Result of Time update

### struct CYBLE_RTUSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T timeUpdateCpHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T timeUpdateStateHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RTUSS_T::serviceHandle**
Handle of the Reference Time Update Service

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RTUSS_T::timeUpdateCpHandle**
Handle of the Time Update Control Point Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_RTUSS_T::timeUpdateStateHandle**
Handle of the Time Update State Characteristic

### struct CYBLE_RTUSC_T

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T charInfo [CYBLE_RTUS_CHAR_COUNT]

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_RTUSC_T::charInfo[CYBLE_RTUS_CHAR_COUNT]**

Characteristic handle and properties

### Enumeration Type Documentation

**enum CYBLE_RTUS_CHAR_INDEX_T**

Characteristic indexes

**Enumerator**

***CYBLE_RTUS_TIME_UPDATE_CONTROL_POINT*** Time Update Control Point Characteristic index

***CYBLE_RTUS_TIME_UPDATE_STATE*** Time Update State Characteristic index

***CYBLE_RTUS_CHAR_COUNT*** Total count of RTUS characteristics

## Scan Parameters Service (ScPS)

### Description

The Scan Parameters Service enables a Server device to expose a Characteristic for the GATT Client to write its scan interval and scan window on the Server device, and enables a Server to request a refresh of the GATT Client scan interval and scan window.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The ScPS API names begin with CyBle_Scps. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- ScPS Server and Client Functions
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- ScPS Server Functions
  *APIs unique to ScPS designs configured as a GATT Server role.*
- ScPS Client Functions
  *APIs unique to ScPS designs configured as a GATT Client role.*
- ScPS Definitions and Data Structures
  *Contains the ScPS specific definitions and data structures used in the ScPS APIs.*

## ScPS Server and Client Functions

### Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Scps

### Functions

- void CyBle_ScpsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

### Function Documentation

**void CyBle_ScpsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for ScPS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

## ScPS Server Functions

### Description

APIs unique to ScPS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Scpss

### Functions

- CYBLE_API_RESULT_T CyBle_ScpssSetCharacteristicValue (CYBLE_SCPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_ScpssGetCharacteristicValue (CYBLE_SCPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_ScpssGetCharacteristicDescriptor (CYBLE_SCPS_CHAR_INDEX_T charIndex, CYBLE_SCPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_ScpssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_SCPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

### Function Documentation

**CYBLE_API_RESULT_T CyBle_ScpssSetCharacteristicValue (CYBLE_SCPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the Scan Parameters service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic.<br>• CYBLE_SCPS_SCAN_INT_WIN - The Scan Interval Window characteristic index.<br>• CYBLE_SCPS_SCAN_REFRESH - The Scan Refresh characteristic index |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_ScpssGetCharacteristicValue (CYBLE_SCPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic value of the Scan Parameters service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic.<br>• CYBLE_SCPS_SCAN_INT_WIN - The Scan Interval Window characteristic index.<br>• CYBLE_SCPS_SCAN_REFRESH - The Scan Refresh characteristic index |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent.

**CYBLE_API_RESULT_T CyBle_ScpssGetCharacteristicDescriptor (CYBLE_SCPS_CHAR_INDEX_T *charIndex*, CYBLE_SCPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of the specified characteristic of the Scan Parameters service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic.<br>• CYBLE_SCPS_SCAN_REFRESH - The Scan Refresh characteristic index. |
| *descrIndex* | The index of the descriptor.<br>• CYBLE_SCPS_SCAN_REFRESH_CCCD - The Client Characteristic Configuration descriptor index of the Scan Refresh characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where the characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_ScpssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_SCPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function notifies the client that the server requires the Scan Interval Window Characteristic to be written with the latest values upon notification.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_SCPSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the characteristic.<br>• CYBLE_SCPS_SCAN_REFRESH - The Scan Refresh characteristic index. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

# ScPS Client Functions

## Description

APIs unique to ScPS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Scpsc

## Functions

- CYBLE_API_RESULT_T CyBle_ScpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_SCPS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_ScpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_SCPS_CHAR_INDEX_T charIndex, CYBLE_SCPS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_ScpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_SCPS_CHAR_INDEX_T charIndex, CYBLE_SCPS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_ScpscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_SCPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the Scan Parameters Service, which is identified by charIndex.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_GATTC_WRITE_RSP;
- CYBLE_EVT_GATTC_ERROR_RSP.

The CYBLE_EVT_SCPSS_SCAN_INT_WIN_CHAR_WRITE event is received by the peer device on invoking this function.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**CYBLE_API_RESULT_T CyBle_ScpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_SCPS_CHAR_INDEX_T *charIndex*, CYBLE_SCPS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets characteristic descriptor of specified characteristic of the Scan Parameters Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_SCPSS_NOTIFICATION_ENABLED;
- CYBLE_EVT_SCPSS_NOTIFICATION_DISABLED.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the SCPS service-specific callback is registered (with CyBle_ScpsRegisterAttrCallback):

- CYBLE_EVT_SCPSC_WRITE_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_SCPS_DESCR_VALUE_T.

Otherwise (if the SCPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - In case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_ScpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_SCPS_CHAR_INDEX_T *charIndex*, CYBLE_SCPS_DESCR_INDEX_T *descrIndex*)**

Gets characteristic descriptor of specified characteristic of the Scan Parameters Service.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_SCPSC_READ_DESCR_RESPONSE;
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a Service Characteristic. |
| *descrIndex* | The index of a Service Characteristic Descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the SCPS service-specific callback is registered (with CyBle_ScpsRegisterAttrCallback):

- CYBLE_EVT_SCPSC_READ_DESCR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_SCPS_DESCR_VALUE_T.

Otherwise (if the SCPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# ScPS Definitions and Data Structures

## Description

Contains the ScPS specific definitions and data structures used in the ScPS APIs.

## Data Structures

- struct CYBLE_SCPSS_T

- struct CYBLE_SCPSC_T
- struct CYBLE_SCPS_CHAR_VALUE_T
- struct CYBLE_SCPS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_SCPS_CHAR_INDEX_T
- enum CYBLE_SCPS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_SCPSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T intervalWindowCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T refreshCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T refreshCccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SCPSS_T::serviceHandle**
Scan Parameter Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SCPSS_T::intervalWindowCharHandle**
Handle of Scan Interval Window Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SCPSS_T::refreshCharHandle**
Handle of Scan Refresh Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SCPSS_T::refreshCccdHandle**
Handle of Client Characteristic Configuration Descriptor

### struct CYBLE_SCPSC_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_SRVR_CHAR_INFO_T intervalWindowChar
- CYBLE_SRVR_CHAR_INFO_T refreshChar
- CYBLE_GATT_DB_ATTR_HANDLE_T refreshCccdHandle

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_SCPSC_T::connHandle**
Peer device handle

**CYBLE_SRVR_CHAR_INFO_T CYBLE_SCPSC_T::intervalWindowChar**
Handle + properties of Scan Interval Window Characteristic

**CYBLE_SRVR_CHAR_INFO_T CYBLE_SCPSC_T::refreshChar**
Handle + properties of Scan Refresh Characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_SCPSC_T::refreshCccdHandle**
Handle of Client Characteristic Configuration Descriptor

### struct CYBLE_SCPS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_SCPS_CHAR_INDEX_T charIndex

- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_SCPS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_SCPS_CHAR_INDEX_T CYBLE_SCPS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_SCPS_CHAR_VALUE_T::value**
Characteristic value

**struct CYBLE_SCPS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_SCPS_CHAR_INDEX_T charIndex
- CYBLE_SCPS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_SCPS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_SCPS_CHAR_INDEX_T CYBLE_SCPS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_SCPS_DESCR_INDEX_T CYBLE_SCPS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_SCPS_DESCR_VALUE_T::value**
Descriptor value

## Enumeration Type Documentation

**enum CYBLE_SCPS_CHAR_INDEX_T**
ScPS Characteristic indexes
**Enumerator**
***CYBLE_SCPS_SCAN_INT_WIN*** Scan Interval Window characteristic index
***CYBLE_SCPS_SCAN_REFRESH*** Scan Refresh characteristic index
***CYBLE_SCPS_CHAR_COUNT*** Total count of characteristics

**enum CYBLE_SCPS_DESCR_INDEX_T**
ScPS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_SCPS_SCAN_REFRESH_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_SCPS_DESCR_COUNT*** Total count of descriptors

# TX Power Service (TPS)

## Description

The Tx Power Service uses the Tx Power Level Characteristic to expose the current transmit power level of a device when in a connection.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The TPS API names begin with CyBle_Tps. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- TPS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- TPS Server Functions
  *APIs unique to TPS designs configured as a GATT Server role.*
- TPS Client Functions
  *APIs unique to TPS designs configured as a GATT Client role.*
- TPS Definitions and Data Structures
  *Contains the TPS specific definitions and data structures used in the TPS APIs.*

# TPS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Tps

## Functions

- void CyBle_TpsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_TpsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for TPS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# TPS Server Functions

## Description

APIs unique to TPS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Tpss

## Functions

- CYBLE_API_RESULT_T CyBle_TpssSetCharacteristicValue (CYBLE_TPS_CHAR_INDEX_T charIndex, uint8 attrSize, int8 *attrValue)
- CYBLE_API_RESULT_T CyBle_TpssGetCharacteristicValue (CYBLE_TPS_CHAR_INDEX_T charIndex, uint8 attrSize, int8 *attrValue)
- CYBLE_API_RESULT_T CyBle_TpssGetCharacteristicDescriptor (CYBLE_TPS_CHAR_INDEX_T charIndex, CYBLE_TPS_CHAR_DESCRIPTORS_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_TpssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_TPS_CHAR_INDEX_T charIndex, uint8 attrSize, int8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_TpssSetCharacteristicValue (CYBLE_TPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, int8 **attrValue*)**

Sets characteristic value of the Tx Power Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored in the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameters failed.

**CYBLE_API_RESULT_T CyBle_TpssGetCharacteristicValue (CYBLE_TPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, int8 **attrValue*)**

Gets characteristic value of the Tx Power Service, which is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the Tx Power characteristic. |
| *attrSize* | The size of the Tx Power characteristic value attribute. |
| *attrValue* | The pointer to the location where Tx Power characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_TpssGetCharacteristicDescriptor (CYBLE_TPS_CHAR_INDEX_T *charIndex*, CYBLE_TPS_CHAR_DESCRIPTORS_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets characteristic descriptor of specified characteristic of the Tx Power Service.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Characteristic Descriptor value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional descriptor is absent.

**CYBLE_API_RESULT_T CyBle_TpssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_TPS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, int8 **attrValue*)**

Sends a notification with the characteristic value, as specified by charIndex, to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_TPSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this. characteristic.
- CYBLE_ERROR_INVALID_STATE - Connection with client is not established.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

# TPS Client Functions

## Description

APIs unique to TPS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Tpsc

## Functions

- CYBLE_API_RESULT_T CyBle_TpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_TPS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_TpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_TPS_CHAR_INDEX_T charIndex, CYBLE_TPS_CHAR_DESCRIPTORS_T descrIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_TpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_TPS_CHAR_INDEX_T charIndex, CYBLE_TPS_CHAR_DESCRIPTORS_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_TpscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_TPS_CHAR_INDEX_T *charIndex*)**

Gets the characteristic value of the Tx Power Service, which is identified by charIndex.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_TPSC_READ_CHAR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the characteristic. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the TPS service-specific callback is registered (with CyBle_TpsRegisterAttrCallback):

- CYBLE_EVT_TPSC_READ_CHAR_RESPONSE - In case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_TPS_CHAR_VALUE_T.

Otherwise (if the TPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - In case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - In case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_TpscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_TPS_CHAR_INDEX_T *charIndex*, CYBLE_TPS_CHAR_DESCRIPTORS_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic descriptor value of the Tx Power Service.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_TPSS_NOTIFICATION_ENABLED
- CYBLE_EVT_TPSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the Characteristic |
| *descrIndex* | The index of the TX Power Service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |

| | |
|---|---|
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the TPS service-specific callback is registered (with CyBle_TpsRegisterAttrCallback):

- CYBLE_EVT_TPSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_TPS_DESCR_VALUE_T.

Otherwise (if the TPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_TpscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_TPS_CHAR_INDEX_T *charIndex*, CYBLE_TPS_CHAR_DESCRIPTORS_T *descrIndex*)**

Gets a characteristic descriptor of the Tx Power Service.

This function call can result in generation of the following events based on the response from the server device:

- CYBLE_EVT_TPSC_READ_DESCR_RESPONSE.
- CYBLE_EVT_GATTC_ERROR_RSP.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the characteristic descriptor. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The component in in invalid state for current operation.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - Cannot process request to send PDU due to invalid operation performed by the application.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the TPS service-specific callback is registered (with CyBle_TpsRegisterAttrCallback):

- CYBLE_EVT_TPSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_TPS_DESCR_VALUE_T.

Otherwise (if the TPS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# TPS Definitions and Data Structures

## Description

Contains the TPS specific definitions and data structures used in the TPS APIs.

