UM1770
User manual

BlueNRG, BlueNRG-MS profiles application interface

Introduction

This document describes the API interfaces and related events of the Bluetooth low energy (BLE) profiles peripheral and central roles.

These APIs allow the management of communication between a user application and the available Bluetooth low energy profiles.

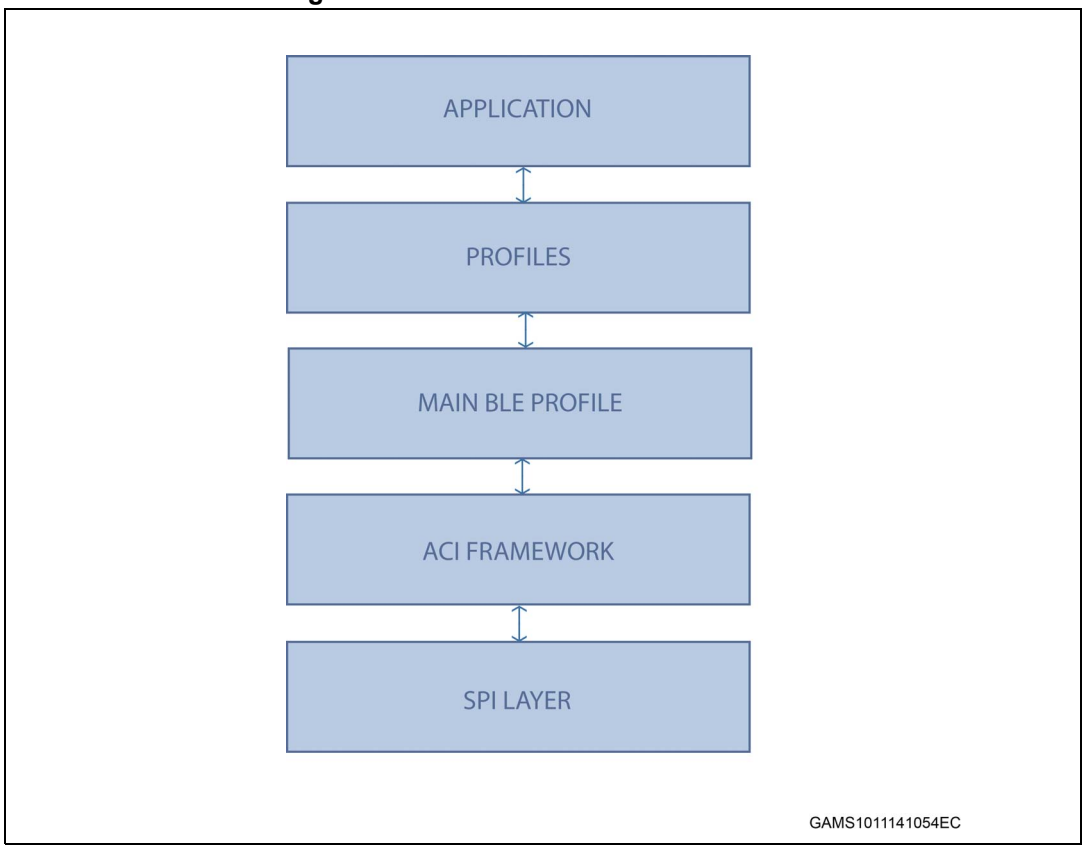
Note: The document content is valid for both BlueNRG and BlueNRG-MS devices. Any reference to BlueNRG device is also valid for the BlueNRG-MS device. Any specific difference is highlighted whenever it is needed.

Contents

1 Architecture

Figure 1 describes the profiles framework architecture:

Figure 1. Profiles framework architecture



The following is a description of each profile layer:

- Application:
  - user/test applications using Bluetooth low energy (BLE) profiles framework
- Profiles:
  - specific profile implementation (alert notification, find me, etc.)
- Main BLE profile:
  - main/common BLE profile framework for all BLE profiles
  - it provides functions for main profile initialization, profile registration, event handlers and notifies events to specific profiles and APIs for device discovery, device connection, services and characteristics discovery for central roles.
- ACI framework:
  - it exposes functions to the upper layers to send the various commands supported by the BlueNRG, BlueNRG-MS device (standard HCI and vendor specific ones)
  - all the commands are sent to the controller via the ACI framework (bluenrg_gap_aci.c, bluenrg_gatt_aci.c, bluenrg_l2cap.c, bluenrg_hal_aci.c).

Note: A specific profile may only require a subset of these commands (automatically stripped by the linker).

- SPI Layer
  - SPI layer APIs (read/write from/to BlueNRG, BlueNRG-MS SPI buffers)

Note: No multiple profiles are supported at the same time.

2 ACI framework

All the BlueNRG commands are sent to the controller via the ACI framework. The ACI framework exposes functions to the upper layers to send the various commands supported by the BlueNRG and to get the events raised from the BlueNRG network coprocessor.

The ACI framework implements the ACI APIs according to the Bluetooth LE stack application command interface APIS defined on the UM1755 and UM1865 user manuals on References Section.

Three types of events are handled within the profile framework:

1. BlueNRG events raised by the BlueNRG network coprocessor (refer to UM1755 and UM1865 user manuals on *Section 8: List of references*);

2. General profile events which are used by all peripheral profiles and are not specific to any profile (refer to *Section 5.1: Generic events*);

3. Profile specific events defined by each profile.

3 Execution context

This section describes the profile peripheral and central execution contest.

3.1 Peripheral roles

The BlueNRG, BlueNRG-MS profiles peripheral framework implements a single task model for execution. The entire processing takes place in the ISR context and the main thread context. The execution context is a while(1) loop.

The following functions are processed in this loop:

1. HCI_Process(): it performs the processing of any pending events read. It is defined on file hci.c (ACI Framework);
2. Profile_Process_Q(): it sends the commands during the initialization or pairing process and updates the main profile state machine.It is defined on file profile.c (main profile file);
3. Profile specific state machine: it is called for checking current main profile and profile state, substate and performing related actions and consequent status updates (functions *_StateMachine(void) on each specific profile);
4. Application state machine: it performs application-specific handling (sending data to profiles, enabling advertising, displaying to the user, etc.).

The PTS profile validation application (profiles_test_application.c) provides an example of such processing.

The BlueNRG events are notified to the main profile application (profile.c) through the HCI_Event_CB() callback which performs the required actions based on main profile state and substate. Further the HCI_Event_CB() function notifies the BlueNRG events to a specific profile by calling the profile callback function * _Event_Handler(). This function is called with the following instruction:

```
gMainProfileCtxt.bleProfileApplNotifyFunc(appNotifEvt,1,&appNotifStatus);
```

Each profile registers its event handler function (* _Event_Handler()) through the profile *_Init() function.

Following are more details about the main profile framework (profile.c) key functions:

- BLE_Profile_Init(): It initializes the main profile.
- BLE_Profiles_Evt_Notify_Cb(): it is the main profile callback function which is called by each profile to notify the profile specific events to the main profile application. Based on such events, main profile application can decide which actions to take in order to handle the specific profile events. This function is provided to each profile when the related *_Init() API is called.
- BLE_Profile_Register_Profile(): it allows registration of each profile callback; it is called within the profile initialization function named: *_Init() function.

Profile specific example: heart rate profile

- Heart rate profile header file: heart_rate.h file;
- Heart rate profile initialization function:HRProfile_Init(). This function performs the following operations:
  - Set the main profile callback function for notifying profile specific events to the main profile application;
  - Set the heart rate profile HRProfile_Rx_Event_Handler() callback function;
  - Register the heart rate with BLE main Profile.

The HRProfile_Init() function is called on the profile main application as follows:

```
if (HRProfile_Init((uint8_t)0xFF,
                   BLE_Profiles_Evt_Notify_Cb,
                   0x04) == BLE_STATUS_SUCCESS)
{
   APPL_MESG_DBG(profiledbgfile,"Initialized Heart Rate Profile \n" );
}
```

- Heart rate profile callback function: HRProfile_Rx_Event_Handler(). This function allows Heart Rate profile to properly handling the BlueNRG events according to the profile state.

3.2 Central roles

The BlueNRG, BlueNRG-MS profiles central framework implements a single task model for execution. The entire processing takes place in the ISR context and the main thread context. The execution context is a while(1) loop.

The following functions are processed in this loop:

1. HCI_Process(): it performs the processing of any pending events read. It is defined on file hci.c (ACI Framework);
2. Master_Process(): it processes the events related to the device discovery, connection, service, characteristics, characteristics descriptors, read, write, notification, indication and it calls the related callbacks.
3. Profile specific state machine: it is called for checking current profile state and performing related actions and consequent status updates (functions *_StateMachine(void) on each specific profile);
4. Application state machine: it performs application-specific actions (sending data to profiles, enabling advertising, displaying to the user, etc.).

The profiles central roles are based on a new set of APIs that allow the execution of the following operations on a BlueNRG, BlueNRG-MS Master/Central device:

- Master configuration functions
- Master security functions
- Master device discovery functions
- Master device connection functions
- Master discovery services & characteristics functions
- Master data exchange functions
- Master common services functions

Following are more details about the central profiles framework key functions, which are similar between the different supported profiles (heart rate collector is taken as reference).

Heart rate collector profile header files:

- heart_rate_collector.h: it defines profile context, states and APIs interfaces;
- heart_rate_collector_config.h: it defines profile parameters (address, security, discovery and connection timings).

Heart rate collector profile, central role key APIs:

- Initialization function: HRC_Init().This function performs the profile initialization using the Master_Init() API with profile specific initialization parameters.
- Security initialization function: HRC_SecuritySet().This function sets the security parameters on the heart rate collector device. It uses the Master_SecuritySet() API with profile specific security parameters.
- Device discovery function: HRC_DeviceDiscovery().This function allows to discover the heart rate sensor (peripheral role) to which connect. It uses the Master_DeviceDiscovery() API with profile specific discovery parameters. The remote device's information discovered during the scan procedure are returned through the HRC_DeviceDiscovery_CB user profile callback.
- Device connection function: HRC_DeviceConnection(). This function starts the connection procedure by using the Master_DeviceConnection() API with profile specific connection parameters in order to connect to the discovered heart rate sensor device. The connection procedure status (connection done event, connection failed event, disconnection event) is returned through the HRC_ConnectionStatus_CB() user profile callback.
- Discover all services function: HRC_ServicesDiscovery(). This function starts the service discovery procedure by using the Master_GetPrimaryServices() API for getting all the service of the connected heart rate sensor device. All the found services information are returned through the HRC_ServicesDiscovery_CB() user profile callback.
- Discover all the characteristic of the heart rate service function: Device_Discovery_CharacServ(HEART_RATE_SERVICE_UUID). This function starts the heart rate sensor characteristic discovery procedure by using the Master_GetCharacOfService() API. It allows to get all the characteristics of the heart rate service of the connected heart rate sensor. The discovered characteristics are returned through the HRC_CharacOfService_CB() user profile callback.

For other APIs description specific to each profile (characteristic descriptor discovery procedure, read, write, notification & indication enabling) refer to the specific profile section.

4 Profiles framework

The section describes the main components of the profiles peripheral and central framework.

4.1 Peripheral framework

The profile peripheral framework consists of the following main components:

1. Profiles libraries for BlueNRG, BlueNRG-MS devices
   - Profile_Library_Release_BlueNRG.a, Profile_Library_Release_BlueNRG-MS.a files (releases version for BlueNRG, BlueNRG-MS devices)
   - Profile_Library_Debug_BlueNRG.a,Profile_Library_Debug_BlueNRG-MS.a files (debug versions with debug messages for BlueNRG, BlueNRG-MS devices)

These binary libraries provide support for each of the following profiles (GAP peripheral role):

- Alert Notification Client
- Alert Notification Server
- Blood Pressure Sensor
- Find Me Locator
- Find Me Target
- Glucose Sensor
- Health Thermometer
- Hearth Rate
- HID
- Phone Alert
- Proximity Monitor
- Proximity Reporter
- Time Client
- Time Server

2. Main profile file
   - profile.c: it provides the common profile framework to all the profiles, and it implements the BlueNRG events callback HCI_Event_CB(void *pckt).
3. Generic profile interface header files:
   a) ble_events.h: it defines the generic and profiles specific events;
   b) ble_profile.h: main profile header file;
   c) ble_status.h: profile status and error codes;
   d) debug.h: function for specific profile debug messages
   e) host_config.h: define values for selecting each specific supported profile (through the BLE_CURRENT_PROFILE_ROLES definition)
   f) uuid.h: profile service & characteristics UUID as defined in the SIG specification (https://developer.bluetooth.org/gatt/profiles/Pages/ProfilesHome.aspx)

Profiles interfaces header files:

Table 1. Profiles interfaces header files

Profile	Header file
Alert notification client	alertNotification_Client.h
Alert notification server	alertNotification_Server.h
Blood pressure sensor	blood_pressure.h
Find Me locator	findme_locator.h
Find Me target	findme_target.h
Glucose sensor	glucose_service.h glucose_sensor.h glucose_racp.h glucose_database.h
Health thermometer	health_thermometer.h
Hearth rate	heart_rate.h
HID device	hid_device.h hid_device_i.h hid_device_i.h
Phone alert	phone_alert_client.h
Proximity monitor	proximity_monitor.h
Proximity reporter	proximity_reporter.h
Time client	time_client.h
Time server	time_server.h time_profile_types.h

4. Profile test application for profile PTS validation: profiles_test_application.c. This file addresses the following features:
   - Set the profile security parameters and initialize the main profile by defining the main profile callback function:
   - Initialize the selected profile (through the BLE_CURRENT_PROFILE_ROLES definition) by calling the profile *_Init() function with the BLE_Profiles_Evt_Notify_Cb() as one of the initialization parameters;
   - It defines the while(1) loop where the HCI_process(), Profile_Process() and profile specific state machine (*_StateMachine()) functions are processed;
   - It allows to enter & process specific user commands (through serial I/O), in order to interact with each profile and performs specific actions (i.e. ask to profile to notify/indicate a characteristic). Such user input commands are used during profile PTS validation tests. The supported user input commands are defined within the profiles_test_application.c file.

Note: An EWARM project defining a workspace for each supported profile and including the profile library is available. By selecting the specific profile workspace, a profile test application supporting the selected profile is built. This application can be used for validating the profile using the PTS USB dongle and related Bluetooth PTS SW tool.

