UM1873
User manual

Getting started with the X-CUBE-BLE1 Bluetooth Low Energy software expansion for STM32Cube

Introduction

This user manual describes how to get started with the X-CUBE-BLE1.

X-CUBE-BLE1 provides a complete middleware for STM32 to build applications using ST's BlueNRG Bluetooth Low Energy device. It is easily portable across different MCU families, thanks to STM32Cube. This package contains sample applications enabling data communication via Bluetooth Low Energy.

The software provides implementation examples for STM32 Nucleo platforms equipped with the X-NUCLEO-IDB04A1 or X-NUCLEO-IDB05A1 expansion boards, featuring Bluetooth Low Energy connectivity.

The software is based on STM32Cube technology and expands STM32Cube based packages.

Contents

1 Acronyms and abbreviations 5

2 What is STM32Cube? 6

2.1 STM32Cube architecture 6

3 X-CUBE-BLE1 software, expansion for STM32Cube 9

3.1 Overview 9

3.1.1 Bluetooth Low Energy 9

3.1.2 Bluetooth operating modes 9

3.1.3 Bluetooth Low Energy software partitioning 9

3.1.4 Profiles and services 10

3.1.5 Bluetooth Low Energy state machine 12

3.2 Architecture 12

3.3 Folder structure 13

3.4 Guide for writing applications 14

3.4.1 APIs 14

3.4.2 Initialization 15

3.4.3 Security requirements 16

3.4.4 Connectable mode 16

3.4.5 Connection with central device 16

3.5 Sample application description 17

3.5.1 Time service 17

3.5.2 LED service 20

3.5.3 Testing the sample application 22

3.5.4 Adding security to sample application 30

4 System setup guide 33

4.1 Hardware description 33

4.1.1 STM32 Nucleo platform 33

4.1.2 BlueNRG expansion board 33

4.1.3 BlueNRG USB dongle 34

4.1.4 Software description 34

4.2 Hardware and software setup 35

4.2.1 Hardware setup 35

4.2.2 Software setup 35

4.2.3 System setup guide 35

5 References 37

6 Revision history 38

List of tables

Table 1: Acronyms and abbreviations 5
Table 2: Document revision history 38

List of figures

Figure 1: Firmware architecture 7
Figure 2: Bluetooth LE protocol stack 9
Figure 3: Structure of a GATT-based profile 11
Figure 4: BLE state machine 12
Figure 5: STM32 Nucleo + BlueNRG expansion board software architecture 13
Figure 6: X-CUBE-BLE1 package folder structure 14
Figure 7: Hardware components setup 22
Figure 8: Initializing the USB dongle 23
Figure 9: Scanning for devices 24
Figure 10: Create connection 25
Figure 11: Discover all supported services 26
Figure 12: Discover all supported characteristics 27
Figure 13: Reading data from time characteristic 28
Figure 14: Writing data to LED button time characteristic 29
Figure 15: Enabling notifications from server 29
Figure 16: Disconnecting the peripheral device 30
Figure 17: Setting the device I/O capabilities 31
Figure 18: Setting the pairing parameters 31
Figure 19: Pairing with the device 32
Figure 20: STM32 Nucleo board 33
Figure 21: BlueNRG expansion board connected to STM32 Nucleo board 34
Figure 22: STEVAL-IDB003V1 BlueNRG USB dongle 34

1 Acronyms and abbreviations

Table 1: Acronyms and abbreviations

Term	Description
ACI	Application controller interface
ATT	Attribute protocol
BLE	Bluetooth Low Energy
BSP	Board support package
BT	Bluetooth
GAP	Generic access profile
GATT	Generic attribute profile
GUI	Graphical user interface
HAL	Hardware abstraction layer
HCI	Host controller interface
HRS	Heart rate sensor
IDE	Integrated development environment
L2CAP	Logical link control and adaptation protocol
LED	Light emitting diode
LL	Link layer
LPM	Low power manager
MCU	Micro controller unit
PCI	Profile command interface
PHY	Physical layer
SIG	Special interest group
SM	Security manager
SPI	Serial peripheral interface
UUID	Universally unique identifier

2 What is STM32Cube?

What is STM32Cube?

STMCube™ represents an original initiative by STMicroelectronics to ease developers' life by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio. Version 1.x of STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as the STM32CubeF4 for STM32F4 series).
  - STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across the STM32 portfolio
  - A consistent set of middleware components, such as RTOS, USB, TCP/IP, graphics
  - All embedded software utilities, including a full set of examples

How does this software complement STM32Cube?