## Data Structures

- struct CYBLE_TPS_CHAR_VALUE_T
- struct CYBLE_TPS_DESCR_VALUE_T
- struct CYBLE_TPSS_T
- struct CYBLE_TPSC_T

## Enumerations

- enum CYBLE_TPS_CHAR_INDEX_T
- enum CYBLE_TPS_CHAR_DESCRIPTORS_T

## Data Structure Documentation

### struct CYBLE_TPS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_TPS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_TPS_CHAR_VALUE_T::connHandle**

Connection handle

**CYBLE_TPS_CHAR_INDEX_T CYBLE_TPS_CHAR_VALUE_T::charIndex**

Characteristic index of Tx Power Service

**CYBLE_GATT_VALUE_T* CYBLE_TPS_CHAR_VALUE_T::value**

Pointer to value of Tx Power Service characteristic

### struct CYBLE_TPS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_TPS_CHAR_INDEX_T charIndex
- CYBLE_TPS_CHAR_DESCRIPTORS_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_TPS_DESCR_VALUE_T::connHandle**
Connection handle

**CYBLE_TPS_CHAR_INDEX_T CYBLE_TPS_DESCR_VALUE_T::charIndex**
Characteristic index of Tx Power Service

**CYBLE_TPS_CHAR_DESCRIPTORS_T CYBLE_TPS_DESCR_VALUE_T::descrIndex**
Characteristic index Descriptor of Tx Power Service

**CYBLE_GATT_VALUE_T* CYBLE_TPS_DESCR_VALUE_T::value**
Pointer to value of Tx Power Service characteristic

**struct CYBLE_TPSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T txPowerLevelCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T txPowerLevelCccdHandle

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_TPSS_T::serviceHandle**
Tx Power Service handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_TPSS_T::txPowerLevelCharHandle**
Tx Power Level Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_TPSS_T::txPowerLevelCccdHandle**
Tx Power Level Client Characteristic Configuration Descriptor handle

**struct CYBLE_TPSC_T**

**Data Fields**

- CYBLE_SRVR_CHAR_INFO_T txPowerLevelChar
- CYBLE_GATT_DB_ATTR_HANDLE_T txPowerLevelCccdHandle

**Field Documentation**

**CYBLE_SRVR_CHAR_INFO_T CYBLE_TPSC_T::txPowerLevelChar**
Tx Power Level Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_TPSC_T::txPowerLevelCccdHandle**
Tx Power Level Client Characteristic Configuration Descriptor handle

## Enumeration Type Documentation

**enum CYBLE_TPS_CHAR_INDEX_T**
TPS Characteristic indexes
**Enumerator**
***CYBLE_TPS_TX_POWER_LEVEL*** Tx Power Level characteristic index
***CYBLE_TPS_CHAR_COUNT*** Total count of characteristics

**enum CYBLE_TPS_CHAR_DESCRIPTORS_T**
TPS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_TPS_CCCD*** Tx Power Level Client Characteristic configuration descriptor index

***CYBLE_TPS_DESCR_COUNT*** Total count of Tx Power Service characteristic descriptors

# User Data Service (UDS)

## Description

The User Data Service exposes user-related data in the sports and fitness environment. This allows remote access and update of user data by a Client as well as the synchronization of user data between a Server and a Client.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The UDS API names begin with CyBle_Uds. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- UDS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- UDS Server Functions
  *APIs unique to UDS designs configured as a GATT Server role.*
- UDS Client Functions
  *APIs unique to UDS designs configured as a GATT Client role.*
- UDS Definitions and Data Structures
  *Contains the UDS specific definitions and data structures used in the UDS APIs.*

# UDS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.

No letter is appended to the API name: CyBle_Uds

## Functions

- void CyBle_UdsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

**void CyBle_UdsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)**

Registers a callback function for service-specific attribute operations. Service-specific write requests from a peer device will not be handled with an unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T for UDS is: typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam), where:<br>• eventCode indicates the event that triggered this callback.<br>• eventParam contains the parameters corresponding to the current event. |

**Side Effects**

The *eventParams in the callback function should not be used by the application once the callback function execution is finished. Otherwise this data may become corrupted.

# UDS Server Functions

## Description

APIs unique to UDS designs configured as a GATT Server role.

A letter 's' is appended to the API name: CyBle_Udss

## Functions

- CYBLE_API_RESULT_T CyBle_UdssSetCharacteristicValue (CYBLE_UDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdssGetCharacteristicValue (CYBLE_UDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdssGetCharacteristicDescriptor (CYBLE_UDS_CHAR_INDEX_T charIndex, CYBLE_UDS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_UdssSetCharacteristicValue (CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent

**CYBLE_API_RESULT_T CyBle_UdssGetCharacteristicValue (CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets the value of the characteristic, as identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - A characteristic is absent.

**CYBLE_API_RESULT_T CyBle_UdssGetCharacteristicDescriptor (CYBLE_UDS_CHAR_INDEX_T *charIndex*, CYBLE_UDS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Gets a characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - Characteristic Descriptor value was read successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - A characteristic is absent.

**CYBLE_API_RESULT_T CyBle_UdssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends a notification of the specified characteristic value, as identified by the charIndex.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_UDSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.

**CYBLE_API_RESULT_T CyBle_UdssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication of the specified characteristic value, as identified by the charIndex.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_UDSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle which consist of the device ID and ATT connection ID. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - An optional characteristic is absent.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the client.
- CYBLE_ERROR_IND_DISABLED - Indication is disabled for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSS_INDICATION_CONFIRMED - If the indication is successfully delivered to the peer device.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - If the indication is successfully delivered to the peer device.

# UDS Client Functions

## Description

APIs unique to UDS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Udsc

## Functions

- CYBLE_API_RESULT_T CyBle_UdscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_UdscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, uint16 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, CYBLE_UDS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_UdscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_UDS_CHAR_INDEX_T charIndex, CYBLE_UDS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_UdscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_UDSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSC_WRITE_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, etc.) are provided with an event parameter structure of type CYBLE_UDS_CHAR_VALUE_T.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_EXEC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_UdscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*)**

This function is used to read the characteristic Value from a server, as identified by its charIndex. As a result a Read Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_UDSS_READ_CHAR events is generated. On successful request execution on the Server side the Read Response is sent to the Client.

The Read Response only contains the characteristic Value that is less than or equal to (MTU - 1) octets in length. If the characteristic Value is greater than (MTU - 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the characteristic Value is required.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_UDS_CHAR_VALUE_T.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_UdscGetLongCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, uint16 *attrSize*, uint8 **attrValue*)

Sends a request to read a long characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the buffer where the read long characteristic descriptor value should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_UDS_CHAR_VALUE_T.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_BLOB_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).

- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_UdscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, CYBLE_UDS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic Value to the server, as identified by its charIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_UDSS_INDICATION_ENABLED
- CYBLE_EVT_UDSS_INDICATION_DISABLED
- CYBLE_EVT_UDSS_NOTIFICATION_ENABLED
- CYBLE_EVT_UDSS_NOTIFICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic descriptor value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSC_WRITE_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index etc.) are provided with an event parameter structure of type CYBLE_UDS_DESCR_VALUE_T.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_UdscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_UDS_CHAR_INDEX_T *charIndex*, CYBLE_UDS_DESCR_INDEX_T *descrIndex*)**

Gets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

| | |
|---|---|
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular descriptor.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In the case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the UDS service-specific callback is registered (with CyBle_UdsRegisterAttrCallback):

- CYBLE_EVT_UDSC_READ_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index, value, etc.) are provided with an event parameter structure of type CYBLE_UDS_DESCR_VALUE_T.

Otherwise (if the UDS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameter structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is some trouble with the requested attribute on the peer device, the details are provided with an event parameter structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# UDS Definitions and Data Structures

## Description

Contains the UDS specific definitions and data structures used in the UDS APIs.

## Data Structures

- struct CYBLE_UDSS_CHAR_T
- struct CYBLE_UDSS_T
- struct CYBLE_UDSC_CHAR_T
- struct CYBLE_UDSC_T
- struct CYBLE_UDS_CHAR_VALUE_T
- struct CYBLE_UDS_DESCR_VALUE_T

## Enumerations

- enum CYBLE_UDS_CHAR_INDEX_T
- enum CYBLE_UDS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_UDSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle

- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_UDS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSS_CHAR_T::charHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSS_CHAR_T::descrHandle[CYBLE_UDS_DESCR_COUNT]**
Handle of descriptor

**struct CYBLE_UDSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_UDSS_CHAR_T charInfo [CYBLE_UDS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSS_T::serviceHandle**
User Data Service handle

**CYBLE_UDSS_CHAR_T CYBLE_UDSS_T::charInfo[CYBLE_UDS_CHAR_COUNT]**
User Data Service characteristics info array

**struct CYBLE_UDSC_CHAR_T**

**Data Fields**

- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_UDS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle

**Field Documentation**

**uint8 CYBLE_UDSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSC_CHAR_T::valueHandle**
Handle of server database attribute value entry

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSC_CHAR_T::descrHandle[CYBLE_UDS_DESCR_COUNT]**
User Data client char. descriptor handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_UDSC_CHAR_T::endHandle**
Characteristic End Handle

**struct CYBLE_UDSC_T**

**Data Fields**

- CYBLE_UDSC_CHAR_T charInfo [CYBLE_UDS_CHAR_COUNT]

**Field Documentation**

**CYBLE_UDSC_CHAR_T CYBLE_UDSC_T::charInfo[CYBLE_UDS_CHAR_COUNT]**
Characteristics handle + properties array

**struct CYBLE_UDS_CHAR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_UDS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

- CYBLE_GATT_ERR_CODE_T gattErrorCode

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_UDS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_UDS_CHAR_INDEX_T CYBLE_UDS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_UDS_CHAR_VALUE_T::value**
Characteristic value

**CYBLE_GATT_ERR_CODE_T CYBLE_UDS_CHAR_VALUE_T::gattErrorCode**
GATT error code for access control

**struct CYBLE_UDS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_UDS_CHAR_INDEX_T charIndex
- CYBLE_UDS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_UDS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_UDS_CHAR_INDEX_T CYBLE_UDS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_UDS_DESCR_INDEX_T CYBLE_UDS_DESCR_VALUE_T::descrIndex**
Index of service characteristic descriptor

**CYBLE_GATT_VALUE_T* CYBLE_UDS_DESCR_VALUE_T::value**
Descriptor value

## Enumeration Type Documentation

**enum CYBLE_UDS_CHAR_INDEX_T**
UDS Service Characteristics indexes

**Enumerator**

***CYBLE_UDS_FNM*** First Name characteristic index
***CYBLE_UDS_LNM*** Last Name characteristic index
***CYBLE_UDS_EML*** Email Address characteristic index
***CYBLE_UDS_AGE*** Age characteristic index
***CYBLE_UDS_DOB*** Date of Birth characteristic index
***CYBLE_UDS_GND*** Gender characteristic index
***CYBLE_UDS_WGT*** Weight characteristic index
***CYBLE_UDS_HGT*** Height characteristic index
***CYBLE_UDS_VO2*** VO2 Max characteristic index
***CYBLE_UDS_HRM*** Heart Rate Max characteristic index
***CYBLE_UDS_RHR*** Resting Heart Rate characteristic index
***CYBLE_UDS_MRH*** Maximum Recommended Heart Rate characteristic index

***CYBLE_UDS_AET*** Aerobic Threshold characteristic index
***CYBLE_UDS_ANT*** Anaerobic Threshold characteristic index
***CYBLE_UDS_STP*** Sport Type for Aerobic and Anaerobic Thresholds characteristic index
***CYBLE_UDS_DTA*** Date of Threshold Assessment characteristic index
***CYBLE_UDS_WCC*** Waist Circumference characteristic index
***CYBLE_UDS_HCC*** Hip Circumference characteristic index
***CYBLE_UDS_FBL*** Fat Burn Heart Rate Lower Limit characteristic index
***CYBLE_UDS_FBU*** Fat Burn Heart Rate Upper Limit characteristic index
***CYBLE_UDS_AEL*** Aerobic Heart Rate Lower Limit characteristic index
***CYBLE_UDS_AEU*** Aerobic Heart Rate Upper Limit characteristic index
***CYBLE_UDS_ANL*** Anaerobic Heart Rate Lower Limit characteristic index
***CYBLE_UDS_ANU*** Anaerobic Heart Rate Upper Limit characteristic index
***CYBLE_UDS_5ZL*** Five Zone Heart Rate Limits characteristic index
***CYBLE_UDS_3ZL*** Three Zone Heart Rate Limits characteristic index
***CYBLE_UDS_2ZL*** Two Zone Heart Rate Limit characteristic index
***CYBLE_UDS_DCI*** Database Change Increment characteristic index
***CYBLE_UDS_UIX*** User Index characteristic index
***CYBLE_UDS_UCP*** User Control Point characteristic index
***CYBLE_UDS_LNG*** Language characteristic index
***CYBLE_UDS_CHAR_COUNT*** Total count of UDS characteristics

**enum CYBLE_UDS_DESCR_INDEX_T**

UDS Service Characteristic Descriptors indexes

**Enumerator**

***CYBLE_UDS_CCCD*** Client Characteristic Configuration descriptor index
***CYBLE_UDS_DESCR_COUNT*** Total count of UDS descriptors

# Wireless Power Transfer Service (WPTS)

## Description

The Wireless Power Transfer Service enables communication between Power Receiver Unit and Power Transmitter Unit in the Wireless Power Transfer systems.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The WPTS API names begin with CyBle_Wpts. In addition to this, the APIs also append the GATT role initial letter in the API name.

## Modules

- WPTS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- WPTS Server Functions
  *APIs unique to WPTS designs configured as a GATT Server role.*
- WPTS Client Functions
  *APIs unique to WPTS designs configured as a GATT Client role.*

- WPTS Definitions and Data Structures
  *Contains the WPTS specific definitions and data structures used in the WPTS APIs.*

# WPTS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Wpts

## Functions

- void CyBle_WptsRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_WptsRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode indicates the event that triggered this callback (e.g. CYBLE_EVT_WPTSS_INDICATION_ENABLED).<br>• eventParam contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_WPTS_CHAR_VALUE_T structure that contains details of the characteristic for which notification enabled event was triggered). |

# WPTS Server Functions

## Description

APIs unique to WPTS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Wptss

## Functions

- CYBLE_API_RESULT_T CyBle_WptssSetCharacteristicValue (CYBLE_WPTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptssGetCharacteristicValue (CYBLE_WPTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptssSetCharacteristicDescriptor (CYBLE_WPTS_CHAR_INDEX_T charIndex, CYBLE_WPTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptssGetCharacteristicDescriptor (CYBLE_WPTS_CHAR_INDEX_T charIndex, CYBLE_WPTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_WptssSendNotification (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_WptssSetCharacteristicValue (CYBLE_WPTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a characteristic value of the Wireless Power Transfer Service in the local GATT database. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was written successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_WptssGetCharacteristicValue (CYBLE_WPTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a characteristic value of the Wireless Power Transfer Service, which is identified by charIndex from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_WptssSetCharacteristicDescriptor (CYBLE_WPTS_CHAR_INDEX_T *charIndex*, CYBLE_WPTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_WPTS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data that should be stored to the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_WptssGetCharacteristicDescriptor (CYBLE_WPTS_CHAR_INDEX_T *charIndex*, CYBLE_WPTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a characteristic descriptor of a specified characteristic of the Wireless Power Transfer Service from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_WPTS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed

**CYBLE_API_RESULT_T CyBle_WptssSendNotification (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends notification with a characteristic value of the WPTS, which is a value specified by charIndex, to the Client device.

On enabling notification successfully for a service characteristic it sends out a 'Handle Value Notification' which results in CYBLE_EVT_WPTSC_NOTIFICATION event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the Client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_NTF_DISABLED - Notification is not enabled by the Client.

**CYBLE_API_RESULT_T CyBle_WptssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value of the Wireless Power Transfer Service, which is a value specified by charIndex, to the Client device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_WPTSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the Client device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - Optional characteristic is absent
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WPTS service-specific callback is registered (with CyBle_WptsRegisterAttrCallback):

- CYBLE_EVT_WPTSS_INDICATION_CONFIRMED - in case if the indication is successfully delivered to the peer device.

Otherwise (if the WPTS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - in case if the indication is successfully delivered to the peer device.