4.2 Central framework

The profiles central framework consists of the following main components:

1. Profiles libraries for BlueNRG, BlueNRG-MS devices

Table 2. Profiles central roles binary libraries (BlueNRG, BlueNRG-MS devices)

Profile	Library debug version (with full set of debug messages)	Library release version (with minimal set of debug messages)
Alert notification client	Alert_Notification_Client_Central_Debug_BlueNRG.a Alert_Notification_Client_Central_Debug_BlueNRG-MS.a	Alert_Notification_Client_Central_Release_BlueNRG.a Alert_Notification_Client_Central_Release_BlueNRG-MS.a
Alert notification server	Alert_Notification_Server_Central_Debug_BlueNRG.a Alert_Notification_Server_Central_Debug_BlueNRG-MS.a	Alert_Notification_Server_Central_Release_BlueNRG.a Alert_Notification_Server_Central_Release_BlueNRG-MS.a
Blood pressure collector	Blood_Pressure_Collector_Central_Debug_BlueNRG.a Blood_Pressure_Collector_Central_Debug_BlueNRG-MS.a	Blood_Pressure_Collector_Central_Release_BlueNRG.a Blood_Pressure_Collector_Central_Release_BlueNRG-MS.a
Find me locator	Find_Me_Locator_Central_Debug_BlueNRG.a Find_Me_Locator_Central_Debug_BlueNRG-MS.a	Find_Me_Locator_Central_Release_BlueNRG.a Find_Me_Locator_Central_Release_BlueNRG-MS.a
Find me target	Find_Me_Target_Central_Debug_BlueNRG.a Find_Me_Target_Central_Debug_BlueNRG-MS.a	Find_Me_Target_Central_Release_BlueNRG.a Find_Me_Target_Central_Release_BlueNRG-MS.a
Glucose collector	Glucose_Collector_Central_Debug_BlueNRG.a Glucose_Collector_Central_Debug_BlueNRG-MS.a	Glucose_Collector_Central_Release_BlueNRG.a Glucose_Collector_Central_Release_BlueNRG-MS.a
Health thermometer collector	Health_Thermometer_Collector_Central_Debug_BlueNRG.a Health_Thermometer_Collector_Central_Debug_BlueNRG-MS.a	Health_Thermometer_Collector_Central_Release_BlueNRG.a Health_Thermometer_Collector_Central_Release_BlueNRG-MS.a
Heart rate collector	Heart_Rate_Collector_Central_Debug_BlueNRG.a Heart_Rate_Collector_Central_Debug_BlueNRG-MS.a	Heart_Rate_Collector_Central_Release_BlueNRG.a Heart_Rate_Collector_Central_Release_BlueNRG-MS.a
HID host	HID_Host_Central_Debug_BlueNRG.a HID_Host_Central_Debug_BlueNRG-MS.a	HID_Host_Central_Release_BlueNRG.a HID_Host_Central_Release_BlueNRG-MS.a
Phone alert status server	Phone_Alert_Status_Server_Central_Debug_BlueNRG.a Phone_Alert_Status_Server_Central_Debug_BlueNRG-MS.a	Phone_Alert_Status_Server_Central_Release_BlueNRG.a Phone_Alert_Status_Server_Central_Release_BlueNRG-MS.a

Table 2. Profiles central roles binary libraries (BlueNRG, BlueNRG-MS devices) (continued)

Profile	Library debug version (with full set of debug messages)	Library release version (with minimal set of debug messages)
Proximity monitor	Proximity_Monitor_Central_Debug_BlueNRG.a Proximity_Monitor_Central_Debug_BlueNRG-MS.a	Proximity_Monitor_Central_Release_BlueNRG.a Proximity_Monitor_Central_Release_BlueNRG-MS.a
Proximity reporter	Proximity_Reporter_Central_Debug._BlueNRG a Proximity_Reporter_Central_Debug_BlueNRG-MS.a	Proximity_Reporter_Central_Release_BlueNRG.a Proximity_Reporter_Central_Release_BlueNRG-MS.a
Time client	Time_Client_Central_Debug_BlueNRG.a Time_Client_Central_Debug_BlueNRG-MS.a	Time_Client_Central_Release_BlueNRG.a Time_Client_Central_Release_BlueNRG-MS.a
Time server	Time_Server_Central_Debug_BlueNRG.a Time_Server_Central_Debug_BlueNRG-MS.a	Time_Server_Central_Release_BlueNRG.a Time_Server_Central_Release_BlueNRG-MS.a

2. Common profiles header files:
   a) debug.h: functions for specific profile debug message;
   b) host_config.h: defines values for identify supported profile types;
   c) uuid.h: profiles services & characteristics UUID as defined in the SIGspecification (https://developer.bluetooth.org/gatt/profiles/Pages/ProfilesHome.aspx);
   d) master_basic_profile.h: header file for the general central profiles APIs (device discovery, connection, service,characteristics discovery, characteristics read, write, notification and indication enabling, …);
   e) master_basic_profile_sm.h: general central profile APIs states and contexts types;
   f) timer.h: header file for profiles timers.
3. Profiles interfaces header files:

Table 3. Profiles central roles: header files

Profile	Header files
Alert notification client	alert_notification_client.h alert_notification_client_config.h
Alert notification server	alert_notification_server.h alert_notification_server_config.h
Blood pressure collector	blood_pressure_collector.h blood_pressure_collector_config.h
Find me locator	find_me_locator.h find_me_locator_config.h
Find me target	find_me_target.h find_me_target_config.h
Glucose collector	glucose_collector.h glucose_collector_config.h glucose_collector_racp.h glucose_collector_racp_CB.h glucose_service.h
Health thermometer collector	health_thermometer_collector.h health_thermometer_collector_config.h health_thermometer_service.h
Heart rate collector	heart_rate_collector.h heart_rate_collector_config.h heart_rate_service.h
HID host	hid_host.h hid_host_config.h
Phone alert status server	phone_alert_status_server.h phone_alert_status_server_config.h
Proximity monitor	proximity_monitor.h proximity_monitor_config.h
Proximity reporter	proximity_reporter.h proximity_reporter_config.h
Time client	time_client.h time_client_config.h time_profile_types.h
Time server	time_server.h time_server_config.h time_profile_types.h

4. Profiles test applications for profiles PTS validation:

Table 4. Profiles central roles: test applications

Profile	PTS test application
Alert notification client	profile_AlertNotificationClient_central_test_application.c
Alert notification server	profiles_AlertNotificationServer_central_test_application
Blood pressure collector	profiles_BloodPressure_central_test_application.c
Find me locator	profiles_FindMeLocator_central_test_application.c
Find me target	profiles_FindMeTarget_central_test_application.c
Glucose collector	profiles_Glucose_central_test_application.c
Health thermometer collector	profiles_HT_central_test_application.c
Heart rate collector	profiles_HR_central_test_application.c
HID host	profiles_HidHost_central_test_application.c
Phone alert status server	profiles_PhoneALertStatusServer_central_test_application.c
Proximity monitor	profiles_ProximityMonitor_central_test_application.c
Proximity reporter	profiles_ProximityReporter_central_test_application.c
Time client	profiles_TimeClient_central_test_application.c
Time server	profiles_TimeServer_central_test_application.c

Each PTS test application file provides a set of commands and callbacks for interacting with the profiles and allowing to address the following features:

- Initialize the profile by calling the profile *_Init() function with the required initialization parameters
- Set the profile security parameters
- Start the discovery procedure for discovering the peripheral device
- Connect to the discovered peripheral device
- Start the service discovery procedures on the connected peripheral device
- Start the characteristic discovery procedure for each supported service
- Found the characteristics descriptors
- Read, write the characteristics with read, write properties
- Enable characteristics indications/notifications for characteristics with indicate/notify property
- Start pairing procedure
- Start the disconnection procedure
- Device discovery callback where the found device is returned
- Device connection callback where the status of the connection is returned
- Device services and characteristics callbacks where the discovered services, characteristics, characteristics descriptors are returned
- Device pairing callback where the pairing status is returned
- Profiles specific callbacks where the characteristic read, notifications and indications are returned.

It also defines the while(1) loop where the HCI_process(), Master_Process() and profile specific state machine (*_StateMachine()) functions are processed. The supported user input commands are defined within the specific profile test application source file and they allow to validate each profile through the related PTS profile tests suite.

Note: 1: An EWARM project defining a workspace for each supported profile and including the related profile library is available. By selecting the specific profile workspace, a profile test application supporting the selected profile is built. This application can be used for validating the specific profile using the PTS USB dongle and related Bluetooth PTS SW tool.

2: Where applicable, a central profile also provides an API (._ConnConf() for running a profile state machine allowing to execute all the keys central role configuration procedures in a single step: connection, service discovery, characteristic discovery and device configuration procedure.*

.

5 Profiles peripheral roles: events

Function calls handle the communication between the application and profiles. Any application using the profiles should first initialize the base profile by calling the BLE_Profile_Init() function, followed by a call to the profile specific initialization function. For example, HRProfile_Init() function is called for the heart rate profile. To enable execution, the application requires a loop that calls the HCI_Process() and Profile_Process() continuously.

The BLE_Profile_Init() function takes two parameters:

- A pointer to the securityParameters; the security parameters should specify the io capabilities, mitm mode, bonding mode and encryption key size.
- A callback function; the callback registered should be of the form: typedef void (* BLE_CALLBACK_FUNCTION_TYPE) (tNotificationEvent event, uint8 evtLen, uint8* evtData).

This function is used by the profile to notify the application of profile events. When the application is notified of an event, it only reads the number of parameters specified in the evtLen parameter.

Below is the list of profiles peripheral events sent by the various profiles to the application.

5.1 Generic events

The events in this category are not specific to any profile.

1. EVT_MP_BLUE_INITIALIZED: this event is sent to the application by the main profile when the controller has been initialized.
2. EVT_MP_CONNECTION_COMPLETE: this event is sent to the application by the main profile when a connection has been successfully established with the peer.
3. EVT_MP_PASSKEY_REQUEST: this event is sent to the application by the main profile when there is a request for passkey during the pairing process from the controller. This event has no parameters. The application must call the function BLE_Profile_Send_Pass_Key() and send the passkey to the controller.
4. EVT_MP_PAIRING_COMPLETE: this event is sent to the application by the main profile when the device is successfully paired with the peer.
5. EVT_MP_DISCONNECTION_COMPLETE: this event is sent to the application by the main profile to notify the result of a disconnection procedure initiated by master/slave.
6. EVT_MP_ADVERTIZE_ERROR: this event is sent by any of the child profiles when enabling of advertising fails. It is the application's responsibility to restart advertising.
7. EVT_MP_ADVERTISING_TIMEOUT: this event is sent by the child profiles when the limited discoverable mode times out or the profile-specific advertising timeout occurs. It is the application's responsibility to restart advertising.

5.2 Alert notification server events

1. EVT_ANS_INITIALIZED: this event is sent by the alert notification server to the application when the initialization sequence has completed and the device is ready to start advertising.

5.3 Alert notification client events

1. EVT_ANC_INITIALIZED: this event is sent by the alert notification client to the application when the initialization sequence has completed and the device is ready to start advertising.
2. EVT_ANC_DISCOVERY_CMPLT: this event is sent by the alert notification client after a connection is established and all the mandatory services, characteristics and descriptors as specified in the profile specification were discovered successfully.
3. EVT_ANC_NEW_ALERT_RECEIVED: this event is sent to the application when a notification for the new alert is received by the alert notification client.
4. EVT_ANC_UNREAD_ALERT_STATUS_RECEIVED: this event is sent to the application when a notification for an unread alert is received by the alert notification client.

5.4 Blood pressure sensor events

1. EVT_BPS_INITIALIZED: this event is sent by the blood pressure sensor to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_BPS_BPM_CHAR_UPDATE_CMPLT: this event is sent to the application when an update to the blood pressure measurement characteristic previously started by the application completes. The status indicates whether the update was successful or not.
3. EVT_BPS_ICP_CHAR_UPDATE_CMPLT: this event is sent to the application when an update to the intermediate cuff pressure characteristic previously started by the application completes. The status indicates whether the update was successful or not.
4. EVT_BPS_IDLE_CONNECTION_TIMEOUT: this event is sent to the application when there is no measurements to be sent to the collector for more than five seconds.

5.5 Find me locator events

1. EVT_FML_INITIALIZED: this event is sent by the find me locator to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_FML_DISCOVERY_CMPLT: this event is sent by the find me locator after a connection is established. The evtData contains the error code:

- 0x00: all the mandatory services, characteristics and descriptors as specified in the profile specification were discovered successfully.
- 0x01: alert characteristic not found.
- 0x02: immediate alert service not found.

5.6 Find me target events

1. EVT_FMT_INITIALIZED: this event is sent to the application when the find me target has completed its initialization sequence and is ready to enable advertising, or the initialization sequence failed. The evtData parameter contains the error code; 0X00 means the initialization was successful.
2. EVT_FMT_ALERT: this event is sent to the application when the client writes to the alert level characteristic with a valid alert level. The application must start alerting if the alert level is 0x01 or 0x02, and stop when the alert level is 0x00.

5.7 Glucose sensor events

1. EVT_GL_INITIALIZED: this event is sent by the glucose sensor to the application when the initialization sequence has completed and the device is ready to start advertising.
2. EVT_GL_IDLE_CONNECTION_TIMEOUT: this event is sent to the application when the connection is idle for more than five seconds.

5.8 Health thermometer events

1. EVT_HT_INITIALIZED: this event is sent by the thermometer to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_HT_TEMPERATURE_CHAR_UPDATE_CMPLT: this event is sent to the application when an update to the temperature measurement characteristic previously started by the application completes. The status indicates whether the update was successful or it failed.
3. EVT_HT_INTERMEDIATE_TEMP_CHAR_UPDATE_CMPLT: this event is sent to the application when an update to the intermediate temperature measurement characteristic previously started by the application completes. The status indicates whether the update was successful or it failed.
4. EVT_HT_MEASUREMENT_INTERVAL_RECEIVED: this event is sent to the application when the collector writes to the measurement interval characteristic.
5. EVT_HT_MEASUREMENT_INTERVAL_UPDATE_CMPLT: this event is sent to the application when an update to the measurement interval characteristic previously started by the application completes. The status indicates whether the update was successful or it failed.
6. EVT_HT_IDLE_CONNECTION_TIMEOUT: this event is sent to the application when there are no measurements to be sent to the collector for more than five seconds.

5.9 Heart rate profile events

The events under this category are those which are sent by the heart rate profile to the application.

1. EVT_HRS_INITIALIZED: this event is sent to the application when the heart rate profile has completed its initialization sequence and is ready to enable advertising or the

initialization sequence failed. The evtData parameter contains the error code; 0X00 means the initialization was successful.

2. EVT_HRS_CHAR_UPDATE_CMPLT: this event is sent to the application whenever it has started a characteristic update procedure to update the heart rate measurement or body sensor location. The evtData contains the status, service handle, and characteristic handle. This has to be changed to give different events for each update since the application is not aware of the handles.
3. EVT_HRS_RESET_ENERGY_EXPENDED: this event is sent to the application when the peer writes a value of 0x01 to the control point characteristic. This event has no parameters. The application must restart accumulating the energy expended values from 0.

5.10 HID events

1. EVT_HID_INITIALIZED: this event is sent to the application when the HID has completed its initialization sequence and is ready to enable advertising, or the initialization sequence failed. The evtData parameter contains the error code; 0X00 means the initialization was successful.
2. EVT_HID_UPDATE_CMPLT: this event is sent to the application when an update previously started by the application completes. The status indicates whether the update was successful or it failed. The evtData also contains the service handle and the characteristic handle.
3. EVT_BATT_LEVEL_READ_REQ: this event is sent to the application when the client requests a battery level reading. On receiving this event, the application can update the battery level characteristic and then call the function Allow_BatteryLevel_Char_Read. If the process takes more than 30 minutes, the GATT channel is closed.

5.11 Phone alert client events

1. EVT_PAC_INITIALIZED: this event is sent by the phone alert client to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_PAC_DISCOVERY_CMPLT: this event is sent by the phone alert client after a connection is established. The evtData contains the error code:
   0x00: all the mandatory services, characteristics and descriptors as specified in the profile specification were discovered successfully.
   0x01: PHONE_ALERT_SERVICE_NOT_FOUND.
   0x02: PHONE_ALERT_STATUS_CHARAC_NOT_FOUND.
   0x03: RINGER_CNTRL_POINT_CHARAC_NOT_FOUND.
   0x04: RINGER_SETTING_CHARAC_NOT_FOUND.
   0x05: PHONE_ALERT_STATUS_DESC_NOT_FOUND.
   0x06: RINGER_CNTRL_POINT_DESC_NOT_FOUND.
   0x07: RINGER_SETTING_DESC_NOT_FOUND.
3. EVT_PAC_ALERT_STATUS: the application can start a procedure to read the alert status characteristic on the peer server using the function PAC_Read_AlertStatus().

The response to this function call is returned in this event. The evtData contains the response received from the server.

4. EVT_PAC_RINGER_SETTING: the application can read the ringer setting on the server by using the function PAC_Read_RingerSetting(). The response to this function call is returned in this event. The evtData contains the response received from the server.

5.12 Proximity monitor events

1. EVT_PM_INITIALIZED: this event is sent by the proximity monitor to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_PM_DISCOVERY_CMPLT: this event is sent by the proximity monitor after a connection is established. The evtData contains the error code.
   – 0x00: all the mandatory services, characteristics and descriptors as specified in the profile specification were discovered successfully.
   – 0x01: link loss service not found.
3. EVT_PM_LINK_LOSS_ALERT: this event is sent to the application when a link loss is detected. The evtData contains the alert level. The application must start an alert for the level specified. The type of alert is decided by the application: the alert can continue for an application-specific duration or until another connection is established. The application must re-enable advertising to establish a new connection.
4. EVT_PM_PATH_LOSS_ALERT: this event is sent to the application by the proximity monitor when a path loss is detected. The evtData contains the alert level. When a path loss is detected, the application can start an alert of any type for the alert level specified.

5.13 Proximity reporter events

The events under this category are those which are sent by the proximity profile in the reporter role to the application.

1. EVT_PR_INITIALIZED: this event is sent to the application when the proximity reporter has completed its initialization sequence and is ready to enable advertising, or the initialization sequence failed. The evtData parameter contains the error code; 0X00 means the initialization was successful.
2. EVT_PR_LINK_LOSS_ALERT: this event is sent to the application when a link loss is detected. The evtData contains the alert level. The application must start an alert for the level specified. The type of alert is decided by the application: the alert can continue for an application-specific duration or until another connection is established. The application must re-enable advertising to establish a new connection.
3. EVT_PR_PATH_LOSS_ALERT: this event is sent to the application by the proximity reporter when a path loss is detected. The evtData contains the alert level. When a path loss is detected, an alert of any type must be started – the desired user action would be to move the device closer to its connected peer. The alert should continue until another event with alert level 0 is issued.