The proposed software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the BlueNRG expansion board and some middleware components for communication with other Bluetooth LE devices. BlueNRG is a very low power Bluetooth Low Energy (BLE) single-mode network processor, compliant with Bluetooth specifications core 4.0. The drivers abstract low-level details of the hardware and allow the middleware components and applications to access the BlueNRG device in a hardware-independent fashion. The software implements low power optimizations to allow system power consumption of a few micro-amps. The package includes the following sample applications that the developer can use to start experimenting with the code:

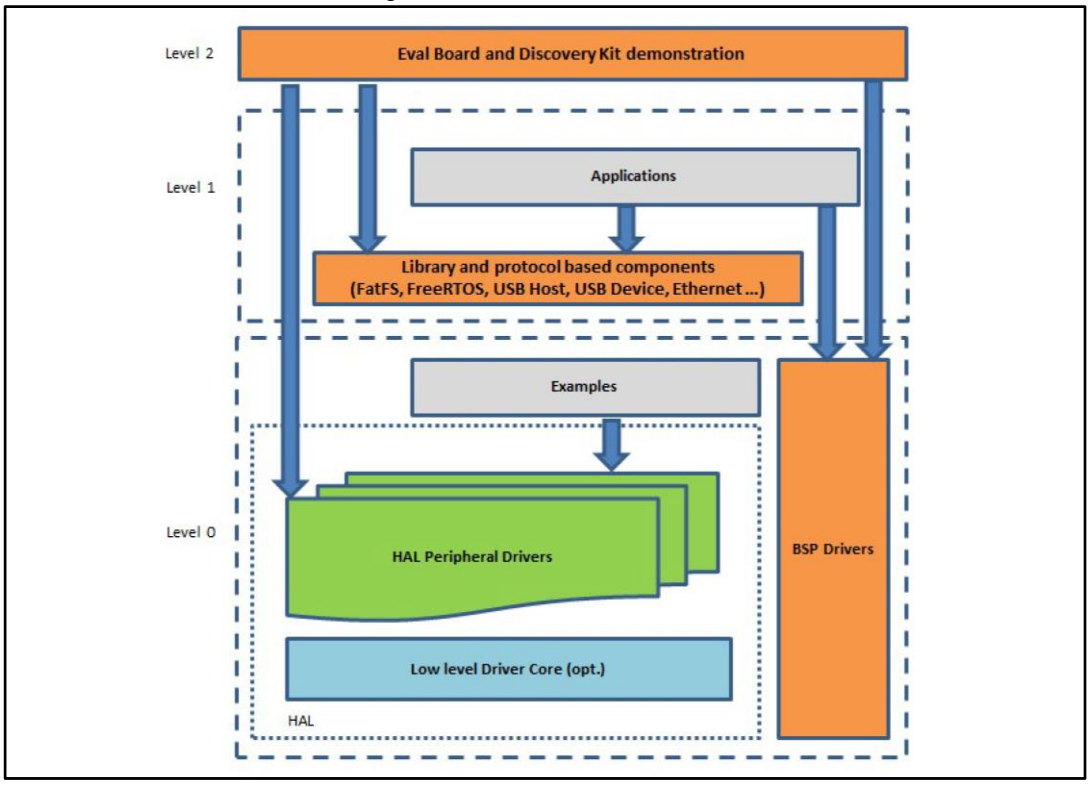
- Sample App
- Sensor Demo
- Virtual COM

Sample App shows communication between two BlueNRG devices. Sensor Demo allows the sending of simulated environmental and acceleration data to a Bluetooth Low Energy-compatible smartphone. Virtual COM allows the use of a companion PC application to send HCI commands and to update the BlueNRG firmware.

2.1 STM32Cube architecture

The STM32Cube firmware solution is based on three independent levels that freely interact with each other, as shown below:

Figure 1: Firmware architecture


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components (FatFS, FreeRTOS, USB Host, USB Device, Ethernet...)
Examples
Level 0
HAL Peripheral Drivers
Low level Driver Core (opt.)
HAL
BSP Drivers

Level 0 is divided into three sub-layers:

- The board support package (BSP) layer offers a set of board hardware APIs (audio codec, IO expander, touchscreen, SRAM driver, LCD drivers, etc.) based on modular architecture which can be rendered compatible with any hardware by simply running the low-level routines. The BSP has two parts:
  - component: the driver associated with the external device on the board (not the STM32); the component driver provides specific APIs to the BSP driver external components and can be ported to any other board.
  - BSP driver: links the component driver to a specific board and provides a set of user-friendly APIs. The naming rule of the APIs is BSP_FUNCT_Action(): ex. BSP_LED_Init(), BSP_LED_On().
- The hardware abstraction layer (HAL) provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs which render user applications unnecessary by providing ready to use processes. For example, it provides APIs for the communication peripherals (I²S, UART, etc.) for initialization and configuration, data transfer management based on polling, interrupts or DMA processes, and management of any communication errors. There are two types of HAL driver APIs:
  - generic APIs which provide common and generic functions to the entire STM32 series.
  - extension APIs which provide specific, customized functions for a particualr family or a certain part number.
- Basic peripheral usage examples: this layer includes the examples built for the STM32 peripheral using the HAL and BSP resources only.

Level 1 is divided into two sub-layers:

- Middleware components: a set of libraries covering USB host and device libraries STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction between layer components is handled directly by calling the feature APIs, while vertical interaction with the low level drivers is managed through specific callbacks and static macros implemented in the library system call interface. For example, the FatFs accesses the microSD drive or the USB mass storage class via the disk I/O driver.
- Middleware examples (or applications) for individual components as well as integration examples across several middleware components are provided.

Level 2 is a single layer providing a global, real-time and graphical demonstration based on the middleware service layer, the low-level abstraction layer and basic peripheral usage applications involving board functions.

3 X-CUBE-BLE1 software, expansion for STM32Cube

3.1 Overview

X-CUBE-BLE1 is a software package that expands the functionality provided by STM32Cube and provides the Bluetooth Low Energy connectivity.

3.1.1 Bluetooth Low Energy

Bluetooth Low Energy is a wireless personal area network technology designed and marketed by the Bluetooth SIG. It can be used for developing new innovative applications in fitness, security, healthcare, etc. using devices which run on coin cell batteries, and can remain operative for “months or years” without draining the battery.

3.1.2 Bluetooth operating modes

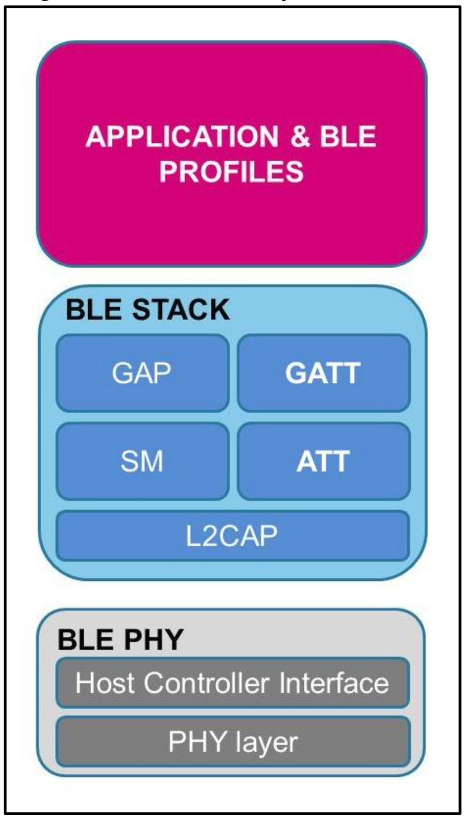
According to the Bluetooth standard specification version 4.0, Bluetooth Classic and Bluetooth Low Energy can both be supported on the same device, in which case it is called a "dual-mode" device. Dual mode devices are also called "Bluetooth smart ready".

A single-mode device is one which supports only the BLE protocol. Single mode devices are called "Bluetooth smart".

3.1.3 Bluetooth Low Energy software partitioning

The BLE protocol stack and a small description of each layer are presented below:

Figure 2: Bluetooth LE protocol stack



A typical BLE system consists of an LE controller and a host. The LE controller consists of a physical layer (PHY) including the radio, a link layer (LL) and a standard host controller interface (HCI). The host consists of an HCI and other higher protocol layers, e.g. L2CAP, SM, ATT/GATT and GAP.

The host can send HCI commands to control the LE controller. The HCI interface and the HCI commands are standardized by the Bluetooth core specification. Please refer to the official document for more information.

PHY layer insures communication with stack and data (bits) transmission over the air. BLE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 radio frequency (RF) channels with 2 MHz channel spacing.