# WPTS Client Functions

## Description

APIs unique to WPTS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Wptsc

## Functions

- void CyBle_WptscDiscovery (CYBLE_GATT_DB_ATTR_HANDLE_T servHandle)
- CYBLE_API_RESULT_T CyBle_WptscSetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_WptscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex, CYBLE_WPTS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WptscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WPTS_CHAR_INDEX_T charIndex, CYBLE_WPTS_DESCR_INDEX_T descrIndex)

## Function Documentation

**void CyBle_WptscDiscovery (CYBLE_GATT_DB_ATTR_HANDLE_T *servHandle*)**

This function discovers the PRU's WPT service and characteristics using the GATT Primary Service Handle, received through the WPT Service Data within the PRU advertisement payload, together with the handle offsets defined A4WP specification.

The PTU may perform service discovery using the CyBle_GattcStartDiscovery(). This function may be used in response to Service Changed indication or to discover services other than the WPT service supported by the PRU.

**Parameters:**

| | |
|---|---|
| *servHandle* | GATT Primary Service Handle of the WPT service. |

**CYBLE_API_RESULT_T CyBle_WptscSetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic (which is identified by charIndex) value attribute in the server. As a result a Write Request is sent to the GATT Server and on successful execution of the request on the Server side the CYBLE_EVT_WPTSS_WRITE_CHAR events is generated. On successful request execution on the Server side the Write Response is sent to the Client.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be send to the server device. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WPTS service-specific callback is registered (with CyBle_WptsRegisterAttrCallback):

- CYBLE_EVT_WPTSC_WRITE_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, etc.) are provided with event parameter structure of type CYBLE_WPTS_CHAR_VALUE_T.

Otherwise (if the WPTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_WptscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |

**Returns:**

Return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WPTS service-specific callback is registered (with CyBle_WptsRegisterAttrCallback):

- CYBLE_EVT_WPTSC_READ_CHAR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index , value, etc.) are provided with event parameter structure of type CYBLE_WPTS_CHAR_VALUE_T.

Otherwise (if the WPTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

### CYBLE_API_RESULT_T CyBle_WptscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*, CYBLE_WPTS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)

This function is used to write the characteristic descriptor to the server, which is identified by charIndex and descrIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_WPTSS_NOTIFICATION_ENABLED
- CYBLE_EVT_WPTSS_NOTIFICATION_DISABLED
- CYBLE_EVT_WPTSS_INDICATION_ENABLED
- CYBLE_EVT_WPTSS_INDICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_WPTS_DESCR_INDEX_T. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WPTS service-specific callback is registered (with CyBle_WptsRegisterAttrCallback):

- CYBLE_EVT_WPTSC_WRITE_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_WPTS_DESCR_VALUE_T.

Otherwise (if the WPTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - in case if the requested attribute is successfully wrote on the peer device.
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_WptscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WPTS_CHAR_INDEX_T *charIndex*, CYBLE_WPTS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of a service characteristic of type CYBLE_WPTS_CHAR_INDEX_T. |
| *descrIndex* | The index of a service characteristic descriptor of type CYBLE_WPTS_DESCR_INDEX_T. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WPTS service-specific callback is registered (with CyBle_WptsRegisterAttrCallback):

- CYBLE_EVT_WPTSC_READ_DESCR_RESPONSE - in case if the requested attribute is successfully wrote on the peer device, the details (char index, descr index, value, etc.) are provided with event parameter structure of type CYBLE_WPTS_DESCR_VALUE_T.

Otherwise (if the WPTS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - in case if the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - in case if there some trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# WPTS Definitions and Data Structures

## Description

Contains the WPTS specific definitions and data structures used in the WPTS APIs.

## Data Structures

- struct CYBLE_WPTSS_CHAR_T
- struct CYBLE_WPTS_CHAR_VALUE_T
- struct CYBLE_WPTS_DESCR_VALUE_T
- struct CYBLE_WPTSS_T
- struct CYBLE_WPTSC_CHAR_T
- struct CYBLE_WPTSC_T

## Enumerations

- enum CYBLE_WPTS_CHAR_INDEX_T
- enum CYBLE_WPTS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_WPTSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_WPTS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSS_CHAR_T::charHandle**

Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T**
**CYBLE_WPTSS_CHAR_T::descrHandle[CYBLE_WPTS_DESCR_COUNT]**

Handle of descriptor

### struct CYBLE_WPTS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_WPTS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_WPTS_CHAR_VALUE_T::connHandle**

Peer device handle

**CYBLE_WPTS_CHAR_INDEX_T CYBLE_WPTS_CHAR_VALUE_T::charIndex**

Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_WPTS_CHAR_VALUE_T::value**

Characteristic value

**struct CYBLE_WPTS_DESCR_VALUE_T**

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_WPTS_CHAR_INDEX_T charIndex
- CYBLE_WPTS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_WPTS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_WPTS_CHAR_INDEX_T CYBLE_WPTS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_WPTS_DESCR_INDEX_T CYBLE_WPTS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_VALUE_T* CYBLE_WPTS_DESCR_VALUE_T::value**
Characteristic value

**struct CYBLE_WPTSS_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_WPTSS_CHAR_T charInfo [CYBLE_WPTS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSS_T::serviceHandle**
Wireless Power Transfer Service handle

**CYBLE_WPTSS_CHAR_T CYBLE_WPTSS_T::charInfo[CYBLE_WPTS_CHAR_COUNT]**
Wireless Power Transfer Characteristic handles

**struct CYBLE_WPTSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_WPTS_DESCR_COUNT]
- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- uint8 properties

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSC_CHAR_T::descrHandle[CYBLE_WPTS_DESCR_COUNT]**
Handles of descriptors

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSC_CHAR_T::valueHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSC_CHAR_T::endHandle**
End handle of a characteristic

**uint8 CYBLE_WPTSC_CHAR_T::properties**
Properties for value field

**struct CYBLE_WPTSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_WPTSC_CHAR_T charInfo [CYBLE_WPTS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WPTSC_T::serviceHandle**

Wireless Power Transfer Service handle

**CYBLE_WPTSC_CHAR_T CYBLE_WPTSC_T::charInfo[CYBLE_WPTS_CHAR_COUNT]**

Wireless Power Transfer Service characteristics info structure

### Enumeration Type Documentation

**enum CYBLE_WPTS_CHAR_INDEX_T**

WPTS Characteristic indexes

**Enumerator**

***CYBLE_WPTS_PRU_CONTROL*** PRU Control Characteristic index

***CYBLE_WPTS_PTU_STATIC_PAR*** PTU Static Parameter Characteristic index

***CYBLE_WPTS_PRU_ALERT*** PRU Alert Characteristic index

***CYBLE_WPTS_PRU_STATIC_PAR*** PRU Static Parameter Characteristic index

***CYBLE_WPTS_PRU_DYNAMIC_PAR*** PRU Dynamic Parameter Characteristic index

***CYBLE_WPTS_CHAR_COUNT*** Total count of WPTS Characteristics

**enum CYBLE_WPTS_DESCR_INDEX_T**

WPTS Characteristic Descriptors indexes

**Enumerator**

***CYBLE_WPTS_CCCD*** Client Characteristic Configuration Descriptor index

***CYBLE_WPTS_DESCR_COUNT*** Total count of Descriptors

## Weight Scale Service (WSS)

### Description

The Weight Scale Service exposes weight and related data from a weight scale (Server) intended for consumer healthcare as well as sports/fitness applications.

Depending on the chosen GATT role in the GUI, you may use a subset of the supported APIs.

The WSS API names begin with CyBle_Wss. In addition to this, the APIs also append the GATT role initial letter in the API name.

### Modules

- WSS Server and Client Function
  *These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.*
- WSS Server Functions
  *APIs unique to WSS designs configured as a GATT Server role.*
- WSS Client Functions
  *APIs unique to WSS designs configured as a GATT Client role.*

- WSS Definitions and Data Structures
  *Contains the WSS specific definitions and data structures used in the WSS APIs.*

# WSS Server and Client Function

## Description

These are APIs common to both GATT Client role and GATT Server role. You may use them in either roles.
No letter is appended to the API name: CyBle_Wss

## Functions

- void CyBle_WssRegisterAttrCallback (CYBLE_CALLBACK_T callbackFunc)

## Function Documentation

### void CyBle_WssRegisterAttrCallback (CYBLE_CALLBACK_T *callbackFunc*)

Registers a callback function for service specific attribute operations. Service specific write requests from peer device will not be handled with unregistered callback function.

**Parameters:**

| | |
|---|---|
| *callbackFunc* | An application layer event callback function to receive events from the BLE Component. The definition of CYBLE_CALLBACK_T is:<br>typedef void (* CYBLE_CALLBACK_T) (uint32 eventCode, void *eventParam)<br>• eventCode - Indicates the event that triggered this callback (e.g. CYBLE_EVT_WSSS_INDICATION_ENABLED).<br>• eventParam - Contains the parameters corresponding to the current event. (e.g. pointer to CYBLE_WSS_CHAR_VALUE_T structure that contains details of the characteristic for which an indication enabled event was triggered). |

# WSS Server Functions

## Description

APIs unique to WSS designs configured as a GATT Server role.
A letter 's' is appended to the API name: CyBle_Wsss

## Functions

- uint8 CyBle_WssGetAdUserIdListSize (void)
- CYBLE_API_RESULT_T CyBle_WssSetAdUserId (uint8 listSize, const uint8 userIdList[])
- CYBLE_API_RESULT_T CyBle_WsssSetCharacteristicValue (CYBLE_WSS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WsssGetCharacteristicValue (CYBLE_WSS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WsssSetCharacteristicDescriptor (CYBLE_WSS_CHAR_INDEX_T charIndex, CYBLE_WSS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)

- CYBLE_API_RESULT_T CyBle_WsssGetCharacteristicDescriptor (CYBLE_WSS_CHAR_INDEX_T charIndex, CYBLE_WSS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WsssSendIndication (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WSS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *attrValue)

## Function Documentation

**uint8 CyBle_WssGetAdUserIdListSize (void )**

Returns the size (in bytes) of User ID List in the advertisement packet.

**Returns:**

Size of User ID List.

**CYBLE_API_RESULT_T CyBle_WssSetAdUserId (uint8 *listSize*, const uint8 *userIdList*[])**

Sets the User ID List to the advertisement packet. To be able to set the User ID List with this function, the advertisement packet should be configured in the component GUI to include Weight Scale Service UUID in the Service Data field. The Service Data should have enough room to fit the User ID List that is planned to be advertised. To reserve the room for the User ID List, the Service Data for WSS should be filled with Unknown User ID - 0xFF. The amount of 0xFF's should be equal to User List Size that is planned to be advertised. This function must be called when CyBle_GetBleSsState() returns CYBLE_BLESS_STATE_EVENT_CLOSE state.

**Parameters:**

| | |
|---|---|
| *listSize* | The size of the User List. |
| *userIdList* | The array contains a User List. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - On NULL pointer, Data length in input parameter exceeds 31 bytes.
- CYBLE_ERROR_INVALID_OPERATION - The advertisement packet doesn't contain the User List or advertisement packet is to small or ADV Event is not closed, BLESS is active or ADV is not enabled.

**CYBLE_API_RESULT_T CyBle_WsssSetCharacteristicValue (CYBLE_WSS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets a value for one of two characteristic values of the Weight Scale Service. The characteristic is identified by charIndex.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of a Weight Scale Service characteristic. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be stored to the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was written successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_WsssGetCharacteristicValue (CYBLE_WSS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a characteristic value of the Weight Scale Service, which is identified by charIndex from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the Weight Scale Service characteristic. |
| *attrSize* | The size of the Weight Scale Service characteristic value attribute. |
| *attrValue* | The pointer to the location where characteristic value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The characteristic value was read successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

**CYBLE_API_RESULT_T CyBle_WsssSetCharacteristicDescriptor (CYBLE_WSS_CHAR_INDEX_T *charIndex*, CYBLE_WSS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sets the characteristic descriptor of the specified characteristic.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the service characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the characteristic descriptor attribute. |
| *attrValue* | The pointer to the descriptor value data to be stored in the GATT database. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_WsssGetCharacteristicDescriptor (CYBLE_WSS_CHAR_INDEX_T *charIndex*, CYBLE_WSS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Reads a a characteristic descriptor of a specified characteristic of the Weight Scale Service from the GATT database.

**Parameters:**

| | |
|---|---|
| *charIndex* | The index of the characteristic. |
| *descrIndex* | The index of the descriptor. |
| *attrSize* | The size of the descriptor value. |
| *attrValue* | The pointer to the location where characteristic descriptor value data should be stored. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.

**CYBLE_API_RESULT_T CyBle_WsssSendIndication (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WSS_CHAR_INDEX_T *charIndex*, uint8 *attrSize*, uint8 **attrValue*)**

Sends an indication with a characteristic value of the Weight Scale Service, which is a value specified by charIndex, to the client's device.

On enabling indication successfully it sends out a 'Handle Value Indication' which results in CYBLE_EVT_WSSC_INDICATION or CYBLE_EVT_GATTC_HANDLE_VALUE_IND (if service specific callback function is not registered) event at the GATT Client's end.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. |

| | |
|---|---|
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic value data that should be sent to the client's device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was handled successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameter failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted.
- CYBLE_ERROR_INVALID_STATE - Connection with the client is not established.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_IND_DISABLED - Indication is not enabled by the client.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WSS service-specific callback is registered (with CyBle_WssRegisterAttrCallback()):

- CYBLE_EVT_WSSS_INDICATION_CONFIRMED - If the indication is successfully delivered to the peer device.

Otherwise (if the WSS service-specific callback is not registered):

- CYBLE_EVT_GATTS_HANDLE_VALUE_CNF - If the indication is successfully delivered to the peer device.

# WSS Client Functions

## Description

APIs unique to WSS designs configured as a GATT Client role.

A letter 'c' is appended to the API name: CyBle_Wssc

## Functions

- CYBLE_API_RESULT_T CyBle_WsscGetCharacteristicValue (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WSS_CHAR_INDEX_T charIndex)
- CYBLE_API_RESULT_T CyBle_WsscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WSS_CHAR_INDEX_T charIndex, CYBLE_WSS_DESCR_INDEX_T descrIndex, uint8 attrSize, uint8 *attrValue)
- CYBLE_API_RESULT_T CyBle_WsscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T connHandle, CYBLE_WSS_CHAR_INDEX_T charIndex, CYBLE_WSS_DESCR_INDEX_T descrIndex)

## Function Documentation

**CYBLE_API_RESULT_T CyBle_WsscGetCharacteristicValue (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WSS_CHAR_INDEX_T *charIndex*)**

This function is used to read a characteristic value, which is a value identified by charIndex, from the server.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The read request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.

- CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE - The peer device doesn't have the particular characteristic.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_STATE - Connection with the server is not established.
- CYBLE_ERROR_INVALID_OPERATION - Operation is invalid for this characteristic.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WSS service-specific callback is registered (with CyBle_WssRegisterAttrCallback):

- CYBLE_EVT_WSSC_READ_CHAR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index , value, etc.) are provided with an event parameter structure of type CYBLE_WSS_CHAR_VALUE_T.

Otherwise (if the WSS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_WsscSetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WSS_CHAR_INDEX_T *charIndex*, CYBLE_WSS_DESCR_INDEX_T *descrIndex*, uint8 *attrSize*, uint8 **attrValue*)**

This function is used to write the characteristic descriptor to the server, which is identified by charIndex and descrIndex.

Internally, Write Request is sent to the GATT Server and on successful execution of the request on the Server side the following events can be generated:

- CYBLE_EVT_WSSS_INDICATION_ENABLED
- CYBLE_EVT_WSSS_INDICATION_DISABLED

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |
| *attrSize* | The size of the characteristic value attribute. |
| *attrValue* | The pointer to the characteristic descriptor value data that should be sent to the server device. |

**Returns:**

A return value is of type CYBLE_API_RESULT_T.

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WSS service-specific callback is registered (with CyBle_WssRegisterAttrCallback):

- CYBLE_EVT_WSSC_WRITE_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index etc.) are provided with event parameter structure of type CYBLE_WSS_DESCR_VALUE_T.

Otherwise (if the WSS service-specific callback is not registered):

- CYBLE_EVT_GATTC_WRITE_RSP - If the requested attribute is successfully written on the peer device.

- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

**CYBLE_API_RESULT_T CyBle_WsscGetCharacteristicDescriptor (CYBLE_CONN_HANDLE_T *connHandle*, CYBLE_WSS_CHAR_INDEX_T *charIndex*, CYBLE_WSS_DESCR_INDEX_T *descrIndex*)**

Sends a request to get the characteristic descriptor of the specified characteristic of the service.

**Parameters:**

| | |
|---|---|
| *connHandle* | The connection handle. |
| *charIndex* | The index of the service characteristic. Starts with zero. |
| *descrIndex* | The index of the service characteristic descriptor. |

**Returns:**

- CYBLE_ERROR_OK - The request was sent successfully.
- CYBLE_ERROR_INVALID_PARAMETER - Validation of the input parameters failed.
- CYBLE_ERROR_INVALID_STATE - The state is not valid.
- CYBLE_ERROR_MEMORY_ALLOCATION_FAILED - Memory allocation failed.
- CYBLE_ERROR_INVALID_OPERATION - This operation is not permitted on the specified attribute.

**Events**

In case of successful execution (return value = CYBLE_ERROR_OK) the next events can appear:
If the WSS service-specific callback is registered (with CyBle_WssRegisterAttrCallback()):

- CYBLE_EVT_WSSC_READ_DESCR_RESPONSE - If the requested attribute is successfully written on the peer device, the details (char index, descr index, value, etc.) are provided with an event parameter structure of type CYBLE_WSS_DESCR_VALUE_T.

Otherwise (if the WSS service-specific callback is not registered):

- CYBLE_EVT_GATTC_READ_RSP - If the requested attribute is successfully read on the peer device, the details (handle, value, etc.) are provided with an event parameters structure (CYBLE_GATTC_READ_RSP_PARAM_T).
- CYBLE_EVT_GATTC_ERROR_RSP - If there is trouble with the requested attribute on the peer device, the details are provided with event parameters structure (CYBLE_GATTC_ERR_RSP_PARAM_T).

# WSS Definitions and Data Structures

## Description

Contains the WSS specific definitions and data structures used in the WSS APIs.

## Data Structures

- struct CYBLE_WSS_CHAR_VALUE_T
- struct CYBLE_WSS_DESCR_VALUE_T
- struct CYBLE_WSSS_CHAR_T
- struct CYBLE_WSSS_T
- struct CYBLE_WSSC_CHAR_T
- struct CYBLE_WSSC_T

## Enumerations

- enum CYBLE_WSS_CHAR_INDEX_T
- enum CYBLE_WSS_DESCR_INDEX_T

## Data Structure Documentation

### struct CYBLE_WSS_CHAR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_WSS_CHAR_INDEX_T charIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_WSS_CHAR_VALUE_T::connHandle**
Peer device handle

**CYBLE_WSS_CHAR_INDEX_T CYBLE_WSS_CHAR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_GATT_VALUE_T* CYBLE_WSS_CHAR_VALUE_T::value**
Characteristic value

### struct CYBLE_WSS_DESCR_VALUE_T

**Data Fields**

- CYBLE_CONN_HANDLE_T connHandle
- CYBLE_WSS_CHAR_INDEX_T charIndex
- CYBLE_WSS_DESCR_INDEX_T descrIndex
- CYBLE_GATT_VALUE_T * value

**Field Documentation**

**CYBLE_CONN_HANDLE_T CYBLE_WSS_DESCR_VALUE_T::connHandle**
Peer device handle

**CYBLE_WSS_CHAR_INDEX_T CYBLE_WSS_DESCR_VALUE_T::charIndex**
Index of service characteristic

**CYBLE_WSS_DESCR_INDEX_T CYBLE_WSS_DESCR_VALUE_T::descrIndex**
Index of descriptor

**CYBLE_GATT_VALUE_T* CYBLE_WSS_DESCR_VALUE_T::value**
Characteristic value

### struct CYBLE_WSSS_CHAR_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T charHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_WSS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSS_CHAR_T::charHandle**
Handle of characteristic value

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSS_CHAR_T::descrHandle[CYBLE_WSS_DESCR_COUNT]**
Array of descriptor handles

### struct CYBLE_WSSS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_WSSS_CHAR_T charInfo [CYBLE_WSS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSS_T::serviceHandle**
Weight Scale Service handle

**CYBLE_WSSS_CHAR_T CYBLE_WSSS_T::charInfo[CYBLE_WSS_CHAR_COUNT]**
Array of characteristics and descriptors handles

**struct CYBLE_WSSC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T valueHandle
- uint8 properties
- CYBLE_GATT_DB_ATTR_HANDLE_T endHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T descrHandle [CYBLE_WSS_DESCR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSC_CHAR_T::valueHandle**
Handle of characteristic value

**uint8 CYBLE_WSSC_CHAR_T::properties**
Properties for value field

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSC_CHAR_T::endHandle**
End handle of characteristic

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSC_CHAR_T::descrHandle[CYBLE_WSS_DESCR_COUNT]**
Array of descriptor handles

**struct CYBLE_WSSC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T serviceHandle
- CYBLE_WSSC_CHAR_T charInfo [CYBLE_WSS_CHAR_COUNT]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_WSSC_T::serviceHandle**
Weight Scale Service handle

**CYBLE_WSSC_CHAR_T CYBLE_WSSC_T::charInfo[CYBLE_WSS_CHAR_COUNT]**
Weight Scale Service characteristics info structure

## Enumeration Type Documentation

**enum CYBLE_WSS_CHAR_INDEX_T**
WSS Characteristic indexes
**Enumerator**
***CYBLE_WSS_WEIGHT_SCALE_FEATURE*** Weight Scale Feature Characteristic index
***CYBLE_WSS_WEIGHT_MEASUREMENT*** Weight Measurement Characteristic index
***CYBLE_WSS_CHAR_COUNT*** Total count of WSS Characteristics

**enum CYBLE_WSS_DESCR_INDEX_T**
WSS Characteristic Descriptors indexes
**Enumerator**
***CYBLE_WSS_CCCD*** Client Characteristic Configuration Descriptor index

***CYBLE_WSS_DESCR_COUNT*** Total count of Descriptors

# Custom Service

## Description

This section contains the description of structs used for Custom Services.

## Data Structures

- struct CYBLE_CUSTOMS_INFO_T
- struct CYBLE_CUSTOMS_T
- struct CYBLE_CUSTOMC_DESC_T
- struct CYBLE_CUSTOMC_CHAR_T
- struct CYBLE_CUSTOMC_T

## Variables

- const CYBLE_CUSTOMS_T cyBle_customs [(`$CustomSCount`)]
- CYBLE_CUSTOMC_T cyBle_customCServ [(`$CustomCCount`)]

## Data Structure Documentation

### struct CYBLE_CUSTOMS_INFO_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T customServCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T customServCharDesc [(`$CustomMaxDescriptorCount`)==0u?1u:(`$CustomMaxDescriptorCount`)]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMS_INFO_T::customServCharHandle**

Custom Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMS_INFO_T::customServCharDesc[(`$CustomMaxDescriptorCount`)==0u?1u:(`$CustomMaxDescriptorCount`)]**

Custom Characteristic Descriptors handles

### struct CYBLE_CUSTOMS_T

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T customServHandle
- CYBLE_CUSTOMS_INFO_T customServInfo [(`$CustomMaxCharacteristicCount`)==0u?1u:(`$CustomMaxCharacteristicCount`)]

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMS_T::customServHandle**

Handle of a Custom Service

**CYBLE_CUSTOMS_INFO_T**
**CYBLE_CUSTOMS_T::customServInfo[(`$CustomMaxCharacteristicCount`)==0u?1u:(`$CustomMaxCharacteristicCount`)]**
Information about Custom Characteristics

**struct CYBLE_CUSTOMC_DESC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T descHandle
- const void * uuid
- uint8 uuidFormat

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMC_DESC_T::descHandle**
Custom Descriptor handle

**const void* CYBLE_CUSTOMC_DESC_T::uuid**
Custom Descriptor 128 bit UUID

**uint8 CYBLE_CUSTOMC_DESC_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**struct CYBLE_CUSTOMC_CHAR_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T customServCharHandle
- CYBLE_GATT_DB_ATTR_HANDLE_T customServCharEndHandle
- const void * uuid
- uint8 uuidFormat
- uint8 properties
- uint8 descCount
- CYBLE_CUSTOMC_DESC_T * customServCharDesc

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMC_CHAR_T::customServCharHandle**
Characteristic handle

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMC_CHAR_T::customServCharEndHandle**
Characteristic end handle

**const void* CYBLE_CUSTOMC_CHAR_T::uuid**
Custom Characteristic UUID

**uint8 CYBLE_CUSTOMC_CHAR_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**uint8 CYBLE_CUSTOMC_CHAR_T::properties**
Properties for value field

**uint8 CYBLE_CUSTOMC_CHAR_T::descCount**
Number of descriptors

**CYBLE_CUSTOMC_DESC_T* CYBLE_CUSTOMC_CHAR_T::customServCharDesc**
Characteristic Descriptors

**struct CYBLE_CUSTOMC_T**

**Data Fields**

- CYBLE_GATT_DB_ATTR_HANDLE_T customServHandle

- const void * uuid
- uint8 uuidFormat
- uint8 charCount
- CYBLE_CUSTOMC_CHAR_T * customServChar

**Field Documentation**

**CYBLE_GATT_DB_ATTR_HANDLE_T CYBLE_CUSTOMC_T::customServHandle**
Custom Service handle

**const void* CYBLE_CUSTOMC_T::uuid**
Custom Service UUID

**uint8 CYBLE_CUSTOMC_T::uuidFormat**
UUID Format - 16-bit (0x01) or 128-bit (0x02)

**uint8 CYBLE_CUSTOMC_T::charCount**
Number of characteristics

**CYBLE_CUSTOMC_CHAR_T* CYBLE_CUSTOMC_T::customServChar**
Custom Service Characteristics

## Variable Documentation

**const CYBLE_CUSTOMS_T cyBle_customs[(`$CustomSCount`)]**
Custom Services GATT DB handles structures

**CYBLE_CUSTOMC_T cyBle_customCServ[(`$CustomCCount`)]**
Custom Services discovered attributes information

## Code snippets

- For an application callback: void CyBle_AppCallback( uint32 eventCode, void *eventParam ){<all general events>}
- For each CyBle_<service>RegisterAttrCallback API function: CyBle_<service>RegisterAttrCallback( CyBle_<service>CallBack );
- For each service callback: void CyBle_<service>CallBack( uint32 eventCode, void *eventParam ) {<all service-specific events>}

## Sample Firmware Source Code

PSoC Creator provides numerous example projects 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:

- AN94020 - Getting Started with PRoC BLE
- AN92584 - Designing for Low Power and Estimating Battery Life for BLE Applications
- AN91184 - Creating BLE Applications Using PSoC 4 BLE
- AN96112 - Creating Custom Profiles Using PSoC 4 BLE
- AN95089 - PSoC® 4/PRoC™ BLE Crystal Oscillator Selection and Tuning Techniques
- AN97060 - PSoC® 4/PRoC™ Over-The-Air (OTA) Firmware Upgrade Guide
- AN85951 - CapSense Design Guide
- AN91445 - Antenna Design Guide
- AN99209 - PSoC® 4 BLE and PRoC™ BLE : Bluetooth LE 4.2 features

Additionally you can look to 100 projects in 100 days blog that describes a variety of projects that expose possible use of BLE component.