5.14 Time client events

1. EVT_TC_INITIALIZED: this event is sent by the time client to the application when the initialization sequence is completed and the device is ready to start advertising.
2. EVT_TC_DISCOVERY_CMPLT: this event is sent by the time client after a connection is established and all the mandatory services, characteristics and descriptors as specified in the profile specification were discovered successfully.
3. EVT_TC_CUR_TIME_VAL_RECEIVED: this event is sent to the application when a notification for the current time characteristic is received by the time client. The event data contains all the fields of the current time characteristic.
4. EVT_TC_READ_CUR_TIME_CHAR: The application can read the current time characteristic on the server by using the function TimeClient_Get_Current_Time(). The response to this function call is returned in this event. The evtData contains the response received from the server.
5. EVT_TC_READ_REF_TIME_INFO_CHAR: The application can read the reference time characteristic on the server by using the function TimeClient_Get_Time_Accuracy_Info_Of_Server(). The response to this function call is returned in this event. The evtData contains the response received from the server.
6. EVT_TC_READ_LOCAL_TIME_INFO_CHAR: The application can read the local time information characteristic on the server by using the function TimeClient_Get_Local_Time_Information(). The response to this function call is returned in this event. The evtData contains the response received from the server.
7. EVT_TC_READ_TIME_WITH_DST_CHAR: The application can read the time with dst change characteristic on the server by using the function TimeClient_Get_Next_DST_Change_Time(). The response to this function call is returned in this event. The evtData contains the response received from the server.
8. EVT_TC_READ_TIME_UPDATE_STATE_CHAR: The application can read the time update state characteristic on the server by using the function TimeClient_Get_Server_Time_Update_State(). The response to this function call is returned in this event. The evtData contains the response received from the server.

5.15 Time server events

1. EVT_TS_INITIALIZED: this event is sent by the time server to the application when the initialization sequence has completed and the device is ready to start advertising.
2. EVT_TS_CHAR_UPDATE_CMPLT: this event is sent to the application when an update previously started by the application completes. The status indicates whether the update was successful or it failed. The evtData also contains the service handle and the characteristic handle.
3. EVT_TS_START_REFTIME_UPDATE: this event is sent to the application when the GET_REFERENCE_UPDATE(0x01) command is written to the updateState characteristic by the time client.
4. EVT_TS_STOP_REFTIME_UPDATE: this event is sent to the application when the CANCEL_REFERENCE_UPDATE(0x02) command is written to the updateState characteristic by the time client.

6 Profiles peripheral roles: APIs interface

This section describes the profiles peripheral roles APIs interface.

6.1 Alert notification Client

6.1.1 ANC_Client_Init()

Description

Initializes the Alert Notification Profile. It returns BLE_STATUS_SUCCESS if the procedure is started successfully. Notification of successful initialization of the profile is sent to the application through the event EVT_ANC_INITIALIZED.

Parameters

- BLE_CALLBACK_FUNCTION_TYPE: callback function called by the profile to notify the application of the events.

6.1.2 ANC_Advertize()

Description

The function puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

- useWhitelist: if the useWhiteList is set to TRUE, the device is configured to use the whitelist which is configured with bonded devices at the time of initialization; otherwise, the device enters limited discoverable mode to connect to any of the available devices.

6.1.3 ANC_Write_Control_Point()

Description

The application calls this to write or update the control point characteristic. The application is notified through the event on successful update.

Parameters

1. Command – ID of the command to be sent. Below is the list of the different command IDs:
   - ENABLE_NEW_ALERT_NOTIFICATION (0x00)
   - ENABLE_UNREAD_ALERT_STATUS_NOTIFICATION (0x01)
   - DISABLE_NEW_ALERT_NOTIFICATION (0x02)
   - DISABLE_UNREAD_ALERT_STATUS_NOTIFICATION (0x03)
   - NOTIFY_NEW_ALERT_IMMEDIATELY (0x04)
   - NOTIFY_UNREAD_ALERT_STATUS_IMMEDIATELY (0x05)

2. Category ID – category which has to be affected by the command. Below is the list of the category IDs.
   - CATEGORY_ID_SIMPLE_ALERT (0x00)
   - CATEGORY_ID_EMAIL (0x01)
   - CATEGORY_ID_NEWS (0x02)
   - CATEGORY_ID_CALL (0x03)
   - CATEGORY_ID_MISSED_CALL (0x04)
   - CATEGORY_ID_SMS_MMS (0x05)
   - CATEGORY_ID_VOICE_MAIL (0x06)
   - CATEGORY_ID_SCHEDULE (0x07)
   - CATEGORY_ID_HIGH_PRIORITIZED_ALERT (0x08)
   - CATEGORY_ID_INSTANT_MESSAGE (0x09)

6.1.4 ANC_Enable_Disable_New_Alert_Notification()

Description

Enables the notifications for the new alert characteristic. After enabling this, the control point characteristic must also be written with the command and category to receive alerts from the peer.

Parameters

1. enable – if set to TRUE, it enables the notifications for the new alert characteristic.

6.1.5 ANC_Enable_Disable_Unread_Alert_Status_Notification()

Description

Enables the notifications for the unread alert status characteristic. After enabling this, the control point characteristic must also be written with the command and category to receive alerts from the peer.

Parameters

1. enable: if set to TRUE, it enables the notifications for the unread alert status characteristic.

6.1.6 ANCProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.2 Alert notification server

6.2.1 ANS_Init()

Description

Initializes the Alert Notification Profile in the server role. It returns BLE_STATUS_SUCCESS if the procedure is started successfully. Notification of successful initialization of the profile is sent to the application through the event EVT_ANS_INITIALIZED.

Parameters

- BLE_CALLBACK_FUNCTION_TYPE: callback function called by the profile to notify the application of the events.
- AlertCategory: bitmask of the categories supported for the new alert characteristic.
  - CATEGORY_ID_SIMPLE_ALERT (0x00)
  - CATEGORY_ID_EMAIL (0x01)
  - CATEGORY_ID_NEWS (0x02)
  - CATEGORY_ID_CALL (0x03)
  - CATEGORY_ID_MISSED_CALL (0x04)
  - CATEGORY_ID_SMS_MMS (0x05)
  - CATEGORY_ID_VOICE_MAIL (0x06)
  - CATEGORY_ID_SCHEDULE (0x07)
  - CATEGORY_ID_HIGH_PRIORITIZED_ALERT (0x08)
  - CATEGORY_ID_INSTANT_MESSAGE (0x09)
- unreadAlertCategory: bitmask of the categories supported for the unread alert status characteristic.
  - CATEGORY_ID_SIMPLE_ALERT (0x00)
  - CATEGORY_ID_EMAIL (0x01)
  - CATEGORY_ID_NEWS (0x02)
  - CATEGORY_ID_CALL (0x03)
  - CATEGORY_ID_MISSED_CALL (0x04)
  - CATEGORY_ID_SMS_MMS (0x05)
  - CATEGORY_ID_VOICE_MAIL (0x06)
  - CATEGORY_ID_SCHEDULE (0x07)
  - CATEGORY_ID_HIGH_PRIORITIZED_ALERT (0x08)
  - CATEGORY_ID_INSTANT_MESSAGE (0x09)

6.2.2 ANS_Advertize()

Description

The function puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

useWhitelist: if the useWhiteList is set to TRUE, the device is configured to use the whitelist which is configured with bonded devices at the time of initialization; otherwise, the device enters limited discoverable mode to connect to any of the available devices.

6.2.3 ANS_Update_New_Alert_Category()

Description

The application calls this to update the alert category characteristic with the new bitmask. It returns BLE_STATUS_SUCCESS if the update is successfully started and BLE_STATUS_INVALID_PARAMS if a bitmask for an invalid category is requested.

Parameters

Length: length of the category field; it must be 0 or 1.

Category: bitmask of the categories supported. The bitmasks are split across two octets with the meanings described in the bluetooth assigned numbers documentation.

6.2.4 ANS_Update_Unread_Alert_Category()

Description

The application calls this to update the unread alert category with the new bitmask. It returns BLE_STATUS_SUCCESS if the update is successfully started and BLE_STATUS_INVALID_PARAMS if a bitmask for an invalid category is set.

Parameters

1. Length: length of the category field; it must be 0 or 1.
2. Category: bitmask of the categories supported. The bitmasks are split across two octets with the meanings described in the bluetooth assigned numbers documentation.

6.2.5 ANS_Update_New_Alert()

Description

The application calls this to update the number of new alerts for the category specified in the new alert characteristic. If the category ID specified is not valid or the text information is longer than 18 octets, BLE_STATUS_INVALID_PARAMS is returned. On successful write, BLE_STATUS_SUCCESS is returned.

Parameters

1. alertCount: alert count for the category specified. The application must maintain the count of new alerts.
2. categoryID: category which is affected by the command.
3. TextInfo: textual information corresponding to the alert.

6.2.6 ANS_Update_Unread_Alert_Status()

Description

The application calls this to update the number of unread alerts for the category specified in the new alert characteristic. If the category ID specified is not valid, BLE_STATUS_INVALID_PARAMS is returned. On successful write to the alert status, BLE_STATUS_SUCCESS is returned.

Parameters

1. categoryID: category which is affected by the command.
2. AlertCount: alert count for the category specified. The application must maintain the count of unread alerts.

6.2.7 ANSProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.3 Blood pressure sensor

6.3.1 BPS_Init()

Description

Initializes and registers the blood pressure Sensor profile with the main Profile. It returns BLE_STATUS_SUCCESS if the procedure is started successfully, or BLE_STATUS_FAILED if not. The application is notified through the event EVT_BPS_INITIALIZED on completion of the initialization procedure.

Parameters

1. intermediateCuffPressureChar: indicates whether the blood pressure service should support the intermediate cuff pressure characteristic.
2. Feature: a bitmask representing the features supported by the device. Below is the list of the features supported by the device:
   - BODY_MOVEMENT_DETECTION_FLAG (0x01)
   - CUFF_FIT_DETECTION_FLAG(0x02)
   - IRREGULAR_PULSE_DETECTION_FLAG (0x04)
   - PULSE_RATE_RANGE_EXCEEDS_UPPER_LIMIT (0x08)
   - PULSE_RATE_RANGE_BELOW_LOWER_LIMIT (0x10)
   - MEASUREMENT_POSITION_DETECTION_FLAG (0x20)
3. BLE_CALLBACK_FUNCTION_TYPE: callback function type to be called by the profile to notify the application of the profile specific events.

6.3.2 BPS_Advertize()

Description

The function puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

useWhitelist: if useWhiteList is set to TRUE, the device is configured to use the whitelist which is configured with bonded devices at the time of initialization; otherwise, the device enters limited discoverable mode to connect to any of the available devices.

6.3.3 BPS_Send_Intermediate_Cuff_Pressure()

Description

This function is called to send the intermediate cuff pressure values during the measurement process until a stable value is obtained. The function can only be used only if the intermediate Cuff Pressure Char is set to 'True' during initialization. The application is notified of successful update through the event EVT_BPS_ICP_CHAR_UPDATE_CMPLT.

Parameters

- icpVal: the intermediate Cuff pressure value structure containing the following structure members:
- Flags: the flags is a bitmask which tells the peer of the data to follow.
  - Bit0 – a value of 1 indicates that the unit is kPa; a value of 0 indicates that the unit is mm Hg.
  - Bit3 – a value of 1 implies there is a user ID field in the data
  - Bit4 – a value of 1 implies there is a measurement status in the data.
- Icp: intermediate cuff pressure value.
- UserID:
- if Bit3 is set, then this field should contain the value of the USER ID.
- MeasurementStatus: a structure containing the values of the various bit mask features supported by the device.

6.3.4 BPS_Send_Blood_Pressure_Measurement()

Description

This function is called to send the Blood Pressure measurement values. It updates the blood pressure measurement characteristic with the value if the device is connected. The application is notified through the event EVT_BPS_BPM_CHAR_UPDATE_CMPLT.

Parameters

bpmval: the blood pressure value structure. The members are similar to those in icpValue.

6.3.5 BPS_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.4 Device information service profile

Different profiles have different requirements of the characteristics for the device information service. The profiles specify the characteristics required during the initialization, but the update to these characteristics must be performed by the application.

6.4.1 BLE_Profile_Update_DIS_SystemID()

Description

The application calls this to update the System ID characteristic of the device information service.

Parameters

1. length – the Length of the characteristic to be updated
2. SystemID – the Characteristic value

6.4.2 BLE_Profile_Update_DIS_ModelNum()

Description

The application calls this to update the Model Number characteristic of device information service.

Parameters

1. length – The Length of the characteristic to be updated.
2. modelNum – The Characteristic value

6.4.3 BLE_Profile_Update_DIS_SerialNum()

Description

The application calls this to update the Serial Number characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. serialNum – the Characteristic value

6.4.4 BLE_Profile_Update_DIS_FirmwareRev()

Description

The application calls this to update the Firmware Revision characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. firmwareRev – the Characteristic value to be written or updated.

6.4.5 BLE_Profile_Update_DIS_HardwareRev()

Description

The application calls this to update the Hardware Revision characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. hardwareRev – the Characteristic value.

6.4.6 BLE_Profile_Update_DIS_SoftwareRev()

Description

The application calls this to update the Software Revision characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. softwareRev – the Characteristic value

6.4.7 BLE_Profile_Update_DIS_manufacturerName()

Description

The application calls this to update the Manufacture Name characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. name – the Characteristic value to be written or updated.

6.4.8 BLE_Profile_Update_DIS_IEEECertification()

Description

The application calls this to update the IEEE Certification characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. ieeeCert – the Characteristic value

6.4.9 BLE_Profile_Update_DIS_pnpId()

Description

The application calls this to update the pnpID characteristic of device information service.

Parameters

1. length – the Length of the characteristic to be updated.
2. pnpId – the Characteristic value to be written or updated.

6.5 Find me locator

6.5.1 FindMeLocator_Init()

Description

The application should call this function to initialize the Find Me Locator Profile. The initialization procedure returns BLE_STATUS_SUCCESS if started successfully, or BLE_STATUS_FAILED if not. The application is notified of successful initialization of the profile by the event EVT_FML_INITIALIZED through the registered callback.

Parameters

- bleSecReq: pointer to the structure denoting the security requirements of the profile.
- BLE_CALLBACK_FUNCTION_TYPE: the Callback function to be called to notify the application of the events.

6.5.2 FML_Advertize()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

6.5.3 FML_Add_Device_To_WhiteList()

Description

This function or command is called by the application to add devices to the whitelist.

Parameters

- addrType: address type of the bdAddr to be added to the whitelist.
  - 0x00: PUBLIC ADDRESS
  - 0x01:
  - RANDOM ADDRESS
- bdAddr: address of the peer device that must be added to the whitelist.

6.5.4 FML_Alert_Target()

Description

The function is called by the application to start a write to the alert level on the find me target. It returns BLE_STATUS_SUCCESS if the procedure is started successfully, otherwise, it returns an error.

Parameters

- alertLevel: the alert level for the target must be configured according to the following alert levels:
  - NO_ALERT (0x00)
  - MILD_ALERT (0x01)
  - HIGH_ALERT (0x02)

6.5.5 FMLProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.6 Find me target

6.6.1 FindMeTarget_Init()

Description

The application calls this function to initialize the find me target profile. The initialization procedure returns BLE_STATUS_SUCCESS if started successfully or BLE_STATUS_FAILED if not. The application is notified of successful initialization of the profile by the event EVT_FMT_INITIALIZED through the registered callback.

Parameters

- BLE_CALLBACK_FUNCTION_TYPE: the callback through which the application is notified of events by the find me.

6.6.2 FMT_Advertize()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

6.6.3 FMT_Add_Device_To_WhiteList()

Description

This function is called by the application to add devices into the whitelist.

Parameters

- addrType: address type of the bdAddr to be added to the whitelist.
  - 0x00: PUBLIC ADDRESS
  - 0x01:
  - RANDOM ADDRESS
- bdAddr: address of the peer device that must be added to the whitelist.

6.6.4 FMTProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.7 Glucose sensor

6.7.1 GL_Init ()

Description

Initializes the Glucose Sensor Profile. It returns BLE_STATUS_SUCCESS if the procedure is started successfully and then notifies the application of successful initialization of the profile through the event EVT_GL_INITIALIZED.