# 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

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The BLE Component has the following specific deviations.

| MISRA-C:2004 Rule | Rule Class (Required/ Advisory) | Rule Description | Description of Deviation(s) |
|---|---|---|---|
| 9.3 | R | In an enumerator list, the '=' construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized. | Violated when a specific value needs to be assigned to an enumerator item. |
| 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. |
| 13.7 | R | Boolean operations whose results are invariant shall not be permitted. | A Boolean operator can yields a result that can be proven to be always "true" or always "false" in some specific configurations because of generalized implementation approach. |
| 17.4 | R | Array indexing shall be the only allowed form of pointer arithmetic. | An array subscript operator is being used to subscript an expression which is not of array type. This is perfectly legitimate in the C language providing the pointer addresses an array element. |
| 18.4 | R | Unions shall not be used. | Deviated for constructing an efficient implementation. |

| MISRA-C:2004 Rule | Rule Class (Required/ Advisory) | Rule Description | Description of Deviation(s) |
|---|---|---|---|
| 19.7 | A | A function should be used in preference to a function-like macro. | Deviated for more efficient code. |

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 |
|---|---|---|
| 76858 | D028204 | 4.2 Host |
| 76764 | D028203 | 4.2 Link Layer |
| 63199 | D025070 | Profiles supported by BLE Component in PSoC Creator |
| 73181 | D026298 | |
| 61908 | D024756 | Host |
| 62243 | D024755 | Link Layer |
| 62245 | D024754 | RF-PHY for 56-QFN package |
| 63368 | D025068 | RF-PHY for 68-ball WLCSP package |
| 62887 | D024757 | PSoC 4 BLE and PRoC BLE end product (56-QFN package) |
| 63683 | D025069 | PSoC 4 BLE and PRoC BLE end product (68-ball WLCSP package) |

## API Memory Usage

The Component memory usage varies significantly, depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements are done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

The Component's BLE Stack is implemented in four libraries and therefore the Component memory usage is directly dependent on the library used. The libraries are:

- HCI Library (used in HCI mode)
- Peripheral (used when the Component is configured for GAP Peripheral or GAP Broadcaster role)
- Central (used when the Component is configured for GAP Central or GAP Observer role)
- Peripheral and Central (used when the Component is configured for GAP Peripheral and Central roles)

### HCI Mode

| Configuration | PSoC 4200 BLE (GCC) | | |
|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Stack Bytes |
| HCI Mode | 41100 | 3177 | 2048 |

### Peripheral and Central Profile Mode

| Configuration | PSoC 4200 BLE (GCC) | | |
|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Stack Bytes |
| Alert Notification Profile (Server) | 91290 | 8466 | 2048 |
| Find Me Profile (Find Me Target role) | 90794 | 8394 | 2048 |
| Internet Protocol Support | 90476 | 11592 | 2048 |
| Phone Alert Status | 91118 | 8443 | 2048 |
| Time | 91842 | 8486 | 2048 |

### Central Profile Mode

| Configuration | PSoC 4200 BLE (GCC) | | |
|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Stack Bytes |
| Alert Notification Profile (Server) | 84238 | 8284 | 2048 |
| Find Me Profile (Find Me Target role) | 83450 | 8189 | 2048 |
| HID over GATT Profile (Host) | 89236 | 8367 | 2048 |
| Phone Alert Status | 83934 | 8227 | 2048 |
| Proximity Profile (Proximity Reporter) | 84314 | 8215 | 2048 |
| Time | 84642 | 8270 | 2048 |

### Peripheral Profile Mode

| Configuration | PSoC 4200 BLE (GCC) | | |
|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Stack Bytes |
| Blood Pressure | 82294 | 8367 | 2048 |
| Bootloader | 81758 | 8249 | 2048 |
| Continuous Glucose Monitoring | 83612 | 8461 | 2048 |
| Cycling Power | 82756 | 8320 | 2048 |
| Cycling Speed and Cadence | 82424 | 8359 | 2048 |
| Custom | 81250 | 8251 | 2048 |
| Environmental Sensing | 86696 | 9978 | 2048 |
| Find Me Profile (Find Me Target role) | 81352 | 8203 | 2048 |
| Glucose Profile (Glucose Sensor) | 82632 | 8378 | 2048 |
| Health Thermometer Profile (Server) | 82716 | 8382 | 2048 |
| Heart Rate Profile (Heart Rate Sensor) | 82250 | 8321 | 2048 |
| HID Over GATT Profile (HID Device) | 84134 | 8493 | 2048 |
| Internet Protocol Support | 83746 | 8523 | 2048 |
| Location and Navigation | 82166 | 8299 | 2048 |
| Proximity Profile (Proximity Reporter) | 82184 | 8240 | 2048 |
| Running Speed and Cadence | 82444 | 8362 | 2048 |
| Scan Parameters Profile (Scan Server) | 81716 | 8226 | 2048 |
| Weight Scale | 87372 | 8933 | 2048 |
| Wireless Power Transfer | 82358 | 8362 | 2048 |
| BLE 4.2. Data Length, Security, Privacy. | 99670 | 19581 | 2048 |

# Resources

The BLE 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.

| Parameter | Description | Min | Typ | Max | Units | Details/Conditions |
|---|---|---|---|---|---|---|
| **RF Receiver Specification** | | | | | | |
| RXS, IDLE | RX sensitivity with idle transmitter | – | –89 | – | dBm | |
| | RX sensitivity with idle transmitter excluding Balun loss | – | –91 | – | dBm | Guaranteed by design simulation |
| RXS, DIRTY | RX sensitivity with dirty transmitter | – | –87 | –70 | dBm | RF-PHY Specification (RCV-LE/CA/01/C) |
| RXS, HIGHGAIN | RX sensitivity in high-gain mode with idle transmitter | – | –91 | – | dBm | |
| PRXMAX | Maximum input power | –10 | –1 | – | dBm | RF-PHY Specification (RCV-LE/CA/06/C) |
| CI1 | Cochannel 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 ±1 MHz | – | 3 | 15 | 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 |
|---|---|---|---|---|---|---|
| CI3 | Adjacent channel interference Wanted signal at –67 dBm and Interferer at FRX ±2 MHz | – | –29 | – | 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 | – | dB | RF-PHY Specification (RCV-LE/CA/03/C) |
| CI5 | Adjacent channel interference Wanted Signal at –67 dBm and Interferer at Image frequency ($F_{IMAGE}$) | – | –20 | – | dB | RF-PHY Specification (RCV-LE/CA/03/C) |
| CI3 | Adjacent channel interference Wanted signal at –67 dBm and Interferer at Image frequency ($F_{IMAGE}$ ± 1 MHz) | – | –30 | – | dB | RF-PHY Specification (RCV-LE/CA/03/C) |
| 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 a –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, third, fourth, and fifth 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 | – | – | –47 | dBm | 1-MHz measurement bandwidth ETSI EN300 328 V1.8.1 |
| **RF Transmitter Specifications** | | | | | | |
| TXP, ACC | RF power accuracy | – | – | ±4 | dB | |
| TXP, RANGE | RF power control range | – | 20 | – | dB | |
| TXP, 0dBm | Output power, 0-dB Gain setting (PA7) | –4 | 0 | 3 | dBm | |

| Parameter | Description | Min | Typ | Max | Units | Details/Conditions |
|---|---|---|---|---|---|---|
| TXP, MAX | Output power, maximum power setting (PA10) | –1 | 3 | 6 | dBm | |
| TXP, MIN | Output power, minimum power setting (PA1) | – | –18 | – | dBm | |
| F2AVG | Average frequency deviation for 10101010 pattern | 185 | – | – | 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) |
| EO | Eye opening = ΔF2AVG/ΔF1AVG | 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) |
| 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 | – | – | –20 | dBm | RF-PHY Specification (TRM-LE/CA/03/C) |
| IBSE2 | In-band spurious emission at ≥3-MHz offset | – | – | -30 | dBm | RF-PHY Specification (TRM-LE/CA/03/C) |
| TXSE1 | Transmitter spurious emissions (average), <1.0 GHz | – | – | -55.5 | dBm | FCC-15.247 |
| TXSE2 | Transmitter spurious emissions (average), >1.0 GHz | – | – | -41.5 | dBm | FCC-15.247 |
| **RF Current Specifications** | | | | | | |
| IRX | Receive current in normal mode | – | 18.7 | – | mA | |
| IRX_RF | Radio receive current in normal mode | – | 16.4 | – | mA | Measured at $V_{DDR}$ |
| IRX, HIGHGAIN | Receive current in high-gain mode | – | 21.5 | – | mA | |
| ITX, 3dBm | TX current at 3-dBm setting (PA10) | – | 20 | – | mA | |
| ITX, 0dBm | TX current at 0-dBm setting (PA7) | – | 16.5 | – | mA | |
| ITX_RF, 0dBm | Radio TX current at 0 dBm setting (PA7) | – | 15.6 | – | mA | Measured at $V_{DDR}$ |
| ITX_RF, 0dBm | Radio TX current at 0 dBm excluding Balun loss | – | 14.2 | – | mA | Guaranteed by design simulation |

| Parameter | Description | Min | Typ | Max | Units | Details/Conditions |
|---|---|---|---|---|---|---|
| ITX,-3dBm | TX current at –3-dBm setting (PA4) | – | 15.5 | – | mA | |
| ITX,-6dBm | TX current at –6-dBm setting (PA3) | – | 14.5 | – | mA | |
| ITX,-12dBm | TX current at –12-dBm setting (PA2) | – | 13.2 | – | mA | |
| ITX,-18dBm | TX current at –18-dBm setting (PA1) | – | 12.5 | – | mA | |
| Iavg_1sec, 0dBm | Average current at 1-second BLE connection interval | – | 18.9 | – | µA | TXP: 0 dBm; ±20-ppm master and slave clock accuracy. |
| Iavg_4sec, 0dBm | Average current at 4-second BLE connection interval | – | 6.25 | – | µA | TXP: 0 dBm; ±20-ppm master and slave clock accuracy. |
| **General RF Specifications** | | | | | | |
| FREQ | RF operating frequency | 2400 | – | 2482 | MHz | |
| CHBW | Channel spacing | – | 2 | – | MHz | |
| DR | On-air data rate | – | 1000 | – | kbps | |
| IDLE2TX | BLE.IDLE to BLE. TX transition time | – | 120 | 140 | µs | |
| IDLE2RX | BLE.IDLE to BLE. RX transition time | – | 75 | 120 | µs | |
| **RSSI Specifications** | | | | | | |
| RSSI, ACC | RSSI accuracy | – | ±5 | – | dB | |
| RSSI, RES | RSSI resolution | – | 1 | – | dB | |
| RSSI, PER | RSSI sample period | – | 6 | – | µs | |

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 set to 12 MHz, connection interval set to 7.5 ms, and Encryption is enabled.

| Operation | Duration (µs) |
|---|---|
| Ble Stack On Time | 10615.8 |
| ‘CyBle_ProcessEvents’ execution time (Best case) | 11.1 |
| Worst case BLE ISR Execution time | 80.3 |
| Start Scan execution time | 4702 |
| Passive Scan receive advertisement duration | 353 |

| Operation | Duration (µs) |
|---|---|
| Active Scan receive {Advertisement + Scan Response} duration | 339.8 |
| Read request processing time on GATT Server (Attribute MTU = 512 Bytes) | 11452.3 |
| Write request processing time on GATT Server (Attribute MTU = 512 Bytes) | 10692 |
| Connection time on GAP Central | 5749.5 |
| Connection time on GAP Peripheral | 3699.2 |
| Start advertisement execution time (Worst Case) | 4436.7 |
| 'CyBle_EnterLPM' execution time (Worst Case) | 294.2 |
| Notification processing time on GATT Server (Attribute MTU = 512 Bytes) | 2826.2 |
| Write command processing time on GATT Server (Attribute MTU = 512 Bytes) | 9486.1 |
| Creating L2CAP COC | 1811.2 |
| Response L2CAP COC | 1034.8 |

# Updating from BLE v1.x to BLE v2.x or later

If you are updating to BLE v2.x or later from version v1.0, 1.10 or 1.20 and if you have used *CYBLE_EVT_GATTS_PREP_WRITE_REQ* or *CYBLE_EVT_GATTS_EXEC_WRITE_REQ* events in your existing design, it is likely that your design will not build after the update.

The reason for this is that the mechanism for the events generation and the event parameters were modified to allow the *CYBLE_EVT_GATTS_PREP_WRITE_REQ* and *CYBLE_EVT_GATTS_EXEC_WRITE_REQ* events to be used by the Long Write Value and Reliable Write procedures.

The following table shows the changes between version 2.x and older versions of the BLE component.

| # | v1.0-1.20 | v2.x and later |
|---|---|---|
| 1 | Single CYBLE_EVT_GATTS_PREP_WRITE_REQ event is generated. | Multiple CYBLE_EVT_GATTS_PREP_WRITE_REQ events are generated |
| 2 | Multiple CYBLE_EVT_GATTS_EXEC_WRITE_REQ events are generated | Single CYBLE_EVT_GATTS_EXEC_WRITE_REQ event is generated. |
| 3 | The *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event has the following parameter structure:<br>`typedef struct {`<br>`  CYBLE_CONN_HANDLE_T connHandle;`<br>`  CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle;`<br>`} CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T` | The *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event has the following parameter structure:<br>`typedef struct {`<br>`  CYBLE_CONN_HANDLE_T connHandle;`<br>`  CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T * baseAddr;`<br>`  uint8 currentPrepWriteReqCount;`<br>`  uint8 gattErrorCode;`<br>`} CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T` |

| # | v1.0-1.20 | v2.x and later |
|---|---|---|
| 4 | The *CYBLE_EVT_GATTS_EXEC_WRITE_REQ* event has the following parameter structure:<br>`typedef struct {`<br>`  CYBLE_CONN_HANDLE_T connHandle;`<br>`  CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle;`<br>`  uint16 length;`<br>`  uint16 offset;`<br>`  uint8 result;`<br>`} CYBLE_GATTS_EXEC_WRITE_REQ_T` | The CYBLE_EVT_GATTS_EXEC_WRITE_REQ event has the following parameter structure:<br>`typedef struct {`<br>`  CYBLE_CONN_HANDLE_T connHandle;`<br>`  CYBLE_GATT_HANDLE_VALUE_OFFSET_PARAM_T * baseAddr;`<br>`  uint8 prepWriteReqCount;`<br>`  uint8 execWriteFlag;`<br>`  CYBLE_GATT_DB_ATTR_HANDLE_T attrHandle;`<br>`  uint8 gattErrorCode;`<br>`} CYBLE_GATTS_EXEC_WRITE_REQ_T` |

The following are detailed descriptions of the changes described in the table, and how they may impact your design:

## Item #1:

In the older versions of the BLE component, the *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event was generated only once when the device received the first Prepare Write Request of a Long Write Value procedure. For responding to the *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event, the CyBle_GattsPrepWriteReqSupport() function should be called by the application to inform the Client if the Server supports Long Writes. This functionality remains in BLE v2.x component.

In BLE v2.x, the CyBle_GattsPrepWriteReqSupport() function should be called each time the device receives the first *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event of Long Write Value procedure. For the Reliable Write Procedure, the *CYBLE_EVT_GATTS_PREP_WRITE_REQ* event is generated for each unique attribute handle, and therefore it requires calling the CyBle_GattsPrepWriteReqSupport() function.

## Item #2:

In the older versions of the BLE component, the *CYBLE_EVT_GATTS_EXEC_WRITE_REQ* event was generated multiple times, and the number of events was dependent on the attribute MTU size and the length of the long attribute. This event contained the burst data of the long attribute, with the length and offset specified in the event parameter structure. When the last *CYBLE_EVT_GATTS_EXEC_WRITE_REQ* was received, the event signaled that the data was actually written to the GATT database.

In the BLE v2.x component, the event is generated once for each Long Write Value procedure, and the event parameter provides the pointer to the start of the buffer where the data is temporarily stored. The data will be written to the GATT database only if there is a successful indication from the user, or if *gattErrorCode* equals to *CYBLE_GATT_ERR_NONE*.

### Item #3:

In the older BLE component versions, the *CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T event* included the *eventParam -> attrHandle* parameter that included the attribute handle of a long attribute value that has been written.

In the BLE v2.x component, this parameter is placed in the following location of the event parameter structure:

*eventParam -> baseAddr[eventParam -> currentPrepWriteReqCount].handleValuePair.attrHandle.*

For detailed description of each element, refer to the CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T section.

### Item #4:

In the older BLE component versions, the *CYBLE_GATTS_EXEC_WRITE_REQ_T* event included the *eventParam -> length* and *eventParam -> offset* parameters. These are respectively equivalent to *eventParam -> baseAddr[n].handleValuePair.value.len* and *eventParam -> baseAddr[n].offset* in the BLE v2.x Component.

The *n* means the number of the burst to which the entire long value is divided. Both the older versions and BLE v2.x components include *eventParam -> attrHandle* parameters. However, in the BLE v2.x component, the parameter has a different purpose. The attribute handle is stored in the *eventParam -> baseAddr[n].handleValuePair.attrHandle* similar to *CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T* struct. In the BLE v2.x component, the *eventParam -> result* was renamed to *eventParam -> execWriteFlag.*

For detailed description of each element, refer to the CYBLE_GATTS_EXEC_WRITE_REQ_T section.

# Component Errata

This section lists known problems with the component.

| Cypress ID | Component Version | Problem | Workaround |
|---|---|---|---|
| 246514 | All | This issue is related to Rx packet for device, which operates in slave role. Some time when there is CRC error, length field of the Rx data packet is corrupt, and actual length is lost.<br>In this case, processing the packet is postponed to the next connection event. This is not expected behavior. | No public workaround available. |

| Cypress ID | Component Version | Problem | Workaround |
|---|---|---|---|
| 210832 | All | Application using IMO to source HFCLK (at 3 MHz) for low power state may lead to CPU not waking up from deep sleep upon disconnection. | As per AN92584 (001-92584 *A), application should use the ECO-sourced HFCLK (at 3 MHz) instead of the IMO.<br>No workaround exists if you insist on using IMO to source the HFCLK instead of the ECO for low power application. |
| 278026 | 3.40 | BLE link may get stuck while sending continuous notifications with slave latency and deep sleep enabled, causing connection timeout on the central side | Use slave latency = 0, or disable quick transmit using the CyBle_SetSlaveLatencyMode() function before sending continuous notifications. |
| 280542 | 3.40 | Link is established by IUT as master. IUT fails to initiate advertisement as slave after link is disconnected. | Call CyBle_Shutdown() API followed by CyBle_StackInit() API to restart the BLE stack when application switches roles. |
| 223246 | 3.0 | Customers using the BLE 3.x component for 4.1 features only will see an increase in Flash by 5 K bytes compared to the previous component versions. | No workaround.<br>The increase is due to enhancements, defect fixes and support for 4.2 features. |

# BLE Stack Changes

This section lists changes made to the BLE Stack.