Parameters

- sequenceNumber: initial sequence number value (number of stored records on glucose measurement database).
- gl_measurement_db_records : pointer to user glucose measurement database.
- gl_measurement_context_db_records : pointer to user glucose measurement context database
- BLE_CALLBACK_FUNCTION_TYPE: callback function to be called by the profile to notify the application of the events.

6.7.2 GL_Advertize()

Description

This command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

- useWhitelist: If the useWhiteList is set to TRUE, the device is configured to use the whitelist which is configured with bonded devices at the time of initialization, otherwise the device enters limited discoverable mode to connect to any of the available devices.

6.7.3 GL_ResetFlags ()

This function allows resetting of the initialization flags for the glucose sensor.

Parameters

- sequenceNumber: last stored sequence number on database.

6.7.4 GL_Send_Glucose_Measurement()

Description

This function is used to update the glucose measurement characteristic value. The function is only called as consequence of the reception of a record access control point command requesting the notification of one or more glucose measurements based on the glucose sensor database stored measurements.

Parameters

- glucoseMeasurementVal: the glucose measurement value structure containing the following members:

1. record_status_flag: flag to identify the glucose database record status
    1. flags field: these flags define which data fields are present in the Characteristic value
        a) GLUCOSE_MEASUREMENT_FLAGS_TIME_OFFSET_IS_PRESENT (0x01)
        b) GLUCOSE_MEASUREMENT_FLAGS_CONCENTRATION_IS_PRESENT (0x02)
        c) GLUCOSE_MEAUREMENTS_FLAG_MMOL_L_UNITS (0x04)
        d) GLUCOSE_MEASUREMENT_FLAGS_STATUS_ANNUNCIATION_IS_PRESENT (0x08)
    3. sequenceNumber field: sequence number of the glucose measurement value
    4. tBasetime baseTime: time of the measurement
    5. timeOffset field: time component used to define the overall user-facing time
    6. glucoseConcentration: glucose concentration field (SFLOAT units of Kg or Liters)
    7. typeSampleLocation Field: measurement type and sample location information type nibble:
        a) GLUCOSE_TYPE_CAPILLARY_WHOLE_BLOOD (0x1)
        b) GLUCOSE_TYPE_CAPILLARY_PLASMA (0x2)
        c) GLUCOSE_TYPE_VENOUS_WHOLE_BLOOD (0x3)
        d) GLUCOSE_TYPE_VENOUS_PLASMA (0x4)
        e) GLUCOSE_TYPE_ARTERIAL_WHOLE_BLOOD (0x5)
        f) GLUCOSE_TYPE_ARTERIAL_PLASMA (0x6)
        g) GLUCOSE_TYPE_UNDERTERMINED_WHOLE_BLOOD (0x7)
        h) GLUCOSE_TYPE_UNDERTERMINED_PLASMA (0x8)
        i) GLUCOSE_TYPE_INTERSISTIAL_FLUID (0x9)
        j) GLUCOSE_TYPE_CONTROL (0xA) sampleLocation nibble:
        a) GLUCOSE_SAMPLE_LOCATION_FINGER (0x10)
        b) GLUCOSE_SAMPLE_LOCATION_AST (0x20)
        c) GLUCOSE_SAMPLE_LOCATION_EARLOBE (0x30)
        d) GLUCOSE_SAMPLE_LOCATION_CONTROL_SOLUTION (0x40)
        e) GLUCOSE_SAMPLE_LOCATION_VALUE_NOT_AVAILABLE (0xF0)
    8. sensorStatusAnnunciation field:
        a) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_DEVICE_BATTERY_LOW (0x0001)
        b) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_MALFUNCTION (0x0002)

c) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SAMPLE_SIZE (0x0004)

d) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_STRIP_INSERTION_ERROR (0x0008)

e) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_STRIP_TYPE_INCORRECT (0x0010)

f) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_RESULT_TOO_HIGH (0x0020)

g) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_RESULT_TOO_LOW(0x0040)

h) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_TEMPERATURE_TO O_HIGH (0x0080)

i) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_TEMPERATURE_TO O_LOW (0x0100)

j) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_SENSOR_READ_INTERRUPTE D (0x0200)

k) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_GENERAL_DEVICE_FAULT (0x0400)

l) GLUCOSE_SENSOR_STATUS_ANNUNCIATION_TIME_FAULT (0x0800)

6.7.5 GL_Send_Glucose_Measurement_Context()

Description

This function is used to update the glucose measurement context characteristic value. The function is only called as consequence of the reception of a record access control point command requesting the notification of one or more glucose measurement based on the glucose sensor database stored measurements if the associated glucose measurement characteristic includes contextual information.

Parameters

- glucoseMeasurementContextVal: the glucose measurement context value structure containing the following members

1. flags field: these flags define which data fields are present in the Characteristic value
   a) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_CARBOHYDRATE_IS_PRESEN T (0x01)
   b) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_MEAL_IS_PRESENT (0x02)
   c) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_TESTER_HEALTH_IS_PRESEN T (0x04)
   d) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_EXERCISE_DURATION_IS_PR ESENT (0x08)
   e) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_MEDICATION_ID_IS_PRESENT (0x10)
   f) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_MEDICATION_LITER_UNITS (0x20)
   g) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_HB1A1C_IS_PRESENT (0x40)
   h) GLUCOSE_MEASUREMENT_CONTEXT_FLAG_EXTENDED_IS_PRESENT (0x80)
2. sequenceNumber field: sequence number of the glucose measurement context value (same as the associated glucose measurement value)
3. extendedFlags field: optional field
4. carbohydrateId field:
   a) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_BREAKFAST (0x01)
   b) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_LUNCH (0x02)
   c) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_DINNER (0x03)
   d) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_SNACK (0x04)
   e) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_DRINK (0x05)
   f) GLUCOSE_MEASUREMENT_CONTEXT_CARBOHYDRATE_SUPPER (0x06)
   g) GLUCOSE_MEASUREMENT_CONTEXT_CARBHYDRATEO_BRUNCH (0x07)
5. carbohydrateUnits field: units of carbohydrate
6. meal field:
   a) GLUCOSE_MEASUREMENT_CONTEXT_MEAL_PREPRANDIAL (0x1)
   b) GLUCOSE_MEASUREMENT_CONTEXT_MEAL_POSTPRANDIAL (0x2)
   c) GLUCOSE_MEASUREMENT_CONTEXT_MEAL_FASTING(0x3)
   d) GLUCOSE_MEASUREMENT_CONTEXT_MEAL_CASUAL(0x4)
   e) GLUCOSE_MEASUREMENT_CONTEXT_MEAL_BEDTIME(0x5)
7. testerHealth field tester nibble:
   a) GLUCOSE_MEASUREMENT_CONTEXT_TESTER_SELF (0x1)
   b) GLUCOSE_MEASUREMENT_CONTEXT_TESTER_HEALTH_CARE_PROFESSI ONAL (0x2)
   c) GLUCOSE_MEASUREMENT_CONTEXT_TESTER_LAB_TEST (0x3)
   d) GLUCOSE_MEASUREMENT_CONTEXT_TESTER_NOT_AVAILABLE (0xF) health nibble:
   e) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_MINOR_ISSUES (0x10)
   f) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_MAJOR_ISSUES (0x20)
   g) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_DURING_MENSES (0x30)
   h) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_UNDER_STRESS (0x40)

i) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_NO_ISSUE (0x50)
j) GLUCOSE_MEASUREMENT_CONTEXT_HEALTH_VALUE_NOT_AVAILABLE (0xF0)

8. exerciseDuration field: exercise duration in seconds
9. exerciseIntensity field: intensity of exercise
10. medicationId field:
    a) GLUCOSE_MEASUREMENT_CONTEXT_MEDICATION_ID_RAPID_ACTING_INS ULIN (0x1)
    b) GLUCOSE_MEASUREMENT_CONTEXT_MEDICATION_SHORT_ACTING_INSU LIN (0x2)
    c) GLUCOSE_MEASUREMENT_CONTEXT_MEDICATION_INTERMEDIATE_ACTIN G_INSULIN (0x3)
    d) GLUCOSE_MEASUREMENT_CONTEXT_MEDICATION_LONG_ACTING_INSULI N (0x4)
    e) GLUCOSE_MEASUREMENT_CONTEXT_MEDICATION_PRE_MIXED_INSULIN (0x5)
11. medicationUnits field: units of kilograms or liters;
12. HbA1c field

6.7.6 GL_Set_Glucose_Feature_Value ()

Description

This function is used to set the glucose feature characteristic value.

Parameters

- value: glucose feature value
  a) GLUCOSE_FEATURE_LOW_BATTERY_DETECTION_IS_SUPPORTED (0x0001)
  b) GLUCOSE_FEATURE_SENSOR_MALFUNCTION_DETECTION_IS_SUPPORTE D (0x0002)
  c) GLUCOSE_FEATURE_SAMPLE_SIZE_IS_SUPPORTED (0x0004)
  d) GLUCOSE_FEATURE_SENSOR_STRIP_INSERTION_ERROR_IS_SUPPORTED (0x0008)
  e) GLUCOSE_FEATURE_SENSOR_STRIP_TYPE_ERROR_IS_SUPPORTED (0x0010)
  f) GLUCOSE_FEATURE_SENSOR_RESULT_HIGH_LOW_DETECTION_IS_SUPPORTED (0x0020)
  g) GLUCOSE_FEATURE_SENSOR_TEMPERATURE_HIGH_LOW_DETECTION_IS
  h) _SUPPORTED (0x0040)
  i) GLUCOSE_FEATURE_SENSOR_READ_INTERRUPT_DETECTION_IS_SUPPORTED (0x0080)
  j) GLUCOSE_FEATURE_GENERAL_DEVICE_FAULT_IS_SUPPORTED (0x0100)
  k) GLUCOSE_FEATURE_TIME_FAULT_IS_SUPPORTED (0x0200)
  l) GLUCOSE_FEATURE_MULTIPLE_BOND_IS_SUPPORTED (0x0400)

6.7.7 GL_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.8 Health thermometer

6.8.1 HT_Init()

Description

Initializes the health thermometer profile. Returns BLE_STATUS_SUCCESS if the procedure is started successfully, and then notifies the application of successful initialization of the profile through the event EVT_HT_INITIALIZED.

Parameters

- thermometerFeatures: bitmask for the characteristics to be added to the health thermometer service. The various bit masks for the characteristics are:
  - INTERMEDIATE_TEMPERATURE_CHAR (0x01)
  - MEASUREMENT_INTERVAL_CHAR (0x02)
  - TEMPERATURE_TYPE (0x04)
- minValidInterval: the minimum valid interval value for the measurement interval characteristic. This is only valid if the MEASUREMENT_INTERVAL_CHAR flag is set in the thermometer features.
- maxValidInterval: the maximum valid interval value for the measurement interval characteristic.
- BLE_CALLBACK_FUNCTION_TYPE: callback function to be called by the profile to notify the application of the events.

6.8.2 HT_Advertize()

Description

This command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

- useWhitelist: If the useWhiteList is set to TRUE, the device is configured to use the whitelist which is configured with bonded devices at the time of initialization; otherwise, the device enters limited discoverable mode to connect to any of the available devices.

6.8.3 HT_Send_Temperature_Measurement()

Description

The application calls this function to update the temperature measurement characteristic. The application is notified through the event EVT_HT_TEMPERATURE_CHAR_UPDATE_CMPLT when the update is complete.

Parameters

- tempMeasurementVal: The temperature measurement value structure contains the following members:

1. flags – bit mask of the fields supported in the characteristic.
   a) FLAG_TEMPERATURE_UNITS_FARENHEIT (0x01)
   b) FLAG_TIMESTAMP_PRESENT (0x02)
   c) FLAG_TEMPERATURE_TYPE (0x04)
2. Temperature – temperature measurement value(4 byte)
3. timeStamp – timestamp of the measurement
4. temperatureType – temperature type
   a) TEMP_MEASURED_AT_ARMPIT (0x01)
   b) TEMP_MEASURED_FOR_BODY (0x02)
   c) TEMP_MEASURED_AT_EAR (0x03)
   d) TEMP_MEASURED_AT_FINGER (0x04)
   e) TEMP_MEASURED_AT_GASTRO_INTESTINAL_TRACT (0x05)
   f) TEMP_MEASURED_AT_MOUTH (0x06)
   g) TEMP_MEASURED_AT_RECTUM (0x07)
   h) TEMP_MEASURED_AT_TOE (0x08)
   i) TEMP_MEASURED_AT_TYMPANUM (0x09)

6.8.4 HT_Send_Intermediate_Temperature()

Description

The application calls this function to update the intermediate temperature measurement characteristic. The application is notified through the event EVT_HT_INTERMEDIATE_TEMP_CHAR_UPDATE_CMPLT when the update completes.

Parameters

flags – bitmask of the fields supported in the characteristic. Refer to *Section 6.8.3: HT_Send_Temperature_Measurement()* for valid values.

Temperature – temperature measurement value.

timestamp – timestamp of the measurement.

temperatureType – temperature type. Refer to *Section 6.8.3: HT_Send_Temperature_Measurement()* for valid values.

6.8.5 HT_Update_Measurement_Interval()

Description

The application calls this to update the measurement interval value characteristic. This is the interval between the temperature updates sent to the collector. On completion of the update, the application is notified through the event EVT_HT_MEASUREMENT_INTERVAL_UPDATE_CMPLT.

Parameters

Interval – the gap interval after which the update of the measurement value is to be performed.

6.8.6 HT_Update_Temperature_Type()

Description

The application calls this to update the temperature type characteristic. The temperature type indicates the part of the body where the temperature is being measured. During an active connection, this setting must remain static and updates are not allowed. On successful update, the event EVT_HT_TEMP_TYPE_CHAR_UPDATE_CMPLT is sent to the application.

Parameters

Type – the type denotes the part of the body where the temperature is measured; below is the list of the type fields:

- TEMP_MEASURED_AT_ARMPIT (0x01)
- TEMP_MEASURED_FOR_BODY (0x02)
- TEMP_MEASURED_AT_EAR (0x03)
- TEMP_MEASURED_AT_FINGER (0x04)
- TEMP_MEASURED_AT_GASTRO_INTESTINAL_TRACT (0x05)
- TEMP_MEASURED_AT_MOUTH (0x06)
- TEMP_MEASURED_AT_RECTUM (0x07)
- TEMP_MEASURED_AT_TOE (0x08)
- TEMP_MEASURED_AT_TYMPANUM (0x09)

6.8.7 HT_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.9 Heart rate

6.9.1 HRProfile_Init()

Description

The application calls this function to initialize the heart rate profile. The initialization procedure returns BLE_STATUS_SUCCESS if started successfully or BLE_STATUS_FAILED if not. On successful initialization of the profile, the application is notified through the event EVT_HRS_INITIALIZED through the registered callback.

Parameters

Feature support flag: the characteristic/feature mask supported by the heart rate profile during initialization. The various characteristics mask supported by the profile are:

1. BODY_SENSOR_LOCATION_SUPPORT_MASK (0x01)
2. ENERGY_EXTENDED_INFO_SUPPORT_MASK (0x02)

- BLE_CALLBACK_FUNCTION_TYPE: the callback function to be registered by the heart rate profile for notification/communication to the main BLE profile.
- Sensor location value: the Value for the body sensor location.

6.9.2 HR_Sensor_Make_Discoverable()

Description

This command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

- useBoundedDeviceList: set this flag to TRUE '1' if Profile needs to advertise to devices already bonded; otherwise, set it to FALSE '0'.

6.9.3 HRProfile_Send_HRM_Value()

Description

This function or command is called by the application to send the heart rate measurement value to the collector. The procedure to send a heart rate measurement value returns BLE_STATUS_SUCCESS if started successfully or BLE_STATUS_FAILED if not. When the measurement value is sent successfully, the application is notified through the event EVT_HRS_CHAR_UPDATE_CMPLT.

Parameters

- heartRateVal: The heart rate measurement structure with the following members:

1. valueformat – indicates the format of the heart measurement value.
   '0' if UINT8
   '1' if UINT16
2. sensorContact – this field indicates whether the sensor is in contact with the body
   '0' is no or poor contact.
   '1' contact o.k.
3. energyExpendedStatus – indicates whether the EE field is present in the current characteristic value.
   '0' not present.
   '1' present.
4. rrIntervalStatus – indicates whether RR interval values are present in the current characteristic value
   '0' not present.
   '1' present.
5. heartRateValue – the heart rate measurement value.
6. energyExpended – the energy expended value.
7. numOfRRIntervalvalues – the maximum length of RR interval values allowed is nine. If the maximum is exceeded, then only the last nine will be considered, assuming they correspond to the most recent collected data.
8. rrIntervalValues[9] – the buffer to hold the nine(9) most recent RR interval values provided by the application.

6.9.4 HRProfile_Set_Sensor_Contact_Support_Bit()

Description

The application should call this function before sending any data to the device in order to enable the sensor contact bit (BODY_SENSOR_LOCATION_SUPPORT_MASK) to include sensor contact information value in the heart rate measurement. It returns BLE_STATUS_SUCCESS when successfully set and BLE_STATUS_FAILED when not set.

6.9.5 HRProfile_Set_Body_Sensor_Location()

Description

Updates the body sensor location characteristic with the value provided. This should be called by the application when not in a connection; it returns BLE_STATUS_SUCCESS if successful.

Parameters

- bdsValue: The position of the body sensor location. The valid sensor location values are: BODY_SENSOR_LOCATION_OTHER(0x00) BODY_SENSOR_LOCATION_WRIST (0x02) BODY_SENSOR_LOCATION_FINGER(0x03)
- BODY_SENSOR_LOCATION_HAND (0x04)
  BODY_SENSOR_LOCATION_EAR_LOBE (0x05)
  BODY_SENSOR_LOCATION_FOOT (0x06)

6.9.6 HRProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.10 Human interface device profile

6.10.1 HidDevice_Init()

Description

Initializes the Hid profile. It returns BLE_STATUS_SUCCESS if the procedure is started successfully, or BLE_STATUS_FAILED if not. The application is notified through the event EVT_HID_INITIALIZED on successful initialization.

Parameters

1. numOfHIDServices – number of HID services to be exposed in the profile.
2. HidServiceData – pointer to the structure tApplDataForHidServ containing the configuration parameters provided by the application at the time of Initialization.
3. NumOfBatteryServices – number of battery services to be exposed in the profile
4. scanParamServiceSupport – adds the scan parameters service during hid initialization if set to '1'.
5. scanRefreshCharSupport – adds the scan parameters refresh characteristics if set to '1'.
6. BLE_CALLBACK_FUNCTION_TYPE – callback function type to be called by the profile to notify the application of the events.

6.10.2 HidDevice_Make_Discoverable()

Description

The function puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

1. useBoundedDeviceList – if this is set to '1', advertising is performed on the devices already bonded using whitelist.

6.10.3 HidDevice_Update_Input_Report()

Description

This is called by the application to update the input report characteristic with the value specified in ipReportValue. It returns BLE_STATUS_SUCCESS if the update was successfully started, otherwise it returns error codes. On successful update of the characteristic, the application is notified through the event EVT_HID_UPDATE_CMPLT.

Parameters

1. hidServiceIndex – the index of the HID service whose report characteristic has to be updated.
2. IpReportIndex – the index of the input report to be updated
3. ipReportValLength – length of the input report
4. ipReportValue – value of the input report.

6.10.4 HidDevice_Update_Feature_Report()

Description

This is called by the application to update the feature report characteristic with the value specified in ftrReportValue. It returns BLE_STATUS_SUCCESS if the update was successfully started, otherwise it returns error codes. On successful update of the characteristic, the application is notified through the event EVT_HID_UPDATE_CMPLT.

Parameters

1. hidServiceIndex – the index of the HID service whose report characteristic has to be updated.
2. ftrReportIndex – the index of the input report to be updated
3. ftrReportValLength – length of the input report
4. ftrReportValue – value of the input report.

6.10.5 HidDevice_Update_Boot_Keyboard_Input_Report()

Description

This is called by the application to update the boot keyboard input report characteristic with the value specified in bootKbdIpReportValue. On successful update of the characteristic, the application is notified through the event EVT_HID_UPDATE_CMPLT.

Parameters

1. hidServiceIndex – the index of the HID service whose report characteristic must be updated.
2. BootKbdIpReportValLength – length of the boot keyboard input report
3. bootKbdIpReportValue – value of the boot keyboard input report

6.10.6 HidDevice_Update_Boot_Mouse_Input_Report()

Description

This is called by the application to update the boot mouse input report characteristic with the value specified in bootMouseIpReportValue. On successful update of the characteristic, the application is notified through the event EVT_HID_UPDATE_CMPLT.

Parameters

1. hidServiceIndex – the index of the HID service whose report characteristic must be updated.
2. BootMouseIpReportValLength – length of the boot mouse input report
3. bootMouseIpReportValue – value of the boot mouse input report

6.10.7 HidDevice_Update_Battery_Level()

Description

This is called by the application to start the update for the battery level characteristic. On successful update, the application is notified through the event EVT_BATTERY_LEVEL_UPDATE_CMPLT.

Parameters

1. batteryServiceIndex – the battery service whose characteristic must be updated.
2. BatteryLevel – value of the battery level characteristic.

6.10.8 HidDevice_Update_Scan_Refresh_Char()

Description

This is called by the application to start the update for the scan refresh characteristic. On successful update, the application is notified through the event EVT_SCAN_REFRESH_UPDATE_CMPLT.

Parameters

1. scanRefresh – the value of the scan refresh characteristic

6.10.9 Allow_BatteryLevel_Char_Read()

Description

This is called by the application when it receives an event EVT_BATT_LEVEL_READ_REQ from the profile. When EVT_BATT_LEVEL_READ_REQ is received, the application should first update the battery level characteristic if required and then call this function. The stack blocks the read response until this function is called by the application. For more details see the event description for EVT_BATT_LEVEL_READ_REQ.

Parameters

1. 1.batteryServiceIndex – the battery service whose characteristic must be updated.

6.10.10 HIDProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.11 Phone alert client

6.11.1 PAC_Init()

Description

Initializes the Phone Alert Status profile for client role and registers the Phone Alert Profile with the main profile. It returns BLE_STATUS_SUCCESS if the procedure is successfully started. The application is notified through the event EVT_PAC_INITIALIZED on successful initialization.

Parameters

1. BLE_CALLBACK_FUNCTION_TYPE – callback function to be called by the profile to notify the application of the events.

6.11.2 PAC_Add_Device_To_WhiteList()

Description

The application calls this function to add the devices to whitelist.

Parameters

1. bdAddr – the address of the peer device that must be added to the whitelist.

6.11.3 PAC_Advertize()

Description

The application calls this function to put the device into discoverable mode.

6.11.4 PAC_Configure_Ringer()

Description

The application calls this to write the ringer mode to the phone alert server. It returns BLE_STATUS_SUCCESS if the parameters are valid and the procedure has been started successfully, otherwise it returns error codes.

Parameters

ringerMode – the ringer mode to be set. The valid ringer modes are:

SILENT_MODE (0x01)

MUTE_ONCE (0x02)

CANCEL_SILENT_MODE (0x03)

6.11.5 PAC_Read_AlertStatus()

Description

When this function is called by the application, the profile starts a GATT procedure to read the characteristic value. The value read is returned via the event EVT_PAC_ALERT_STATUS to the application through the callback.

6.11.6 PAC_Read_RingerSetting()

Description

When this function is called by the application, the profile starts a GATT procedure to read the characteristic value. The value read is returned via the event EVT_PAC_RINGER_SETTING to the application.

6.11.7 PACProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.12 Proximity monitor

6.12.1 ProximityMonitor_Init()

Description

Initializes the proximity profile in the monitor role. It returns BLE_STATUS_SUCCESS if the procedure is started successfully and notifies the application through the event EVT_PM_INITIALIZED.

Parameters

1. BLE_CALLBACK_FUNCTION_TYPE: the callback function to be called by the profile to notify the application of the profile specific events.

6.12.2 ProximityMonitor_Make_Discoverable()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

1. useBoundedDeviceList – set this to '1' if advertising is to be performed on the devices already bonded.

6.12.3 ProximityMonitorProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.13 Proximity reporter

6.13.1 ProximityReporter_Init()

Description

Initializes the proximity profile in reporter role. Returns BLE_STATUS_SUCCESS if the procedure is started successfully and notifies the application through the event EVT_PR_INITIALIZED.

Parameters

1. BLE_CALLBACK_FUNCTION_TYPE: the callback function to be called by the profile to notify the application of the profile specific events.
2. immAlertTxPowerSupport: set this to a non-zero value if the TX power level and immediate alert services are to be exposed by the profile, otherwise set it to '0'.

6.13.2 ProximityReporter_Make_Discoverable()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

1. useBoundedDeviceList – set this to '1' if advertising is to be performed on the devices already bonded.

6.13.3 ProximityReporterProfile_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.14 Time client

6.14.1 TimeClient_Init()

Description

Initializes the time profile in client role. The application is notified through the event EVT_TC_INITIALIZED on successful initialization.

Parameters

1. BLE_CALLBACK_FUNCTION_TYPE – callback function to be called by the profile to notify the application of the events.

6.14.2 TimeClient_Make_Discoverable()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

1. useBoundedDeviceList – set this to '1' if advertising is to be performed on the devices already bonded.

6.14.3 TimeClient_Get_Current_Time()

Description

The application calls this function to read the current time characteristic. Once the read is complete, the read value is sent to the application through an event EVT_TC_READ_CUR_TIME_CHAR. The event data contains the following fields:

1. byte 0 and 1 – year
2. byte 2 – month
3. byte 3 – date
4. byte 4 – hours
5. byte 5 – minutes
6. byte 6 – seconds
7. byte 7 – day_of_week
8. byte 8 – fractions256
9. byte 9 – adjust_reason

6.14.4 TimeClient_Get_Local_Time_Information()

Description

The application calls this function to read the local time information characteristic. Once the read is complete, the read value is sent to the application through an event called EVT_TC_READ_LOCAL_TIME_INFO_CHAR.

6.14.5 TimeClient_Get_Time_Accuracy_Info_Of_Server()

Description

The application calls this function to read the reference time information characteristic. Once the read is complete, the read value is sent to the application through an event called EVT_TC_READ_REF_TIME_INFO_CHAR.

6.14.6 TimeClient_Get_Next_DST_Change_Time()

Description

The application calls this function to read the time with DST information characteristic on the server. Once the read is complete, the read value is sent to the application through an event called EVT_TC_READ_TIME_WITH_DST_CHAR.

6.14.7 TimeClient_Get_Server_Time_Update_State()

Description

The application should call this function to read the time update state characteristic on the server. Once the read is complete, the read value is sent to the application through an event called EVT_TC_READ_TIME_UPDATE_STATE_CHAR.

6.14.8 TimeClient_Update_Reference_Time_On_Server()

Description

The application calls this function to write the time update control point characteristic on the server.

Parameters

1. ctlValue: writing a value of 1 starts the update procedure; a value of 0 cancels the update procedure.

6.14.9 TimeClient_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

6.15 Time server

6.15.1 TimeServer_Init()

Description

Initializes the time profile in server role. On successful initialization, the application is notified through the event EVT_TS_INITIALIZED.

Parameters

1. BLE_CALLBACK_FUNCTION_TYPE – callback function to be called by the profile to notify the application of the events.
2. ServicesToBeSupported – the bit mask of the optional services to be supported. Below are the bitmask values:
   a) NEXT_DST_CHANGE_SERVICE_BITMASK (0x01)
   b) REFERENCE_TIME_UPDATE_SERVICE_BITMASK (0x02)

6.15.2 TimeServer_Make_Discoverable()

Description

The command puts the device into discoverable mode. BLE_STATUS_SUCCESS is returned if the command to put the device into discoverable mode was issued successfully.

Parameters

1. useBoundedDeviceList- this is set to '1' if advertising is to be done to the devices already bonded.

6.15.3 TimeServer_Update_Current_Time_Value()

Description

This is called by the application to update the current time characteristic with the timeValue On completion of the update, the application is notified through the event EVT_TS_CHAR_UPDATE_CMPLT.

Parameters

- timeValue – current time structure containing the following structure members.

1. year
2. month
3. date
4. hours
5. minutes
6. seconds
7. day_of_week
8. fractions256
9. adjustReason: the adjust reason parameter can take any of the below values:
   a) ADJUST_REASON_NO_REASON(0x00)
   b) ADJUST_REASON_MANUAL_TIME_UPDATE (0x01)
   c) ADJUST_REASON_EXTERNAL_REFERENCE_TIME_UPDATE (0x02)
   d) ADJUST_REASON_CHANGE_OF_TIME_ZONE (0x04)
   e) ADJUST_REASON_CHANGE_OF_DST (0x08)

6.15.4 TimeServer_Update_Local_Time_Information()

Description

This is called by the application to update the localTimeInfo characteristic with the value specified and notifies the application with the EVT_TS_CHAR_UPDATE_CMPLT event on successful update.

Parameters

- localTimeInfo – local time structure containing the following structure members:
  – timeZone
  – dstOffset

6.15.5 TimeServer_Update_Reference_Time_Information()

Description

This is called by the application to update the reference time information characteristic with the value specified and the application is notified with the EVT_TS_START_REFTIME_UPDATE event on successful update.

Parameters

- refTimeInfo – the new reference time information. This structure containing the following members:
  – source
  – accuracy
  – daysSinceUpdate
  – hoursSinceUpdate

6.15.6 TimeServer_Update_Next_DST_Change()

Description

This is called by the application to update the next DST change characteristic with the value specified.

Parameters

- timeDST – the new DST information. This structure contains the following members:
  - year
  - month
  - date
  - hours
  - minutes
  - seconds
  - dstOffset

6.15.7 TimeServer_StateMachine()

Description

The application calls this function for checking current main profile and profile state, substate and performing related actions and consequent states updates.

Parameters

None

Parameters

- value: glucose feature value
  a) GLUCOSE_FEATURE_LOW_BATTERY_DETECTION_IS_SUPPORTED (0x0001)
  b) GLUCOSE_FEATURE_SENSOR_MALFUNCTION_DETECTION_IS_SUPPORTED (0x0002)
  c) GLUCOSE_FEATURE_SAMPLE_SIZE_IS_SUPPORTED (0x0004)
  d) GLUCOSE_FEATURE_SENSOR_STRIP_INSERTION_ERROR_IS_SUPPORTED (0x0008)
  e) GLUCOSE_FEATURE_SENSOR_STRIP_TYPE_ERROR_IS_SUPPORTED (0x0010)
  f) GLUCOSE_FEATURE_SENSOR_RESULT_HIGH_LOW_DETECTION_IS_SUPPORTED (0x0020)
  g) GLUCOSE_FEATURE_SENSOR_TEMPERATURE_HIGH_LOW_DETECTION_IS_SUPPORTED (0x0040)
  h) GLUCOSE_FEATURE_SENSOR_READ_INTERRUPT_DETECTION_IS_SUPPORTED (0x0080)
  i) GLUCOSE_FEATURE_GENERAL_DEVICE_FAULT_IS_SUPPORTED (0x0100)
  j) GLUCOSE_FEATURE_TIME_FAULT_IS_SUPPORTED (0x0200)
  k) GLUCOSE_FEATURE_MULTIPLE_BOND_IS_SUPPORTED (0x0400)

7 Profiles central roles: APIs interface and callbacks

This section describes the profiles central roles APIs interface and related user application callbacks. The user application callbacks are the functions with *_CB suffix. For a detailed description of each API parameters refer to the related header files and associated html documentation.

7.1 Alert notification client

7.1.1 ANC_Init (ancInitDevType param)

Init the Central device.

7.1.2 ANC_DeviceDiscovery (ancDevDiscType param)

Start the device discovery procedure.

7.1.3 ANC_SecuritySet (ancSecurityType param)

Setup the device security parameters.

7.1.4 ANC_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the ANC_SecuritySet().

7.1.5 ANC_Clear_Security_Database ()

Clear Security database.

7.1.6 ANC_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.1.7 ANC_DeviceConnection (ancConnDevType param)

Start the Connection procedure with the peer device.

7.1.8 ANC_DeviceDisconnection (void)

Start the device disconnection procedure.

7.1.9 ANC_ConnectionParameterUpdateRsp (uint8_t accepted, ancConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters, it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and

characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.1.10 ANC_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.1.11 ANC_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.1.12 ANC_ConnConf (ancConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.1.13 ANC_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.1.14 void ANC_StateMachine (void)

Run the profile central state machine.

7.1.15 ANC_Start_New_Alert_Client_Char_Descriptor_Discovery (void)

Discovery New Alert Client Characteristic Descriptor.

7.1.16 ANC_Start_Unread_Alert_Status_Client_Char_Descriptor_Discovery (void)

Discovery Unread Alert Status Client Characteristic Descriptor.