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 3.5.3.348 | Updated storage manager to re-Calculate CRC for any amount of change to the Flash. | The CRC was not being updated when amount of data written was less than 1 block size (128 bytes), leading to a trigger of CYBLE_EVT_FLASH_CORRUPT event on stack re-initialization. |
| | For DTM TX command, max payload length will be tested based on the BLESS IP version. | For 128K and 256V1 Devices, the maximum payload length is 37 bytes. |
| 3.5.2.344 | Interoperability issue with iphone8/Iphone8Plus. | BLE spec erratum E7791 and 8745 were addressed to resolve this issue |
| | Flash memory consumtion was reduced approxomatly 1 kB. | Enhancement. More flash memory remains for user applications. |
| 3.5.1.340 | The stack will not reject LL_LENGTH_REQ PDU with inconsistent TX Octet and TX time parameter values. | The stack API CyBle_SetDataLength() accepts inconsistent TX Octet and TX time values. BLE Spec 4.2 does not mandate this check, but it was incorporated in the design.<br>BLE Spec 5.0 explicitly requires this check to be omitted. This causes BLE 5.0 compliant controllers to be incompatible with PSoC 4 BLE devices. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 3.5.0.335 | Modified the API for updating the advertisement data and scan response data such that advertisement data and scan response data will not be updated if an update to the same is currently pending. | Multiple calls to update Advertisement data before ADV_CLOSE Event occurs causes ACL TX Buffer to overflow. |
| | Added a fix for handling Read by group request for an attribute whose start handle is disabled in the GATT DB. | IUT responds infinitely to Read by group type request (Primary Service discovery) if the attribute corresponding to start handle sent in read by group type request is disabled in GATT database. |
| | Added a fix in the procedure for handling a scenario where the DUT receives an unknown response for a DLE request. | DLE negotiation failure was observed for the case where the peer device does not support DLE. |
| 3.4.0.326 | Changed GATT attribute permission routine to handle GATT access during pairing process. | GATT characteristic value which need encryption for read and write could be read/written during pairing process. |
| | Global security context is now reset on soft reset and shutdown. | Global security context was not getting reset on calling CyBle_SoftReset() and CyBle_Shutdown() APIs. |
| | Updated the condition to send 'CYBLE_EVT_PENDING_FLASH_WRITE' event if device is previously added to whitelist. | When bonding is completed, GAP module is trying to add device to whitelist. If device is added to whitelist before pairing, controller will reject this request & hence stack is not sending 'CYBLE_EVT_PENDING_FLASH_WRITE' event to application. |
| | Updated the API documentation for CyBle_SetCeLengthParam API. | CyBle_SetCeLengthParam API returns "CYBLE_ERROR_NO_DEVICE_ENTITY" instead of described "CYBLE_ERROR_NO_CONNECTION" |
| | Updated check to report the last service in response even if it does not have any characteristics. | When the last service has no characteristics, it is not reported during service discovery procedure. |
| | Fix handling of GATT Read Multiple Request at server when number of handles is >= 15 | When GATT server receives a read multiple characteristic request from a client with number of handles greater than or equal to 15, it does not send a read multiple characteristic response. |
| | ADV filter policy check is updated to check for whitelist when privacy 1.2 is enabled. | Whitelist filter policy does not work when privacy 1.2 is enabled |
| | Added condition to process pending encryption procedure when connection update procedure is completed. | When controller is waiting for Connection Update instantcontroller pauses the encryption procedure pending during pairing procedure. After connection Update instant occurs, controller is not resuming the pending Encryption procedure. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 3.3.0.309 | An erroneous internal state handling was modified in CyBle_SoftReset () to prevent BLE Stack operation failure. | BLE stack API functions were not operating correctly after calling CyBle_SoftReset(). |
| | Reading GATT-DB is optimized at server side so that response need not be limited to only 3 handle/value/UUID pair rather it should limit to ATT-MTU size. | Better user experience. |
| | At server side GATT-DB read operation in BLE Stack is modified to read GATT-DB after giving “CYBLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ” event to application. | Application can modify the GATT DB in “CYBLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ” event handler so that stack responds with the updated value. |
| | New function -CyBle_GapSetSecurityRequirements added | This function will enable application to restrict pairing based on security requirements like encryption key size, security level etc. BLE Stack will reject pairing if security requirements are not met. |
| | New function - CyBle_GapSetLocalP256Keys() added | To set device’s own P256 public and private keys. |
| | New event - CYBLE_EVT_GAP_SMP_LOC_P256_KEYS_GEN_AND_SET_COMPLETE added | This event gets generated once execution CyBle_GapGenerateLocalP256Keys() is completed. This event contains generated keys which can be used by the peer device for OOB pairing. |
| | Pairing procedure is modified to not to distribute SMP Keys if bonding is not enabled. | SMP Keys were distributed when bonding flag was not enabled. This was resulting in inter-op issue.<br>Some of the device (like iOS) would force disconnect if SMP keys are not distributed at the end of pairing. So it is recommended to enable bonding all the time for such devices. |
| | New enum- CYBLE_HCI_ERROR_T added | This will help application to map the error number returned along with CYBLE_EVT_HCI_STATUS event. |
| | New function CyBle_HciSendPacket() & event CYBLE_EVT_HCI_PKT has been added. | This enhancement allows user to send HCI command and data using this function. This function and event are available only when BLE Component is built for controller only mode with soft transport enabled. |
| | New function CyBle_IsStackIdle() added.<br>Return value of this function indicates to the application whether any transfer queued by the application is completed or not. | One of the use case scenarios where this function is used – in case of OTA after transferring all the OTA packets application needs to know whether all queued packets are transferred before shutting down BLE Stack. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | At Peripheral side when encryption is failed due to LTK loss, CYBLE_EVT_GAP_AUTH_FAILED event with reason code CYBLE_GAP_AUTH_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE is given to application. | Behavior is made uniform between central and peripheral. |
| | CyBle_GattcSignedWriteWithoutRsp() will return CYBLE_ERROR_INVALID_OPERATION if the link is encrypted. | Defect fixing. GATT signed write operation should not be allowed on encrypted link. |
| | Memory related issue: memory used for value field in signed write command at GATT client is freed after processing command. | Defect fixing: Application was not able to send GATT signed write command after 20 iterations. |
| | Using CyBle_L2capCbfcRegisterPsm(), application can register L2CAP COC PSM value with even number also. | L2CAP COC PSM value should not be restricted to odd numbers. |
| | If GATT server receives any of GATT PDU with length of PDU greater than negotiated MTU, then GATT server sends GATT Error Response with reason code Invalid PDU. | When GATT Server received GATT PDU longer than MTU length, server was responding with "Unlikey Error". |
| | New functions CyBle_StartTransmitterTest(), CyBle_StartReceiverTest(), CyBle_TestEnd() have been added. | These functions allow user to perform Direct Test Mode (DTM) operation in SoC mode. |
| | New event “CYBLE_EVT_FLASH_CORRUPT” is added to notify application of data corruption observed during Stack initialization. | Enhancement to detect flash data corruption. |
| | BLE Stack firmware has been modified to handle SupTo in LPM mode when master is out of range | Defect fix. When a supervision timeout was triggered, either by going out of range or by isolating the central, the peripheral got stuck in SysPmSleep and does not give the disconnect event. |
| | Privacy Errata changes as per ESR10 has been implemented.<br>New function CyBle_GapSetPrivacyMode() has been added. | Privacy feature Enhancements.<br>This function allows user to set either device privacy mode or network privacy mode for the peer device. |
| | Limited High Duty Cycle Non-Connectable Advertising feature | Enhancement. Allows user to choose minimum ADV interval >= 20ms for Non-Connectable Undirected ADV & Scannable Undirected Adv type. |
| | New Event CYBLE_ISR_BLESS_ADV_CLOSE has been added | Enhancement. With this application can synchronize its operation to BLE events for better power consumption. |
| 3.2.0.250 | New API added CyBle_RegisterBlessInterruptCallback() to register application routine to monitor CYBLE_ISR_BLESS_CONN_CLOSE_CE event. | Enhancement. Application can register it’s routine with BLE stack to monitor CYBLE_ISR_BLESS_CONN_CLOSE_CE. With this application can synchronize its operation to BLE events for better power consumption. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | New API and Event Added:<br>API - CyBle_SetAppEventMask()<br>Event - CYBLE_EVT_GAP_SCAN_REQ_RECVD | Enhancement.<br>New helps selectively enable/disable newly added events.<br>With the help of event - CYBLE_EVT_GAP_SCAN_REQ_RECVD, application can update Scan Response data after every scan response sent by peripheral. |
| | Modified Low Power Mode functionality to take care of spurious interrupt. | Defect fix. Sometime system was hung due to spurious interrupt in CyBle_ExitLPM(). |
| | Added a new API CyBle_GapDisconnectWithReason() | Enhancement. Provides flexibility to the application to send reason code for disconnecting a LE connection. |
| | Modified CyBle_GappEnterDiscoveryMode() API to allow broadcasting of ADV packet without AD Flag data type when ADV packet is non connectable and all flag bits are zero. | Enhancement to include modification made in “Supplement to Bluetooth Core Specification - CSSv6” Part A, section 1.3. |
| | New API added CyBle_GappSetNumOfAdvPkts(). | Enhancement. Application can send only predefined number of Advertisement packets using this API. |
| | New Event added - CYBLE_EVT_GAP_CONN_ESTB | Enhancement. This event notifies Application when device receives first empty packet in data channel. |
| | API CyBle_GapcCancelConnection() modified to remove the device added during CyBle_GapcInitConnection(). | Defect. It was not possible to connect to more than 5 devices with different BD address when connection was cancelled. |
| | Controller FW is updated to take care of updating Slave Latency in corner condition. | Defect fix. Some time peripheral was not getting into latency mode resulting into higher power consumption. |
| | BLE Stack is modified, not to add a device to resolving list with ID Address as zero. | Enhanced not to add ID Address as zero to the resolving list. |
| | BLE Stack DLE functionality is modified to handle corner condition - when suggested data length set by application is 27 and while processing data length request, BLE Stack was not considering application set suggested data length (i.e. 27). | Defect fix. |
| | New API added - CyBle_GapSetRxDataLength() | Enhancement. Application will be able to set maximum octet that BLE controller can receive. |
| | BLE Stack DLE code is modified to give event CYBLE_EVT_DATA_LENGTH_CHANGED after queuing response to controller. | Defect fix.<br>Event was triggered immediately after device received data length request from peer before queuing response to controller. |
| | API - CyBle_GapAuthReq() in SMP-Master mode is modified to send application specific error code. | Application was not able to send an error in SMP master mode when it receives security request from SMP-slave. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Added a new API CyBle_GenerateAesCmac() to expose AES CMAC engine to application. | Enhancement |
| | Modified CyBle_GattDBGetGroupRangeValpair() API, additional check before length of attribute before writing to GATT-DB. | Defect fix. GATT indication or notifications after GATT-Write of large size packet, some time resulting in system hang. |
| | API - CyBle_GattcReadCharacteristicValue() modified to return attributes value if the connection handle is of type primary service declaration. | Modified to keep consistency between attribute type of 16-bit and 128-bit UUID. |
| | Modified GATT DB for each attribute to have separate permissions for read and write operations. | Enhancement. By default each attribute will have the same permission for both read and write operation. Application can change these permissions for read and write operation. |
| | The event CYBLE_EVT_L2CAP_CBFC_DATA_WRITE_IND is notified to application after transferring CBFC data to controller instead of notifying after queuing to L2CAP module. | Enhancement. When all memory inside BLE stack was consumed to receive packet then application was getting memory allocation failure all the time and CYBLE_EVT_STACK_BUSY_STATUS was not triggered. Now with the combination of CYBLE_EVT_STACK_BUSY_STATUS & CYBLE_EVT_L2CAP_CBFC_DATA_WRITE_IND, application will know when to queue next packet for transmission. |
| 3.1.0.194 | Modified the L2CAP CBFC flow control algorithm such that the CyBle_L2capChannelDataWrite() API doesn't give CYBLE_ERROR_MEMORY_ALLOCATION_FAILED error when the ratio of transmit packet length to MPS size is more than 8. | Data transfer was failing due to insufficient memory when the application sent L2CAP data with the following L2CAP configuration: MTU = 512 and MPS = 23. |
| | Modified the BLE Stack to store Local LTK in retention memory. | The application is easier to use by not being required to keep a copy of LTK & IRK in retention memory, because this is done within stack.<br>Application was storing LTK, IRK, and bdHandle in retention memory and was setting these keys after the BLE Stack was on. |
| | Modified the CyBle_GapGetChannelMap() API to give correct channel map value. | Defect fix. CyBle_GapGetChannelMap API was giving incorrect value. |
| | Added new CyBle_GapFixAuthPassKey() API to enable the application to set or clear the pass-key. | Enhancement. Provides flexibility to modify fixed pass-key instead of stack generating random number every time. |
| | Updated BLESS to go to Deep Sleep even if connection parameter / channel map update procedure is in progress. | Improved power consumption even during LL-control procedures like connection establishment and channel map update. |
| | Added new CyBle_SetSeedForRandomGenerator() API. | Provides flexibility to the application to improve randomness. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Updated the description for the CyBle_EnterLPM API to describe the clock switching procedure between ECO and IMO if the application is using ECO for non-BLE functions. | Gives more clarity. |
| | Modified the CyBle_StoreAppData() API to return CYBLE_ERROR_FLASH_WRITE_NOT_PERMITED when flash write failed due to wrong Flash address. | API CyBle_StoreAppData() was returning CYBLE_ERROR_OK for invalid Flash address. |
| | Modified the GATT long procedures to give CYBLE_EVT_GATTC_LONG_PROCEDURE_END event in the following condition also – ATT handle is equal to end handle of the request. | In long procedures if received response contain end handle of the request then CYBLE_EVT_GATTC_LONG_PROCEDURE_ END event was not raised by BLE Stack. |
| | Defines SMP_SC_PAIR_PROP_MITM_MASK and SMP_SC_PAIR_PROP_KP_MASK are mapped to CYBLE_GAP_SMP_SC_PAIR_PROP_MITM_MASK and CYBLE_GAP_SMP_SC_PAIR_PROP_KP_MASK respectively. | Modified for more readability. |
| | Modified the BLE Controller module to not decrypt non encrypted terminate packet. | BLE controller was decrypting non-encrypted terminate packet when encryption was in progress. Due to this peripheral was not getting disconnected. |
| | Modified the handling of Find Information Request to handle Primary Service of type 128-bit UUID. | GATT-Server was not giving 128 Bit Service UUID if handle of 128 bit service UUID lies between start and end handle of Find by Information request. |
| | PDU timer in BLE Controller is killed once BLESS gives device disconnect interrupt. | If device connected and disconnected continuously some time API CyBle_GapDisconnect() takes long time to disconnect. |
| | Modified the DLE negation logic to consider Max Data Len as 27 when suggested data length is 27. | When suggested data length is 27 and during negotiation device receives Max Data Len greater than 27 and less then Max Supported Len then device consider negotiated Data Len as Max Data Len instead of suggested data length (i.e. 27) |
| | Modified the CyBle_GapGetDataLength() API to return correct value. | API CyBle_GapGetDataLength() was not returning correct data in “readParam” parameter. |
| 3.0.0.153 | Modified the BLE Stack ISR to ignore SCAN_INTR interrupt if scan operation is stopped by application. | Due to race condition between application stopping scan and BLESS raising SCAN_INTR interrupt was causing BLE Stack FW to read invalid data from FIFO while processing advertisement packet. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Enhanced the BLE Stack to configure the queue depth for the prepare write command. | Because the write queue depth was fixed, it was not possible to execute the prepare write command if the maxAttribLength is greater than 10 times the ATT MTU size. |
| | Implemented the ECDH algorithm such that, at the end of each stage, the BLE stack can process commands from the master, for example, the channel map update. | The ECDH algorithm execution takes about 3 seconds. During this time no commands from the master could be processed by the peripheral. This resulted in an inter-op issue. |
| | Modified the BLE Stack to handle invalid offset (0xFFFF). | The read long characteristic was timing out with invalid offset (oxFFFF). |
| | Changed all 4.2 APIs that passed a pointer as an input to the stack to a constant. | To avoid the application being modified within the BLE stack. |
| | Added new API CyBle_SetSlaveLatencyMode API. | This API was added to override the Slave latency setting so that data is transmitted quickly even when slave latency is enabled. |
| | Modified CHANNEL_MAP_UPDATE PDU handling for improved power consumption. | Improved power consumption in the system where frequent channel map updates take place. |
| | Fixed a memory leak issue observed during device disconnect when active data transfer is in progress. | Fixed a defect. |
| | Updated the CyBle_SetCeLengthParam API such that, at the time of connection creation, the CE length is set to Maximum available length and application would modify CE length upon CONN_UPDATE event. | Enhanced to support CE Length configuration during run time. |
| | Optimized the BLE Stack to get better throughput. | Throughput optimization. |
| | Modified the BLE Stack to give only one CYBLE_EVT_GAP_DATA_LENGTH_CHANGE event when the length update procedure is initiated by both master and slave. | Two CYBLE_EVT_GAP_DATA_LENGTH_CHANGE events were received by the application when the length update procedure was initiated by both master and slave. |
| | Added eew CYBLE_EVT_GATTC_LONG_PROCEDURE_END event to notify completion of discover characteristic by UUID procedure. | Application could not know the completion of discover characteristic by UUID procedure. |
| | Reinitialized some variables after shutdown. | Fixed defect. |
| 3.0.0.103 | Enhanced BLE Stack to support BLE 4.2 features:<br>LE Secure connection<br>LL Privacy<br>LE Data Length Extension | Enhancement. New BLE 4.2 features implementation. |
| | New CyBle_GattcDiscoverPrimaryServices API added. | Enhancement. It was not possible to discover a partial data base using the existing CyBle_GattcStartDiscovery API. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Internal L2CAP queue elements are freed after device disconnects. | Defect fix. While the application is continuously transmitting data packets, if the peer device gets disconnected, then the internal L2CAP queue elements were not freed. This resulted in a failure to establish a connection. |
| | CyBle_GattsNotification API is modified to return CYBLE_ERROR_MEMORY_ALLOCATION_FAILED when memory was not available. | Defect fix. CyBle_GattsNotification API was returning CYBLE_ERROR_INVALID_OPERATION instead of CYBLE_ERROR_MEMORY_ALLOCATION_FAILED when memory was not available. |
| | Modified stack to reserve memory for ATT/GATT response handling when a peripheral is continuously transmitting data (notification / indication). | Defect fix. When the application continuously transmits data using notification or indication, all the BLE Stack memory was consumed for transmitting data. This resulted in no memory available for responding to a new request. This meant no response was sent for a request when a continuous notification was in progress. |
| 2.3.0.46 | Updated internal operation of the CyBle_GappStopAdvertisement() API to wait on BLESS hardware ADV_ON_STATUS bit until advertising is actually stopped. It is done to reflect integrated “Advertising Status” for BLESS hardware and BLE Stack to support correct ADV stop operation to support all different IMO and BLESS frequency ranges. | BLESS DSM entry was not happening when a device advertisement of type high duty cycle ADV_DIRECT_IND was stopped by the application and the CPU was running at 7 MHz or less frequency. |
| | Updated description of GapcSetHostChannelClassification() API.<br>Updated the HCI event handler function to return HCI Status event to application, when invalid parameters are passed to the function. | API description was not clear enough to use this API. Host was not returning the HCI status event for invalid input parameters. |
| | Updated the description for the CyBle_L2capChannelDataWrite() API.<br>BLE Stack will return error code ‘CYBLE_ERROR_INVALID_PARAMETER’ when data input size is higher than permitted in the channel. | ‘CYBLE_ERROR_INVALID_PARAMETER’ error code in more accurate than default ‘CYBLE_ERROR_MAX’ for this condition. |
| | Changed a default random address to Static random address in the BLE configuration data file. | The default random address, returned by the Stack, did not meet the criteria for a random address. Note that application is expected to set the random address and not use a default random address. |