7.1.17 ANC_Enable_Disable_New_Alert_Notification (uint8_t enable)

Enable, Disable the new alert characteristic notification

7.1.18 ANC_Enable_Disable_Unread_Alert_Status_Notification (uint8_t enable)

Enable, Disable the unread alert status characteristic notification

7.1.19 ANC_Write_Control_Point (tCommandID command, tCategoryID category)

It allows to write the control point characteristic

7.1.20 ANC_CP_Check_Write_Response_Handler (uint8_t err_code)

It checks the write response status and error related to the Alert Notification Control Point (ANCP) write request to reset energy expended. It should be called within the Profile event

handler, on EVT_BLUE_GATT_ERROR_RESP, EVT_BLUE_GATT_PROCEDURE_COMPLETE.

7.1.21 ANC_Read_New_Alert_Category (void)

Read Supported New Alert Category characteristic.

7.1.22 ANC_Read_New_Alert_Category_CB (void)

Store & analyse the Characteristic Value for Supported New Alert Category Char.

7.1.23 ANC_Read_Unread_Alert_Status_Category (void)

Read Supported Unread Alert Status Category characteristic.

7.1.24 ANC_Read_Unread_Alert_Status_Category_CB (void)

Store & analyse the Characteristic Value for Supported Unread Alert Status Category Char.

7.1.25 ANC_New_Alert_Notification_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

Callback function: it stores the New Alert Characteristics Notification.

7.1.26 ANC_Unread_Alert_Status_Notification_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

Callback function: it stores the Unread Alert Status Characteristics Notification.

7.1.27 ANC_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

Callback function: it contains all the information of the device discovered during the central procedure.

7.1.28 ANC_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

Callback function: it contains all the information of the service discovery procedure.

7.1.29 ANC_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

Callback function: it contains the status of the connection procedure.

7.1.30 ANC_ConnectionParameterUpdateReq_CB (ancConnUpdateParamType *param)

Callback function: it called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by ANC_ConnectionParameterUpdateRsp().

7.1.31 ANC_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

Callback function: it contains all the information of the characteristics of service discovery procedure.

7.1.32 ANC_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

Callback function: it contains all the information of the characteristics descriptor of a service.

7.1.33 ANC_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

Callback function: it returns the read characteristic value of the connected peer device. It also returns the status of the read procedure.

7.1.34 ANC_PinCodeRequired_CB (void)

Callback function: it is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the ANC_SendPinCode() function to set the pin code.

7.1.35 ANC_EnableNotification_CB (uint8_t status)

Callback function: it returns the status of the characteristic enable notification procedure on the connected peer device.

7.1.36 ANC_FullConfError_CB (uint8_t error_type, uint8_t code)

Callback function: it called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the ANC_ConnConf() function call.

7.1.37 ANC_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

Callback function: it is called when an Intermediate Cuff Pressure notification is received.

7.1.38 ANC_Pairing_CB (uint16_t conn_handle, uint8_t status)

Callback function: it called for providing the pairing procedure status.

7.1.39 ANC_CP_Write_Response_CB (uint8_t err_code)

Callback function: it called when as response to a ANCP write procedure is received just to inform user application (error messages are already raised by device code).

7.2 Alert notification server

7.2.1 ANS_Init (ansInitDevType param)

Init the Central device.

7.2.2 ANS_DeviceDiscovery (ansDevDiscType param)

Start the device discovery procedure.

7.2.3 ANS_SecuritySet (ansSecurityType param)

Setup the device security parameters.

7.2.4 ANS_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the ANS_SecuritySet().

7.2.5 ANS_Clear_Security_Database ()

Clear Security database.

7.2.6 ANS_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.2.7 ANS_DeviceConnection (ansConnDevType param)

Start the Connection procedure with the peer device.

7.2.8 ANS_DeviceDisconnection (void)

Start the device disconnection procedure.

7.2.9 ANS_ConnectionParameterUpdateRsp (uint8_t accepted, ansConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters, it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.2.10 ANS_Add_Services_Characteristics (void)

This function adds Alert Notiifcation Server service & related characteristics.

7.2.11 ANS_Update_New_Alert_Category (uint8_t len, uint8_t *category)

This function updates the alert category characteristic with the new bitmask. The supported categories have to be constant when in connection.

7.2.12 ANS_Update_Unread_Alert_Category (uint8_t len, uint8_t *category)

This function updates the unread alert status category characteristic with the new bitmask. The supported categories have to be constant when in connection.

7.2.13 ANS_Update_New_Alert (tCategoryID categoryID, uint8_t alertCount, tTextInfo textInfo)

This function updates the number of new alerts for the category specified in the new alert characteristic.

7.2.14 ANS_Update_Unread_Alert_Status (tCategoryID categoryID, uint8_t alertCount)

This function updates the number of unread alerts for the category specified in the unread alert status characteristic.

7.2.15 ANC_Handle_ControlPoint_Write (uint8_t *attVal)

This function writes the alert control point.

7.2.16 ANS_StateMachine (void)

Run the profile central state machine.

7.2.17 ANS_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.2.18 ANS_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.2.19 ANS_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.2.20 ANS_ConnectionParameterUpdateReq_CB (ansConnUpdateParamType *param)

This callback is called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by ANS_ConnectionParameterUpdateRsp().

7.2.21 ANS_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the ANS_SendPinCode() function to set the pin code.

7.2.22 ANS_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.3 Blood pressure collector

7.3.1 BPC_Init (bpcInitDevType param)

Init the Central device.

7.3.2 BPC_DeviceDiscovery (bpcDevDiscType param)

Start the device discovery procedure.

7.3.3 BPC_SecuritySet (bpcSecurityType param)

Setup the device security parameters.

7.3.4 BPC_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the BPC_SecuritySet().

7.3.5 BPC_Clear_Security_Database ()

Clear Profile Security database.

7.3.6 BPC_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.3.7 BPC_DeviceConnection (bpcConnDevType param)

Start the Connection procedure with the peer device.

7.3.8 BPC_DeviceDisconnection (void)

Start the device disconnection procedure.

7.3.9 BPC_ConnectionParameterUpdateRsp (uint8_t accepted, bpcConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters, it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.3.10 BPC_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.3.11 BPC_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.3.12 BPC_ConnConf (bpcConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.3.13 void BPC_StateMachine (void)

Run the profile internal state machine. This function is required when the application executes the BPC_ConnConf().

7.3.14 BPC_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.3.15 BPC_Start_Blood_Pressure_Measurement_Client_Char_Descriptor_Discovery (void)

Discovery Blood Pressure Measurement Client Characteristic Descriptor.

7.3.16 BPC_Start_ICP_Client_Char_Descriptor_Discovery (void)

Discovery Intermediate Cuff Pressure Client Characteristic Descriptor.

7.3.17 BPC_ReadDISManufacturerNameChar (void)

Read Device Info Manufacturer Name Characteristic.

7.3.18 BPC_ReadDISModelNumberChar (void)

Read Device Info Model Number Characteristic.

7.3.19 BPC_ReadDISSystemIDChar (void)

Read Device Info System ID Characteristic.

7.3.20 BPC_Enable_BP_Measurement_Char_Indication (void)

Enable Blood Pressure Measurement Characteristic for Indication.

7.3.21 BPC_Enable_ICP_Char_Notification (void)

Enable Intermediate Cuff Pressure Characteristic for Notification.

7.3.22 BPC_Read_BP_Feature (void)

Read Blood Pressure Feature characteristic.

7.3.23 BPC_Read_BP_Feature_CB (void)

Store & analyse the Characteristic Value for Blood Pressure feature Char.

7.3.24 BPC_BP_Measurement_Indication_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores the Blood Pressure Measurement Characteristics Indication.

7.3.25 BPC_ICP_Notification_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores the Intermediate Cuff Pressure Characteristics Notification.

7.3.26 BPC_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.3.27 BPC_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.3.28 BPC_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.3.29 BPC_ConnectionParameterUpdateReq_CB (bpcConnUpdateParamType *param)

This callback is called when the device receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by BPC_ConnectionParameterUpdateRsp().

7.3.30 BPC_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.3.31 BPC_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service.

7.3.32 BPC_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the read characteristic value of the connected peer device. Returns, also, the status of the read procedure.

7.3.33 BPC_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the BPC_SendPinCode() function to set the pin code.

7.3.34 BPC_EnableIndication_CB (uint8_t status)

This callback returns the status of the characteristic enable indication procedure on the connected peer device.

7.3.35 BPC_EnableNotification_CB (uint8_t status)

This callback returns the status of the characteristic enable notification procedure on the connected peer device.

7.3.36 BPC_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the BPC_ConnConf() function call.

7.3.37 BPC_IndicationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when a Blood Pressure Measurement indication is received.

7.3.38 BPC_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when an Intermediate Cuff Pressure notification is received.

7.3.39 BPC_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.4 Find me locator

7.4.1 FML_Init (fmlInitDevType param)

Init the Central device.

7.4.2 FML_DeviceDiscovery (fmlDevDiscType param)

Start the device discovery procedure.

7.4.3 FML_SecuritySet (fmlSecurityType param)

Setup the device security parameters.

7.4.4 FML_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the FML_SecuritySet().

7.4.5 FML_Clear_Security_Database ()

Clear Glucose Collector Security database.

7.4.6 FML_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.4.7 FML_DeviceConnection (fmlConnDevType param)

Start the Connection procedure with the peer device.

7.4.8 FML_DeviceDisconnection (void)

Start the device disconnection procedure.

7.4.9 FML_ConnectionParameterUpdateRsp (uint8_t accepted, fmlConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters , it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.4.10 FML_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.4.11 FML_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.4.12 FML_ConnConf (fmlConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.4.13 FML_StateMachine (void)

Run the Glucose Collector internal state machine. This function is required when the application executes the FML_ConnConf().

7.4.14 FML_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.4.15 FML_ALert_Target (uint8_t alertLevel)

It causes an alert the Find Me Target device.

7.4.16 FML_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.4.17 FML_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.4.18 FML_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.4.19 FML_ConnectionParameterUpdateReq_CB (fmlConnUpdateParamType *param)

This callback is called when the Glucose Collector receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by FML_ConnectionParameterUpdateRsp().

7.4.20 FML_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.4.21 FML_PinCodeRequired_CB (void)

This function is called from the Glucose Collectorprofile when the MITM pin code is required from the peer device. Inside this function the application needs to call the FML_SendPinCode() function to set the pin code.

7.4.22 FML_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the FML_ConnConf() function call.

7.4.23 FML_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.BPC_Init (bpcInitDevType param)

Init the Central device.

7.5 Find me target

7.5.1 FMT_Init (fmtInitDevType param)

Init the Central device.

7.5.2 FMT_DeviceDiscovery (fmtDevDiscType param)

Start the device discovery procedure.

7.5.3 FMT_SecuritySet (fmtSecurityType param)

Setup the device security parameters.

7.5.4 FMT_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the FMT_SecuritySet().

7.5.5 FMT_Clear_Security_Database ()

Clear Security database

7.5.6 FMT_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.5.7 FMT_DeviceConnection (fmtConnDevType param)

Start the Connection procedure with the peer device.

7.5.8 FMT_DeviceDisconnection (void)

Start the device disconnection procedure.

7.5.9 FMT_ConnectionParameterUpdateRsp (uint8_t accepted, fmtConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.5.10 FMT_Add_Services_Characteristics (void)

Add the Profile service & related characteristics.

7.5.11 FMT_StateMachine (void)

Run the profile central state machine. Is it needed? TBR???

7.5.12 FMT_Set_Alert_Level_Value (uint8_t value)

Set the alert level characteristic value.

7.5.13 uint8_t FMT_Get_Alert_Level_Value (void)

Get the currente alert level value.

7.5.14 FMT_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.5.15 FMT_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.5.16 FMT_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.5.17 FMT_ConnectionParameterUpdateReq_CB (fmtConnUpdateParamType *param)

This callback is called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by FMT_ConnectionParameterUpdateRsp().

7.5.18 FMT_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the FMT_SendPinCode() function to set the pin code.

7.5.19 FMT_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.5.20 FMT_Alert_Level_Value_CB (uint8_t alert_level)

User Callback which is called each time alert level value is received: user specific implementation action should be done accordingly.

7.6 Glucose collector

7.6.1 GL_Collector_Init (glcInitDevType param)

Init the Central device.

7.6.2 GL_Collector_DeviceDiscovery (glcDevDiscType param)

Start the device discovery procedure.

7.6.3 GL_Collector_SecuritySet (glcSecurityType param)

Setup the device security parameters.

7.6.4 GL_Collector_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the GL_Collector_SecuritySet().

7.6.5 GL_Collector_Clear_Security_Database ()

Clear Glucose Collector Security database.

7.6.6 GL_Collector_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.6.7 GL_Collector_DeviceConnection (glcConnDevType param)

Start the Connection procedure with the peer device.

7.6.8 GL_Collector_DeviceDisconnection (void)

Start the device disconnection procedure.

7.6.9 GL_Collector_ConnectionParameterUpdateRsp (uint8_t accepted, glcConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.6.10 GL_Collector_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.6.11 GL_Collector_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.6.12 GL_Collector_ConnConf (glcConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.6.13 GL_Collector_StateMachine (void)

Run the Glucose Collector internal state machine. This function is required when the application executes the GL_Collector_ConnConf().

7.6.14 GL_Collector_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.6.15 GL_Collector_Start_Glucose_Measurement_Characteristic_Descriptor_Discovery (void)

Discovery Glucose Measurement characteristic descriptor.

7.6.16 GL_Collector_Start_Glucose_Measurement_Context_Characteristic_Descriptor_Discovery (void)

Discovery Glucose Measurement Context characteristic descriptor.

7.6.17 GL_Collector_Start_RACP_Characteristic_Descriptor_Discovery (void)

Discovery Glucose RACP characteristic descriptor.

7.6.18 GL_Collector_Send_RACP (uint8_t racp_opcode, uint8_t racp_operator, uint8_t racp_filter_type, tfilterTypeParameter *racp_filter_parameter_1, tfilterTypeParameter *racp_filter_parameter_2)

Perform the requested RACP procedure.

7.6.19 GL_Collector_ReadFeatureChar (void)

Read Glucose Sensor Feature Characteristic.

7.6.20 GL_Collector_ReadDISManufacturerNameChar (void)

Read Device Info Manufacturer Name Characteristic.

7.6.21 GL_Collector_ReadDISModelNumberChar (void)

Read Device Info Model Number Characteristic.

7.6.22 GL_Collector_ReadDISSystemIDChar (void)

Read Device Info System ID Characteristic.

7.6.23 GL_Collector_Enable_Glucose_Measurement_Char_Notification (void)

Enable Glucose Measurement Characteristic Notification.

7.6.24 GL_Collector_Enable_Glucose_Measurement_Context_Char_Notification (void)

Enable Glucose Measurement Context Characteristic Notification.

7.6.25 GL_Collector_Enable_Glucose_RACP_Char_Indication (void)

Enable Glucose RACP Characteristic Indication.

7.6.26 BOOL GL_Collector_Util_Perform_RACP_Post_processing (void)

Utility for allowing post processing of received notifications to a single RACP procedure.

7.6.27 GL_Collector_PostProcess_RACP_Notification_SM (void)

State Machine allowing to handle the post processing utility of received notifications to a single RACP procedure.

7.6.28 GL_Collector_RACP_Check_Write_Response (uint8_t err_code)

It checks the write response status and error related to the RACP procedure It should be called within the Profile event handler, on EVT_BLUE_GATT_PROCEDURE_COMPLETE.

7.6.29 GL_Collector_RACP_Indications (uint16_t attr_handle, uint8_t data_lenght, uint8_t *value)

It handles the RACP Characteristic Indication from a Glucose Sensor.

7.6.30 GL_Collector_ProcedureTimeoutHandler (void)

RACP write procedure timed out handler.

7.6.31 GL_Collector_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.6.32 GL_Collector_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.6.33 GL_Collector_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.6.34 GL_Collector_ConnectionParameterUpdateReq_CB (glcConnUpdateParamType *param)

This callback is called when the Glucose Collector receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by GL_Collector_ConnectionParameterUpdateRsp().

7.6.35 GL_Collector_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.6.36 GL_Collector_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service.

7.6.37 GL_Collector_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the read characteristic value of the connected peer device. Returns, also, the status of the read procedure.

7.6.38 GL_Collector_PinCodeRequired_CB (void)

This function is called from the Glucose Collector profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the GL_Collector_SendPinCode() function to set the pin code.