| | All References to MTU in BLE Stack header files are replaced with either GATT MTU or L2CAP MTU explicitly. | MTU is used for both ATT and L2CAP MTU references. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Removed ‘CYBLE_ERROR_NO_DEVICE_ENTITY’ error code from CyBle_GapRemoveOldestDeviceFromBondedList() API Description. | ‘CYBLE_ERROR_NO_DEVICE_ENTITY’ error code is never returned by BLE Stack. |
| | Added descriptions for the following ENUM definitions:<br>CYBLE_EVT_HOST_INVALID<br>CYBLE_BLESS_PWR_LVL_T<br>CYBLE_BLESS_ECO_CLK_DIV_T | Provide meaningful description to ENUMs. |
| | SMP FSM handler was modified to update negotiated authentication parameters to authenticated property, if OOB is used. | Core v4.1, Vol 3, Part H, Section 2.3.5.1<br>"If the out of band authentication method is used the key is assumed to be Authenticated MITM Protection." |
| | Changed number of bits used to generate random number for passkey display from 16 bits to 20 bits. Change is made in SMP FSM handler. | Passkey generated to display was never larger than 65535. As per spec (Core v4.2, Vol 3, Part C, Section 3.2.3.3) value should be between 000000 – 999999. |
| | BLE Stack updated to filter duplicate “scannable unidirect” type of advertising packets. | BLE device continuously receives Advertisement report if Filter Policy is set to "Scan Request: White list" |
| | Documentation update for the following functions:<br>CyBle_EnterLPM()<br>CyBle_ExitLPM()<br>CyBle_ProcessEvents() | Documented usage of CPU Sleep Mode in BLE Stack while BLESS force exit is issued by BLE Stack when BLESS is in BLE Deep Sleep Mode. More clarity is added in all BLESS Low Power Mode APIs as call to CyBle_ProcessEvents(), CYBLe_ExitLPM() can cause BLESS force DSM exit. |
| | Updated the BT Timer code to handle timer creation with any order of timeouts. | If two timers were started simultaneously, the timeout didn’t happen as per timeout provided. If a second timer was started with a lesser timeout thsn first, then second timer did not work as expected. |
| 2.2.0.36 | Updated Synth Delay to interop with HP laptop and to avoid the extra modulated stream on '0's. | Touch Mouse was not able to establish connection with HP laptop which has Ralink RT3290 BLE4.0 Chipset. |
| | Clear the disconnection status in LL Connection Entity every time upon CONN_FAILED interrupt.<br>UNSPECIFIED ERROR passed to application for any other possible corner case for application to remain in synch with BLE Component/Stack. | Sometime application never receives disconnect event even when the peer device is powered off. |
| | Changed the sequence of enabling/disabling interrupts during SCAN start and SCAN stop to avoid race condition where high priority interrupts occur to avoid SCAN FIFO becomes full. | Central hangs in scanning mode when lots of devices are advertising. GAPC_SCAN_PROGRESS_RESULT event is never generated. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.1.0.30. | Updated existing interface between BLE component and BLE stack CyBle_StackInit() API for providing the flash address for storing the information with respect to bonding | To allow to retain the information of bonding when application is updated using OTA. |
| | Reduced the BLE stack start up time by removing the delay of 10ms used for FPGA. Reducing redundant HCI command exchanges between the Host and Controller layer during initialization in SoC mode | Reduced the BLE stack start up time which reduces the power during initialization. |
| | Dynamic memory usage within BLE Stack is optimized. | Effective RAM utilization |
| | Enhancement to register multiple L2CAP PSM specified during the BLE stack initialization. | Enhancement |
| | Memory corruption due to out of bound copying during read request is fixed. | Defect fix. |
| | Defect fixed to enable retrieving SMP keys using IDADDR. | Privacy 1.1: Device was not able to identify the device when connected with a public address which was previously Bonded with random address. |
| 2.0.0.81 | Removed autonomous initiation of VERSION_EXCHANGE after connection establishment from BLE Stack. | Resolves the interoperability issue with MI4 phone Bluetooth host. No impact to existing functionality. |
| | Added configurability for optimal RAM usage and consequently updated following.<br>Updated existing interface between BLE Component and BLE Stack for CyBle_StackInit() API<br>Added CyBle_L2capSetConfig() API | Added configurability for optimal RAM usage in BLE Component and Stack based on application configuration/requirement for usage of MTU and L2CAP features.<br>CyBle_L2capSetConfig() API is added to configure the BLE Stack for following L2CAP configuration:<br>Total dynamic channels (CIDs) required by application.<br>Total number of Credit Based Flow Control (CBFC) Protocol Service Multiplexing (PSM) channels required.<br>L2CAP Signaling transactions related timeout |
| | Updated handling of “CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT” in to filter and do not propagate advertising reports of type ADV_DIR_IND. | ADV_DIR_IND shall be sent to application only during observation procedure. This was being sent when the device is performing Limited or General Discovery procedure. |
| | Updated CyBle_GapcSetRemoteAddr() API | CyBle_GapcSetRemoteAddr was failing on subsequent call when same peer device changes its address between public and random. API is the updated to fix the issue. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Updated CyBle_GapRemoveOldestDeviceFromBondedList() API. | The oldest device from the bond list was not getting removed from retention memory. It was only getting removed from RAM.<br>Added error code return value CYBLE_ERROR_NO_DEVICE_ENTITY to caller API in case where no device is present in bond list and the API is invoked. |
| | Updated CyBle_GapcResolveDevice() API | The CyBle_GapcResolveDevice() API had a side-effect, as the value of the input parameter identity resolution key “uint8 *irk” was getting changed after API execution. |
| | Updated CyBle_SetTxPower API | The API is changed for user convenience to avoid the value change of input parameter “CYBLE_BLESS_PWR_IN_DB_T *bleSsPwrLvl” after API execution. |
| | Updated handling of internal low power operation when simultaneous operation for ADV, CONN and SCAN is in progress. | Updated the internal low power operation for CONN to sustain when non-connectable ADV or passive SCAN is going on. |
| | Added event “CYBLE_EVT_GATTC_STOP_CMD_COMPLETE”<br>Updated internal handling of GATT stop procedure to propagate “CYBLE_EVT_GATTC_STOP_CMD_COMPLETE” to application. | Added event “CYBLE_EVT_GATTC_STOP_CMD_COMPLETE” to indicate CyBle_GattcStopCmd() API operation is complete. |
| | GATT Database is enhanced to support varying length characteristic at run time. | Upcoming profile application such as User Data Service (UDS) require supporting varying length characteristic.<br>Previous approach had current attribute length store in FLASH and hence prevented run time modification. |
| | Updated BLE Stack to give timeout event CYBLE_EVT_TIMEOUT correctly for discovery procedure or observation procedure. | Observation procedure timeout did not occur after step 3:<br>Connect with peer device and start any GATT procedure (MTU Exchange).<br>Disconnect from peer device<br>Start observation procedure with timeout |
| | Bonded device list handling is updated for clearing bond device operation. | Sixth time connection was failing after following steps are performed for 5-6 times<br>Change local device address<br>Connect and bond with peer device<br>Disconnect and clear bonding info |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Updated BLE Stack to return CYBLE_ERROR_INVALID_PARAMETER when GATT write operation with invalid length is performed. | Error code "CYBLE_ERROR_INVALID_PARAMETER" was not given when GATT write characteristic operation was performed with invalid length with respect to set MTU size. |
| | L2CAP module modified to fix memory leak | Memory leak in L2CAP credit based flow control (CBFC) data path is fixed |
| 1.0.0.184 | Updated the CyBle_GattcDiscoverCharacteristicByUuid API to achieve characteristic discovery with 128-bit UUID using this API. | Defect fix |
| | Optimized the BLE Stack to reduce the system power consumption for BLE solutions. | Power optimization for BLE solutions |
| | Corrected the GATT server access error code when the attribute is not found. | Defect fix |
| | Provided more clarification for CYBLE_EVT_STACK_BUSY_STATUS event handling. | Better user experience. |
| 1.0.0.181 | Update internal device settings. | Fix for BLE RF link (transmit/receive) issues observed on some devices. Increase of ~0.3 mA on Rx current. |
| 1.0.0.169 | 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 |
|---|---|---|
| 3.53 | BLE Stack was updated to version 3.5.3.348. | See BLE Stack Changes. |
| 3.52 | BLE Stack was updated to version 3.5.2.344. | See BLE Stack Changes. |
| 3.51 | BLE Stack was updated to version 3.5.1.340. | See BLE Stack Changes. |
| 3.50 | Added storing the CCCD value in flash memory for the Tx Power Level Characteristic (TPS). | BLE specification requirement. |
| | Enabled editing the Profile configuration on the Profiles tab in the OTA Stack only mode. | Flexibility. |
| | BLE Stack was updated to version 3.5.0.335. | See BLE Stack Changes. |
| 3.40 | Added support for the Pulse Oximeter Profile and Service. | New feature. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Updated Wireless Power Profile to version 1.3 of the Cycling Power Profile specification. | Support the new specification version. |
| | Added event CYBLE_EVT_GATTC_DISC_SKIPPED_SERVICE, which indicates that the service (not defined in the GATT database) was found during the server device discovery. | New feature. |
| | Updated Cycling Power Service to version 1.1 of the Cycling Power Service specification. | Support the new specification version. |
| | BLE Stack was updated to version 3.4.0.326. | See BLE Stack Changes. |
| 3.30.a | BLE Stack changes section update. | See BLE Stack Changes. |
| 3.30 | Added support for the Automation IO Profile and Service. | New feature. |
| | Added HCI over software transport mode. | New feature. |
| | Added possibility to delete mandatory characteristics and descriptors in the customizer. | Flexibility. |
| | Customizer read-only security parameters were removed from the Security node of the GAP Settings tab.<br>Enable Link Layer Privacy parameter was moved to the General node of the GAP Settings tab. | To refine component usability. |
| | The Write permission for characteristics is set when any of the following Properties are checked:<br>- Write<br>- WriteWithoutResponse<br>- SignedWrite<br>- ReliableWrite.<br>Previously it was set only for the Write and WriteWithoutResponse Properties only. | BLE specification requirement. |
| | The Strict Pairing option was enabled in the GUI. | This feature allows the device to use specified security settings during pairing or reject pairing request if the security requirements are not satisfied. |
| | The following functions were added<br>CyBle_HciUartTransportEnable();<br>CyBle_HciSoftTransportEnable();<br>CyBle_HciSendPacket(). | This was done to allow sending of HCI commands to BLE Stack. |
| | BLE Stack was updated to version 3.3.0.309. | See BLE Stack Changes. |
| 3.20 | Added support for the Indoor Positioning Service. | New feature. |
| | Added possibility to modify UART Flow Control settings in the HCI mode. | Flexibility. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Added possibility to generate 128-bit UUIDs for custom services, characteristics and descriptors. The default 128-bit UUIDs are generated randomly for each custom service, characteristic and descriptor. | New feature. |
| | BLE Stack was updated to version 3.2.0.250. | See BLE Stack Changes. |
| 3.10 | Added a check box that allows enabling / disabling the automatic update of the characteristics and descriptors security permissions after a GAP Security settings change. | New feature. |
| | Added a check box that allows enabling / disabling the L2CAP logical channels functionality. | It enables configuration of the L2CAP logical channels. |
| | Updated the Continuous Glucose Monitoring Service conditional characteristics fields according to the specification. | To conform with the Continuous Glucose Monitoring Service requirements. |
| | BLE Stack was updated to version 3.1.0.194. | See BLE Stack Changes. |
| 3.0 | Added support for BLE 4.2 Stack protocol to the component | New feature-support added.<br>**Note** The BLE component 3.0 supporting BLE 4.2 is provided as Beta Level for early design starts. For all other MPN users, Cypress recommends continuing to use BLE component version 2.30 or earlier. |
| | Added support for the HTTP Proxy Service to the component. | New feature-support added. |
| | Added TX power level validation in the customizer. In case when one of the TX power levels on the GAP tab equals 3 dBm and other isn't, an error icon is shown. | This was done because of internal limitations for a TX power settings. |
| | The CyBle_GapRemoveBondedDevice() was added to the component. | The function allows removing the bonding information of the device including CCCD values. |
| | The CyBle_GattcStartPartialDiscovery() was added to the component. | The function allows partial service discovery of the remote device |
| | Internal function CyBle_IsDeviceAddressValid() was made public. | The function is used to verify if a public device address is programmed to flash memory |
| | Added pa_en output terminal and Enable external Power Amplifier field on the Advanced tab of the BLE customizer. | To enable connection of a high active external power amplifier to the device. |
| | Advanced tab was added to the component customizer GUI. | New feature-support added. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Added the implementation of a GATT Server role to the GATT Client devices.<br>In order to enable GATT Server role for the existing GATT Client configurations, you need to do the following steps: 1) Open the customizer. 2) On the General tab, open the Profile role combo box and re-select the currently selected GATT role item (without switching between the Profile role items). | BLE specification requirement |
| | BLE Stack was updated to version 3.0.0.153. | See BLE Stack Changes. |
| 2.30 | Added validation of the TX power level in the component GUI. 3 dBm value can be set only for both Adv/Scan TX power level and Connection TX power level simultaneously. | Hardware limitations. |
| | The new QD ID and Declaration ID# for BLE component Profiles were added in the table of Bluetooth Qualification section | New QD ID and Declaration ID# were introduced to include qualification details about UDS, WSS, WSP, BCS and CTS (v1.1). |
| | The generation of an erroneous value length for a Custom Descriptor with 32-bit or 128-bit UUID was fixed. | In case when 32-bit or 128-bit UUID was used for the Custom Descriptor and BLE device was acting as a GATT Server, a wrong Descriptor UUID and value length were generated by the component. |
| | Updated the CyBle_NdcssGetCharacteristicValue() and CyBle_RtussGetCharacteristicValue() functions. They were always returning CYBLE_ERROR_INVALID_PARAMETER. | The reason for this was an incorrect condition check that was done after the value was written to the GATT database. |
| | Updated the following services: HIDS, SCPS, ESS, BMS, UDS, CTS.<br>In cases of security mode usage, where pairing is required, these services were generating WRITE CHARACTERISTIC/DESCRIPTOR, NOTIFIFICATION or INDICATION ENABLED/DISABLED events even though the device wasn't paired. Also, the data wasn't written to the GATT DB. | Due to erroneous code, the events were generated prior to checking security settings. |
| | BLE Stack was updated to version 2.3.0.46. | See BLE Stack Changes. |
| 2.20 | Support of the following profiles/services was added to the component:<br>Apple Notification Center Service (ANCS)<br>Body Composition Service (BCS)<br>Bootloader Service (BTS)<br>User Data Service (UDS)<br>Weight Scale Profile (WSP)<br>Weight Scale Service (WSS) | New feature-support added. |
| | BLE Code Snippets feature description added. | New feature-support added. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | A defect in the Current Time Service was fixed. The optional write permission of the Current Time and Local Time Information characteristics are now controlled by the corresponding permission flags in the BLE component customizer GUI. | In previous BLE component versions, the Current Time and Local Time Information characteristics were always writable regardless of permission flag settings.<br>If you write the Current Time and/or Local Time Information characteristics in your projects, make sure to update the corresponding permission flags properly, because by default the optional write permission is disabled. |
| | BLE stack was updated to version 2.2.0.36. | See BLE Stack Changes. |
| 2.10.a | Added the Component Errata section | Document known issues. |
| 2.10 | Support of the Wireless Power Transfer (WPT) Profile was added to the component. | New feature-support added. |
| | BLE stack was updated to version 2.1.0.30. | See BLE Stack Changes. |
| 2.0.a | Minor datasheet edits. | Fixed several typos. |
| 2.0 | Support of the following profiles was added to the component:<br>• Environmental Sensing Profile (ESP)<br>• Continuous Glucose Monitoring Profile (CGMP)<br>• Bond Management Service (BMS)<br>• Internet Protocol Support Profile (IPSP) | New feature-support added. |
| | Changed long write and reliable write procedures.<br>Refer to the Updating to v2.x section for more information on the design impact of this change. | The component addresses a defect, where the application did not have the option to validate the data and only one prepare write event and multiple execute write events were going to the application.<br>User impact:<br>This change may have backward compatibility issues for some designs. The details are described in the Updating to v2.x section.<br>The following structures are modified:<br>'CYBLE_GATTS_PREP_WRITE_REQ_PARAM_T'<br>'CYBLE_GATTS_EXEC_WRITE_REQ_T' |
| | Updated CyBle_StoreBondingData API description. New BLE device with 256K of Flash memory is not affected by modification of the clock settings. | New flash memory type doesn't require clock settings modification. |
| | BLE stack was updated to version 2.0.0.81 | See BLE Stack Changes. |
| 1.20 | Improved TX power level performance for +3 dBm option. | +3 dBm Tx Power level had no effect compared to 0 dBm |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Fixed Advertising Channel Map bit mask for "Channel 39" and "Channels 37 and 38" items. | Advertising Channel Map bit masks generated for "Channel 39" and "Channels 37 and 38" items were swapped. |
| | Changed the functions CyBle_CscssGetCharacteristicDescriptor() and CyBle_RscssGetCharacteristicDescriptor() to use CyBle_GattsReadAttributeValue() instead of CyBle_GattsWriteAttributeValue(). | This corrected the functions that were not working. |
| | For Health Thermometer Service the “Out of Range” error code changed from 0xff (defined by Supplement to Bluetooth Core Specification) to 0x80 which is defined by HEALTH THERMOMETER SERVICE specification. | The change was made to bring the implementation in accordance with the Health Thermometer Service specification. |
| | Added CyBle_ChangeAdDeviceAddress API to update the Bluetooth device address in the advertisement or scan response data structure.<br>Added CyBle_GattGetBusyStatus API description in datasheet | Device address was not updated in advertisement packet when silicon generated option selected in customizer. |
| | Fixed scanning state in Central role to reflect the customizer selection. | BLE Scan Type was always set to active scan. |
| | Extended values input range for several characteristics to include "Unknown" value:<br>- Time Zone<br>- DST Offset<br>- Day of Week | Characteristics for CTS did not allow 'Unknown' settings |
| | Simplified the usage of CyBLE_GapUpdateAdvData API. Now CyBLE_GapUpdateAdvData API works in all BLESS states. | Better user experience. |
| | BLE stack was updated to version 1.0.1.184 | See BLE Stack Changes. |
| 1.10 | BLE Stack was updated to version 1.0.0.181. | See BLE Stack Changes. |
| 1.0.b | Support of the following profiles was added to the component:<br>• Phone Alert Status Profile (PASP)<br>• Location and Navigation Profile (LNP)<br>• Cycling Speed and Cadence Profile (CSCP)<br>• Cycling Power Profile (CPP) | New feature-support added. |
| | The CYBLE_L2CAP_COMMAND_REJ_REASON_T event was renamed to CYBLE_EVT_L2CAP_COMMAND_REJ. | The event was renamed to be consistent with other event name formats. |
| | The CYBLE_EVT_GAP_RESOLVE_PVT_ADDR_VERIFY_CNF event was removed. | The event became obsolete. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | The following members of the `CYBLE_API_RESULT_T` structure were deprecated:<br>`CYBLE_ERROR_GATT_DB_INVALID_OFFSET,`<br>`CYBLE_ERROR_GATT_DB_NULL_PARAMETER_NOT_ALLOWED,`<br>`CYBLE_ERROR_GATT_DB_UNSUPPORTED_GROUP_TYPE,`<br>`CYBLE_ERROR_GATT_DB_INSUFFICIENT_BUFFER_LEN,`<br>`CYBLE_ERROR_GATT_DB_MORE_MATCHING_RESULT_FOUND,`<br>`CYBLE_ERROR_GATT_DB_NO_MATCHING_RESULT,`<br>`CYBLE_ERROR_GATT_DB_HANDLE_NOT_FOUND,`<br>`CYBLE_ERROR_GATT_DB_HANDLE_NOT_IN_RANGE,`<br>`CYBLE_ERROR_GATT_DB_HANDLE_IN_GROUP_RANGE,`<br>`CYBLE_ERROR_GATT_DB_INVALID_OPERATION,`<br>`CYBLE_ERROR_GATT_DB_UUID_NOT_IN_BT_SPACE,`<br>`CYBLE_ERROR_GATT_DB_INVALID_ATTR_HANDLE,`<br>`CYBLE_ERROR_GATT_DB_INSUFFICIENT_SECURITY,`<br>`CYBLE_ERROR_GATT_DB_INSUFFICIENT_ENC_KEY_SIZE,`<br>`CYBLE_ERROR_GATT_DB_INVALID_INSTANCE,`<br>`CYBLE_ERROR_GATT_DB_INCORRECT_UUID_FRMT,`<br>`CYBLE_ERROR_GATT_DB_UUID_FRMT_UNSUPPORTED,`<br>`CYBLE_ERROR_GATT_DB_TYPE_MISMATCH,`<br>`CYBLE_ERROR_GATT_DB_INSUFFICIENT_ENCRYPTION,`<br>`CYBLE_ERROR_L2CAP_NOT_ENOUGH_CREDITS` | The elements weren't used as return values in any of the API functions. |
| | Removed WDT from the BLE Component. | In the preliminary release of the BLE Component, the protocol procedure timeout functionality was implemented using the WDT. For the production release, the Component was optimized to use the BLESS Link Layer timer. |
| | Edits to the datasheet. | Update Configure dialog screen captures.<br>Added the APIs into the datasheet.<br>Added Unsupported Features section.<br>Added characterization data.<br>Addressed all Errata from the preliminary version of the datasheet and removed the section. |
| 1.0.a | Edits to the datasheet. | Added sections to describe WDT counter and interrupt.<br>Clarified descriptions for several APIs and GUIs.<br>Added Errata section.<br>Moved API documentation to separate CHM file.<br>Updated Functional Description section. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.0 | Initial document for new Component. | |
| | Initial BLE Stack version 1.0.0.169. | |

© Cypress Semiconductor Corporation, 2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

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.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.