7.6.39 GL_Collector_EnableNotification_CB (uint8_t status)

This callback returns the status of the characteristic enable notification procedure on the connected peer device.

7.6.40 GL_Collector_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the GL_Collector_ConnConf() function call.

7.6.41 GL_Collector_RACP_Write_Response_CB (uint8_t err_code)

This callback is called when as response to a RACP write procedure is received just to inform user application (error messages are already raised by glucose collector code).

7.6.42 GL_Collector_RACP_Received_Indication_CB (uint8_t racp_response, uint8_t value, uint8_t num_records)

This callback is called when a RACP Indication to a RACP write procedure is received just to inform user application (error messages are already raised by glucose collector code).

7.6.43 GL_Collector_EnableNotificationIndication_CB (uint8_t status)

This callback returns the status of Characteristic notification/indication procedure.

7.6.44 GL_Collector_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when a Glucose Measurement or Measurement Context Notification is received as consequence of a RACP procedure

7.6.45 GL_Collector_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.7 Health Thermometer collector

7.7.1 HT_Collector_Init (htcInitDevType param)

Init the Central device.

7.7.2 HT_Collector_DeviceDiscovery (htcDevDiscType param)

Start the device discovery procedure.

7.7.3 HT_Collector_SecuritySet (htcSecurityType param)

Setup the device security parameters.

7.7.4 HT_Collector_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the HT_Collector_SecuritySet().

7.7.5 HT_Collector_Clear_Security_Database ()

Clear Glucose Collector Security database.

7.7.6 HT_Collector_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.7.7 HT_Collector_DeviceConnection (htcConnDevType param)

Start the Connection procedure with the peer device.

7.7.8 HT_Collector_DeviceDisconnection (void)

Start the device disconnection procedure.

7.7.9 HT_Collector_ConnectionParameterUpdateRsp (uint8_t accepted, htcConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters , it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.7.10 HT_Collector_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.7.11 HT_Collector_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.7.12 HT_Collector_ConnConf (htcConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.7.13 HT_Collector_StateMachine (void)

Run the Glucose Collector internal state machine. This function is required when the application executes the HT_Collector_ConnConf()

7.7.14 HT_Collector_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.7.15 HT_Collector_Start_Temperature_Measurement_Client_Char_Descriptor_Discovery (void)

Discovery Temperature Measurement Client Characteristic Descriptor.

7.7.16 HT_Collector_Start_Intermediate_Temperature_Client_Char_Descriptor_Discovery (void)

Discovery Intermediate Temperature Client Characteristic Descriptor.

7.7.17 HT_Collector_Start_Measurement_Interval_Client_Char_Descriptor_Discovery (void)

Discovery Measurement Interval Client Characteristic Descriptor.

7.7.18 HT_Collector_ReadDISManufacturerNameChar (void)

Read Device Information Service Characteristics.

7.7.19 HT_Collector_ReadDISModelNumberChar (void)

Read Device Info Model Number Characteristic.

7.7.20 HT_Collector_ReadDISSystemIDChar (void)

Read Device Info System ID Characteristic.

7.7.21 HT_Collector_Enable_Temperature_Measurement_Char_Indication (void)

Enable Temperature Measurement Characteristic for Indication.

7.7.22 HT_Collector_Enable_Intermediate_Temperature_Char_Notification (void)

Enable Intermediate Temperature Characteristic for Notification.

7.7.23 HT_Collector_Enable_Measurement_Interval_Char_Indication (void)

Enable Measurement Interval Characteristic for Indication.

7.7.24 HT_Collector_Read_Measurement_Interval (void)

Read Measurement Interval characteristic.

7.7.25 HT_Collector_Write_Measurement_Interval (uint16_t writeValue)

Write Measurement Interval characteristic.

7.7.26 HT_Collector_Read_Measurement_Interval_Valid_Range_Descr (void)

Read Measurement Interval Valid Range descriptor.

7.7.27 HT_Collector_Read_Temperature_Type (void)

Read Temperature Type characteristic.

7.7.28 HT_Collector_Read_Measurement_Interval_CB (void)

Read the Characteristic Value for Measurement Interval Char.

7.7.29 HT_Collector_Measurement_Interval_Check_Write_Response_CB (uint8_t err_code)

It checks the write response status and error related to the Measurement_Interval characteristic write request It should be called within the Profile event handler, on EVT_BLUE_GATT_ERROR_RESP, EVT_BLUE_GATT_PROCEDURE_COMPLETE

7.7.30 HT_Collector_Read_Measurement_Interval_Valid_Range_CB (void)

Read the Value for Measurement Interval Valid Range Descriptor.

7.7.31 HT_Collector_Read_Temperature_Type_CB (void)

Read the Characteristic Value for Temperature Type Char.

7.7.32 HT_Collector_Temperature_Measurement_Indication_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores the Temperature Measurement Characteristics Indication.

7.7.33 HT_Collector_Intermediate_Temperature_Notification_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores the Intermediate Temperature Characteristics Notification.

7.7.34 HT_Collector_Measurement_Interval_Indication_CB (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores the Measurement Interval Characteristics Indication.

7.7.35 HT_Collector_Read_Measurement_Interval_Valid_Range_Descriptor (void)

Read Measurement Interval Valid Range Descriptor characteristic.

7.7.36 HT_Collector_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.7.37 HT_Collector_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.7.38 HT_Collector_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.7.39 HT_Collector_ConnectionParameterUpdateReq_CB (htcConnUpdateParamType *param)

This callback is called when the Glucose Collector receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by HT_Collector_ConnectionParameterUpdateRsp().

7.7.40 HT_Collector_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.7.41 HT_Collector_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service.

7.7.42 HT_Collector_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the read characteristic value of the connected peer device. Returns, also, the status of the read procedure.

7.7.43 HT_Collector_PinCodeRequired_CB (void)

This function is called from the Glucose Collector profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the HT_Collector_SendPinCode() function to set the pin code.

7.7.44 HT_Collector_EnableNotification_CB (uint8_t status)

This callback returns the status of the characteristic enable notification procedure on the connected peer device.

7.7.45 HT_Collector_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the HT_Collector_ConnConf() function call.

7.7.46 HT_Collector_EnableNotificationIndication_CB (uint8_t status)

This callback returns the status of Characteristic notification/indication procedure.

7.7.47 HT_Collector_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when a Notification is received.

7.7.48 HT_Collector_IndicationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when an Indication is received.

7.7.49 HT_Collector_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.8 Heart rate collector

7.8.1 HRC_Init (hrcInitDevType param)

Init the Central device.

7.8.2 HRC_DeviceDiscovery (hrcDevDiscType param)

Start the device discovery procedure.

7.8.3 HRC_SecuritySet (hrcSecurityType param)

Setup the device security parameters.

7.8.4 HRC_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the HRC_SecuritySet().

7.8.5 HRC_Clear_Security_Database ()

Clear Profile Security database.

7.8.6 HRC_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.8.7 HRC_DeviceConnection (hrcConnDevType param)

Start the Connection procedure with the peer device.

7.8.8 HRC_DeviceDisconnection (void)

Start the device disconnection procedure.

7.8.9 HRC_ConnectionParameterUpdateRsp (uint8_t accepted, hrcConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.8.10 HRC_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.8.11 HRC_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.8.12 HRC_ConnConf (hrcConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.8.13 HRC_StateMachine (void)

Run the profile internal state machine. This function is required when the application executes the HRC_ConnConf().

7.8.14 HRC_findHandles (uint16_t uuid_service, uint16_t *start_handle, uint16_t *end_handle)

Find the device services.

7.8.15 HRC_Start_Heart_Rate_Measurement_Characteristic_Descriptor_Discovery (void)

Discovery Heart Rate Measurement characteristic descriptor.

7.8.16 HRC_ReadDISManufacturerNameChar (void)

Read Device Info Manufacturer Name Characteristic.

7.8.17 HRC_Enable_HR_Measurement_Char_Notification (void)

Enable Heart Rate Measurement Characteristic Notification.

7.8.18 HRC_Read_Body_Sensor_Location (void)

Body Sensor Location characteristic read.

7.8.19 HRC_Write_HR_Control_Point (void)

Heart Rate Control Point characteristic write.

7.8.20 HRC_Read_Body_Sensor_Location_Handler (void)

Store & analyse the Characteristic Value for Body Sensor Location Char.

7.8.21 HRC_CP_Check_Write_Response_Handler (uint8_t err_code)

It checks the write response status and error related to the Heart Rate Control Point (HRCP) write request to reset energy expended It should be called within the Profile event handler, on EVT_BLUE_GATT_ERROR_RESP, EVT_BLUE_GATT_PROCEDURE_COMPLETE

7.8.22 HRC_Notification_Handler (uint16_t attr_handle, uint8_t data_length, uint8_t *value)

It stores and analyse the Heart Rate Measurement Characteristics Notifications.

7.8.23 HRC_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.8.24 HRC_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.8.25 HRC_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.8.26 HRC_ConnectionParameterUpdateReq_CB (hrcConnUpdateParamType *param)

This callback is called when the device receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by HRC_ConnectionParameterUpdateRsp().

7.8.27 HRC_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.8.28 HRC_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service

7.8.29 HRC_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the read characteristic value of the connected peer device. Returns, also, the status of the read procedure.

7.8.30 HRC_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the HRC_SendPinCode() function to set the pin code.

7.8.31 HRC_EnableNotification_CB (uint8_t status)

This callback returns the status of the characteristic enable notification procedure on the connected peer device.

7.8.32 HRC_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the HRC_ConnConf() function call.

7.8.33 HRC_CP_Write_Response_CB (uint8_t err_code)

This callback is called when as response to a RACP write procedure is received just to inform user application (error messages are already raised by device code).

7.8.34 HRC_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when a body sensor is received.

7.8.35 HRC_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.9 HID host device

7.9.1 HID_Init (hidInitDevType param)

Init the HID Host device.

7.9.2 HID_SecuritySet (hidSecurityType param)

Setup the device security parameters.

7.9.3 HID_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the HID_SecuritySet().

7.9.4 HID_ClearBondedDevices (void)

Delete the information database of the bonded devices.

7.9.5 HID_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.9.6 HID_DeviceDiscovery (hidDevDiscType param)

Start the device discovery procedure.

7.9.7 HID_DeviceConnection (hidConnDevType param)

Start the Connection procedure with the peer device.

7.9.8 HID_DeviceDisconnection (void)

Start the device disconnection procedure.

7.9.9 HID_ConnectionParameterUpdateRsp (uint8_t accepted, hidConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.9.10 HID_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.9.11 HID_GetIncludedBatterySerivces (void)

This function performs the relationship discovery to find included services, to discovery all Battery Services with characteristis described within a HID Service Report Map characteristic value.

7.9.12 HID_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.9.13 HID_DiscCharacDesc (uint16_t uuid_charac)

Discovery all the characteristic descriptors for the given characteristic.

7.9.14 HID_NumberOfReportDescriptor (void)

Returns the number of USB HID report descriptors present in the peer device.

7.9.15 HID_ReadReportDescriptor (uint8_t reportToRead, uint16_t *reportDataLen, uint8_t *reportData, uint16_t maxSize)

Read the USB HID report descriptor for the specified HID service in the peer device.

7.9.16 HID_ReadReportValue (uint16_t characToRead)

Read all the Report values for each HID Services present in the peripheral device.

7.9.17 HID_ReadHidInformation (void)

Read the HID Information value for all HID Services in the peer HID Device.

7.9.18 HID_ReadBatteryLevel (void)

Read the Battery Level values of the peer device.

7.9.19 HID_ReadBatteryClientCharacDesc (void)

Read the Battery Level Client Characteristic Descriptor.

7.9.20 HID_ReadPnPID (void)

Read the Pnp ID value of the Device Information Service.

7.9.21 HID_ReadBootReport (uint16_t bootReportUUID)

Read the Boot Report characteristic value.

7.9.22 HID_ReadBootReportClientCharacDesc (uint16_t bootReportUUID)

Read the Boot Report Client Characteristic Descriptor value.

7.9.23 HID_WriteScanIntervalWindowParam (uint16_t scanInterval, uint16_t scanWindow)

Write the Scan Interval Window parameters to the HID peer device.

7.9.24 HID_ScanRefreshNotificationStatus (uint8_t enabled)

Set the notification status for the Scan Refresh Characteristic.

7.9.25 HID_ConnConf (hidConnDevType connParam, hidConfDevType confParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.9.26 HID_StateMachine (void)

Run the HID host internal state machine. This function is required when the application executes the HID_ConnConf().

7.9.27 HID_GetReportId (uint8_t type, uint8_t *numReport, uint8_t *ID)

Return at the end of the configuration returns all the Report ID of the requested report Type.

7.9.28 HID_SetReport (uint8_t noResponseFlag, uint8_t type, uint8_t ID, uint8_t dataLen, uint8_t *data)

Set the report value in the peer device.

7.9.29 HID_GetReport (uint8_t type, uint8_t ID)

Set the report value in the peer device.

7.9.30 HID_SetControlPoint (uint8_t suspend)

Set the HID Control Point, to inform the HID Device that the HID Host is entering/exiting the suspended state.

7.9.31 HID_SetProtocol (uint8_t mode)

Set the HID Device Protocol Mode.

7.9.32 HID_GetProtocol (void)

Get the Protocol Mode: BOOT_PROTOCOL_MODE,REPORT_PROTOCOL_MODE.

7.9.33 HID_SetBootReport (uint8_t type, uint8_t dataLen, uint8_t *data)

Set the Boot Report value.

7.9.34 HID_SetNotificationStatus (uint8_t type, uint8_t enabled)

Set the Notification status enabled/disabled for the input report in the HID Device.

7.9.35 HID_SetHostMode (uint8_t mode)

Set the HID Host mode.

7.9.36 HID_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.9.37 HID_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.9.38 HID_ConnectionParameterUpdateReq_CB (hidConnUpdateParamType *param)

This callback is called when the HID host receives a connection parameter update request from the HID peripheral device connected. The response from the HID host device may be ignored or sent by HID_ConnectionParameterUpdateRsp().

7.9.39 HID_PinCodeRequired_CB (void)

This function is called from the HID host profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the HID_SendPinCode() function to set the pin code.

7.9.40 HID_PairingFailed_CB (void)

This function is called from the HID host profile when during the reconnection the pairing procedure fails. So, the application needs repeat the pairing procedure.

7.9.41 HID_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the HID_ConnConf() function call.

7.9.42 HID_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.9.43 HID_IncludedServices_CB (uint8_t status, uint8_t numIncludedServices, uint8_t *includedServices)

This callback contains all the information of the included service discovery procedure.

7.9.44 HID_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.9.45 HID_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service.

7.9.46 HID_ReadReportDescriptor_CB (uint8_t status)

This callback returns the status of the USB HID report descriptor read procedure. if the status is BLE_STATUS_SUCCESS the reportDataLen and reportData variable will contain the HID report descriptor information.

7.9.47 HID_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the data read from the peer device. Returns, also, the status of the read procedure.

7.9.48 HID_InformationData_CB (uint8_t status, uint16_t version, uint8_t countryCode, uint8_t remoteWake, uint8_t normallyConnectable)

This callback returns the HID Information data read for an HID Service. This callback will be called for each HID Service present in the HID peer Device.

7.9.49 HID_BetteryLevelData_CB (uint8_t status, uint8_t namespace, uint16_t description, uint8_t level)

This callback returns the level of the Battery services present in the HID Device. This callback will be called for each Battery Service present in the HID peer Device.

7.9.50 HID_BatteryClientCahracDesc_CB (uint8_t status, uint8_t notification, uint8_t indication)

This callback returns the value of the Battery Level client characteristic descriptor for the peer HID Device.

7.9.51 HID_PnPID_CB (uint8_t status, uint8_t vendorIdSource, uint16_t vendorId, uint16_t productId, uint16_t productVersion)

This callback returns the values of the PnP ID characteristic for the peer HID Device.

7.9.52 HID_BootReportValue_CB (uint8_t status, uint8_t dataLen, uint8_t *data)

This callback returns the report value for the Boot Keyboard and Mouse characteristics.

7.9.53 HID_ReadBootReportClientCharacDesc_CB (uint8_t status, uint8_t notification, uint8_t indication)

This callback returns the client characteristic descriptor value for the Boot Keyboard and Mouse characteristics.

7.9.54 HID_ProtocolMode_CB (uint8_t status, uint8_t mode)

This callback returns the protocol mode of the HID peer Device.

7.9.55 HID_SetProcedure_CB (uint8_t status)

This callback returns the status of the set procedure.

7.9.56 HID_NotificationChageStatus_CB (uint8_t status)

This callback reports the status of the enable/disable notification.

7.9.57 HID_ReportDataReceived_CB (uint8_t type, uint8_t id, uint8_t data_length, uint8_t *data_value)

This callbak reports the notification recevied from the HID Host.

7.10 Phone Alert Status Server

7.10.1 PASS_Init (passInitDevType param)

Init the Central device.

7.10.2 PASS_DeviceDiscovery (passDevDiscType param)

Start the device discovery procedure.

7.10.3 PASS_SecuritySet (passSecurityType param)

Setup the device security parameters.

7.10.4 PASS_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the PASS_SecuritySet().

7.10.5 PASS_Clear_Security_Database ()

Clear Security database.

7.10.6 PASS_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.10.7 PASS_DeviceConnection (passConnDevType param)

Start the Connection procedure with the peer device.

7.10.8 PASS_DeviceDisconnection (void)

Start the device disconnection procedure.

7.10.9 PASS_ConnectionParameterUpdateRsp (uint8_t accepted, passConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters, it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.10.10 PASS_Add_Services_Characteristics (void)

Add the Phone Alert Status service & related characteristics.

7.10.11 PASS_StateMachine (void)

Run the profile central state machine.

7.10.12 PASS_Alert_Control_Point_Handler (uint8_t alert_control_point_value)

Alert Control Point characteristic handler.

7.10.13 PASS_Set_AlertStatus_Value (uint8_t value)

Set the alert status characteristic value.

7.10.14 PASS_Set_RingerSetting_Value (uint8_t value)

Set the ringer setting characteristic value.

7.10.15 BOOL PASS_Get_Mute_Once_Silence_Ringer_Status (void)

It returns the mute_once_silence status.

7.10.16 PASS_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.10.17 PASS_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.10.18 PASS_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.10.19 PASS_ConnectionParameterUpdateReq_CB (passConnUpdateParamType *param)

This callback is called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by PASS_ConnectionParameterUpdateRsp().

7.10.20 PASS_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the PASS_SendPinCode() function to set the pin code.

7.10.21 PASS_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.10.22 PASS_Ringer_State_CB (uint8_t alert_control_point_value)

User defined callback which is called each time a Alert Control Point arrives: user specific implementation action should be done accordingly.

7.11 Proximity Monitor

7.11.1 PXM_Init (pxmInitDevType param)

Init the Proximity Monitore device.

7.11.2 PXM_DeviceDiscovery (pxmDevDiscType param)

Start the device discovery procedure.

7.11.3 PXM_SecuritySet (pxmSecurityType param)

Setup the device security parameters.

7.11.4 PXM_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the PXM_SecuritySet().

7.11.5 PXM_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.11.6 PXM_DeviceConnection (pxmConnDevType param)

Start the Connection procedure with the peer device.

7.11.7 PXM_DeviceDisconnection (void)

Start the device disconnection procedure.

7.11.8 PXM_ConnectionParameterUpdateRsp (uint8_t accepted, pxmConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.11.9 PXM_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.11.10 PXM_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.11.11 PXM_TxPwrLvl_DiscCharacDesc (void)

Discovery all the characteristic descriptors on the Tx Power Level service.

7.11.12 PXM_ConfigureLinkLossAlert (uint8_t level)

Configure the Alert Level of the Link Loss service.

7.11.13 PXM_ReadTxPower (void)

Read the Tx Power of the peer device.

7.11.14 PXM_EnableTxPowerNotification (void)

Enable the Tx Power Notification.

7.11.15 PXM_ConfigureImmediateAlert (uint8_t level)

Configure the Alert Level for the Immediate Alert service.

7.11.16 PXM_ConnConf (pxmConnDevType connParam, pxmConfDevType confParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.11.17 PXM_StateMachine (void)

Run the proximity monitor internal state machine. This function is required when the application executes the PXM_ConnConf().

7.11.18 PXM_GetRSSI (int8_t *value)

Return the RSSI value from the currrent connection.

7.11.19 PXM_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.11.20 PXM_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.11.21 PXM_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.11.22 PXM_ConnectionParameterUpdateReq_CB (pxmConnUpdateParamType *param)

This callback is called when the proximity monitor receives a connection parameter update request from the proximity reporter device connected. The response from the proximity central device may be ignored or sent by PXM_ConnectionParameterUpdateRsp().

7.11.23 PXM_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.11.24 PXM_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics descriptor of a service.

7.11.25 PXM_ConfigureAlert_CB (uint8_t status)

This callback returns the status of the Link Loss Alert Configuration procedure on the proximity reporter peer device.

7.11.26 PXM_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the Tx Power characteristic value of the proximity reporter peer device. Returns, also, the status of the read procedure.

7.11.27 PXM_LinkLossAlert (uint8_t level)

This function is called from the proximity monitor profile to alert the application that the peer device is disconnected, so, the application needs to alert the user with the right level.

7.11.28 PXM_PathLossAlert (uint8_t level)

This function is called from the proximity monitor profile to alert the application that the path loss exceeds a threshold during the connection with the peer device. So, the application needs to alert the user with the right level.

7.11.29 PXM_TxPowerNotificationReceived (int8_t data_value)

This function is called from the proximity monitor profile when a Tx Power notification is received.

7.11.30 PXM_PinCodeRequired_CB (void)

This function is called from the proximity monitor profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the PXM_SendPinCode() function to set the pin code.

7.11.31 PXM_EnableNotification_CB (uint8_t status)

This callback returns the status of the Tx Power enable notification procedure on the proximity reporter peer device.

7.11.32 PXM_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the PXM_ConnConf() function call.

7.12 Proximity Reporter

7.12.1 PXR_Init (pxrInitDevType param)

Init the Central device.

7.12.2 PXR_DeviceDiscovery (pxrDevDiscType param)

Start the device discovery procedure.

7.12.3 PXR_SecuritySet (pxrSecurityType param)

Setup the device security parameters.

7.12.4 PXR_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the PXR_SecuritySet().

7.12.5 PXR_Clear_Security_Database ()

Clear Security database.

7.12.6 PXR_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.12.7 PXR_DeviceConnection (pxrConnDevType param)

Start the Connection procedure with the peer device.

7.12.8 PXR_DeviceDisconnection (void)

Start the device disconnection procedure.

7.12.9 PXR_ConnectionParameterUpdateRsp (uint8_t accepted, pxrConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.12.10 PXR_Add_Services_Characteristics (void)

Add the Proximity reporter service & related characteristics.

7.12.11 PXR_StateMachine (void)

Run the profile central state machine.

7.12.12 PXR_Received_Alert_Handler (uint16_t attrHandle, uint8_t attValue)

It is called on when an alert level characteristic is written.

7.12.13 PXR_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.12.14 PXR_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.12.15 PXR_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.12.16 PXR_ConnectionParameterUpdateReq_CB (pxrConnUpdateParamType *param)

This callback is called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by PXR_ConnectionParameterUpdateRsp().

7.12.17 PXR_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the PXR_SendPinCode() function to set the pin code.

7.12.18 PXR_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.12.19 PXR_Alert_CB (uint8_t alert_type, uint8_t alert_value)

User defined callback which is called each time an alert is raised: user specific implementation action should be done accordingly.

7.13 Time Client

7.13.1 TimeClient_Init (tipcInitDevType param)

Init the Central device.

7.13.2 TimeClient_DeviceDiscovery (tipcDevDiscType param)

Start the device discovery procedure.

7.13.3 TimeClient_SecuritySet (tipcSecurityType param)

Setup the device security parameters.

7.13.4 TimeClient_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the TimeClient_SecuritySet().

7.13.5 TimeClient_Clear_Security_Database ()

Clear time client Security database.

7.13.6 TimeClient_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.13.7 TimeClient_DeviceConnection (tipcConnDevType param)

Start the Connection procedure with the peer device.

7.13.8 TimeClient_DeviceDisconnection (void)

Start the device disconnection procedure.

7.13.9 TimeClient_ConnectionParameterUpdateRsp (uint8_t accepted, tipcConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.13.10 TimeClient_ServicesDiscovery (void)

Start the service discovery procedure on the peer device.

7.13.11 TimeClient_DiscCharacServ (uint16_t uuid_service)

Start the characteristic discovery procedure on the peer device.

7.13.12 TimeClient_Start_Current_Time_Characteristic_Descriptor_Discovery (void)

Start the characteristic descriptor discovery procedure for current time.

7.13.13 TimeClient_ConnConf (tipcConnDevType connParam)

Run a profile state machine to execute all the central role procedure: connection, service discovery, characteristic discovery and peer device configuration procedure.

7.13.14 TimeClient_StateMachine (void)

Run the profile internal state machine. This function is required when the application executes the TimeClient_ConnConf().

7.13.15 TimeClient_findHandles (uint16_t uuid_service, uint16_t*start_handle, uint16_t *end_handle)

Find the device services.

7.13.16 TimeClient_ReadCurrentTimeChar (void)

Read Current Time Characteristic.

7.13.17 TimeClient_ReadLocalTimeChar (void)

Read Local Time Characteristic.

7.13.18 TimeClient_ReadNextDSTChangeTimeChar (void)

Read Next DST Change Time Characteristic.

7.13.19 TimeClient_ReadReferenceTimeInfoChar (void)

Read Reference Time Info Characteristic.

7.13.20 TimeClient_ReadServerTimeUpdateStatusChar (void)

Read Server Time Update Status Characteristic.

7.13.21 TimeClient_Set_Current_Time_Char_Notification (BOOL value)

Set Current Time Characteristic Notification.

7.13.22 TimeClient_Update_Reference_Time_On_Server (uint8_t ctlValue)

Start a gatt write without response procedure to write the time update control point characteristic on the server. The information read will be passed to the application in the form of an event.

7.13.23 TimeClient_DisplayCurrentTimeCharacteristicValue (tCurrentTime data_value)

Display the current time characteristics fields (year, data, ...).

7.13.24 TimeClient_Decode_Read_Characteristic_Value (uint8_t data_length, uint8_t *data_value)

Decode and store the read characteristic value.

7.13.25 TimeClient_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.13.26 TimeClient_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.13.27 TimeClient_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.13.28 TimeClient_ConnectionParameterUpdateReq_CB (tipcConnUpdateParamType *param)

This callback is called when the time client receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by TimeClient_ConnectionParameterUpdateRsp().

7.13.29 TimeClient_CharacOfService_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristics of service discovery procedure.

7.13.30 TimeClient_CharacDesc_CB (uint8_t status, uint8_t numCharac, uint8_t *charac)

This callback contains all the information of the characteristic descriptor of a service.

7.13.31 TimeClient_DataValueRead_CB (uint8_t status, uint16_t data_len, uint8_t *data)

This callback returns the read characteristic value of the connected peer device. Returns, also, the status of the read procedure.

7.13.32 TimeClient_PinCodeRequired_CB (void)

This function is called from the time client profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the TimeClient_SendPinCode() function to set the pin code.

7.13.33 TimeClient_EnableNotification_CB (uint8_t status)

This callback returns the status of the characteristic enable notification procedure on the connected peer device.

7.13.34 TimeClient_FullConfError_CB (uint8_t error_type, uint8_t code)

This callback is called when an error occurs during the full configuration and connection. The full configuration and connection procedure is run with the TimeClient_ConnConf() function call.

7.13.35 TimeClient_EnableNotificationIndication_CB (uint8_t status)

This callback returns the status of Characteristic notification/indication procedure.

7.13.36 TimeClient_NotificationReceived_CB (uint8_t handle, uint8_t length, uint8_t *data_value)

This callback is called when a Current Time Notification is received.

7.13.37 TimeClient_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.14 Time Server

7.14.1 TimeServer_Init (tipsInitDevType param)

Init the Central device.

7.14.2 TimeServer_DeviceDiscovery (tipsDevDiscType param)

Start the device discovery procedure.

7.14.3 TimeServer_SecuritySet (tipsSecurityType param)

Setup the device security parameters.

7.14.4 TimeServer_StartPairing (void)

Start Pairing procedure with the peer device according the device security parameters set with the TimeServer_SecuritySet().

7.14.5 TimeServer_Clear_Security_Database ()

Clear Security database.

7.14.6 TimeServer_SendPinCode (uint32_t pinCode)

Send the pin code during the pairing procedure when required from the peer device.

7.14.7 TimeServer_DeviceConnection (tipsConnDevType param)

Start the Connection procedure with the peer device.

7.14.8 TimeServer_DeviceDisconnection (void)

Start the device disconnection procedure.

7.14.9 TimeServer_ConnectionParameterUpdateRsp (uint8_t accepted, tipsConnUpdateParamType *param)

This function shall be called by the master when it receives a connection parameter update request from the peer device. If the master rejects these parameters it can send the response with the accepted parameter set to FALSE or ignore the request. This function shall be managed only if the master has already discovered all the services and characteristics. If these discovery procedure are ongoing, the master basic profile sends automatically the connection parameter response with status rejected.

7.14.10 TimeServer_Add_Services_Characteristics (void)

Add the Phone Alert Status service & related characteristics.

7.14.11 TimeServer_StateMachine (void)

Run the profile central state machine.

7.14.12 TimeServer_Update_Current_Time_Value (tCurrentTime timeValue)

It updates the current time characteristic with the timeValue specified.

7.14.13 TimeServer_Update_Local_Time_Information (tLocalTimeInformation localTimeInfo)

It updates the localTimeInfo characteristic with the value specified.

7.14.14 TimeServer_Update_Reference_Time_Information (tReferenceTimeInformation refTimeInfo)

It updates the reference time information characteristic with the value specified.

7.14.15 TimeServer_Update_Next_DST_Change (tTimeWithDST timeDST)

It updates the next DST change characteristic with the value specified.

7.14.16 Update_Reference_Time (uint8_t errCode, tCurrentTime *newTime)

It updates the reference time.

7.14.17 TimeServer_Allow_Curtime_Char_Read ()

It sends the allow read command to the controller.

7.14.18 TimeServer_Update_Control_Point_Handler (tTimeUpdateControlPoint update_control_point_value)

It handles the update control point request.

7.14.19 TimeServer_DeviceDiscovery_CB (uint8_t status, uint8_t addr_type, uint8_t *addr, uint8_t data_length, uint8_t *data, uint8_t RSSI)

This callback contains all the information of the device discovered during the central procedure.

7.14.20 TimeServer_ServicesDiscovery_CB (uint8_t status, uint8_t numServices, uint8_t *services)

This callback contains all the information of the service discovery procedure.

7.14.21 TimeServer_ConnectionStatus_CB (uint8_t connection_evt, uint8_t status)

This callback contains the status of the connection procedure.

7.14.22 TimeServer_ConnectionParameterUpdateReq_CB (tipsConnUpdateParamType *param)

This callback is called when the receives a connection parameter update request from the peripheral device connected. The response from the central device may be ignored or sent by TimeServer_ConnectionParameterUpdateRsp().

7.14.23 TimeServer_PinCodeRequired_CB (void)

This function is called from the profile when the MITM pin code is required from the peer device. Inside this function the application needs to call the TimeServer_SendPinCode() function to set the pin code.

7.14.24 TimeServer_Pairing_CB (uint16_t conn_handle, uint8_t status)

This callback is called for providing the pairing procedure status.

7.14.25 TimeServer_Notify_State_To_User_Application_CB (uint8_t event_value)

User callback which is called each time a specific event occurs.

8 List of references

Table 5. References

Name	Title
Bluetooth specifications	Specification of the Bluetooth system V4.0 and V4.1
UM1755	BlueNRG Bluetooth LE stack application command interface (ACI) User manual
UM1865	BlueNRG-MS Bluetooth LE stack application command interface (ACI) User manual
UM1686	BlueNRG Development Kits User manual
UM1870	BlueNRG-MS Development Kits User manual

Appendix A List of acronyms and abbreviations

This appendix lists the standard acronyms and abbreviations used throughout the document.

Table 6. List of acronyms

Term	Meaning
ACI	Application command interface
BLE	Bluetooth low energy
USB	Universal serial bus

9 Revision history

Table 7. Document revision history

Date	Revision	Changes
26-Aug-2014	1	Initial release.
04-Dec-2014	2	Ported profiles to simplified ACI framework (the one provided within the BlueNRG DK SW package).
25-Oct-2016	3	Document updated to include references to BlueNRG-MS device. Added *Section 3.2: Central roles*, *Chapter 7: Profiles central roles: APIs interface and callbacks* and *List of acronyms and abbreviations*. Updated *Table 5: References*.