In BLE, when a device only needs to broadcast data, it transmits the data in advertising packets through the advertising channels. Any device that transmits advertising packets is called an advertiser. Devices that only aim at receiving data through the advertising channels are called scanners. Bidirectional data communication between two devices requires them to connect to each other. BLE defines two device roles at the link layer (LL) for a created connection: the master and the slave. These are the devices that act as initiator and advertiser during the connection creation, respectively.

The host controller interface (HCI) layer provides a standardized interface to enable communication between the host and controller. In BlueNRG, this layer is implemented through the SPI hardware interface.

In BLE, the main goal of L2CAP is to multiplex the data of three higher layer protocols, ATT, SMP and link layer control signaling, on top of a link layer connection.

The SM layer is responsible for pairing and key distribution, and enables secure connection and data exchange with another device.

At the highest level of the core BLE stack, the GAP specifies device roles, modes and procedures for the discovery of devices and services, the management of connection establishment and security. In addition, GAP handles the initiation of security features. The BLE GAP defines four roles with specific requirements on the underlying controller: Broadcaster, Observer, Peripheral and Central.

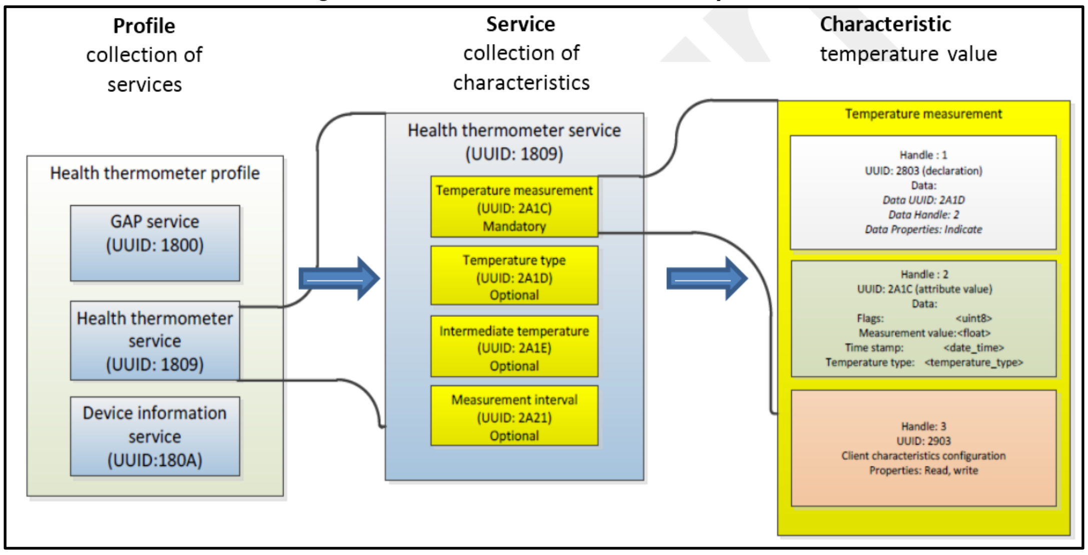
The ATT protocol allows a device to expose certain pieces of data, known as "attributes", to another device. The ATT defines the communication between two devices playing the roles of server and client, respectively, on top of a dedicated L2CAP channel. The server maintains a set of attributes. An attribute is a data structure that stores the information managed by the GATT, the protocol that operates on top of the ATT. The client or server role is determined by the GATT, and is independent of the slave or master role.

The GATT defines a framework that uses the ATT for the discovery of services, and the exchange of characteristics from one device to another. GATT specifies the structure of profiles. In BLE, all pieces of data that are being used by a profile or service are called "characteristics". A characteristic is a set of data which includes a value and properties.

3.1.4 Profiles and services

The BLE protocol stack is used by the applications through its GAP and GATT profiles. The GAP profile is used to initialize the stack and setup the connection with other devices. The GATT profile is a way of specifying the transmission - sending and receiving - of short pieces of data known as ‘attributes’ over a Bluetooth smart link. All current Low Energy application profiles are based on GATT. The GATT profile allows the creation of profiles and services within these application profiles. Here is a depiction of how the data services are setup in a typical GATT server.

Figure 3: Structure of a GATT-based profile


Profile
collection of services
Service
collection of characteristics
Characteristic
temperature value
Health thermometer profile
GAP service (UUID: 1800)
Health thermometer service (UUID: 1809)
Device information service (UUID:180A)
Health thermometer service (UUID: 1809)
Temperature measurement (UUID: 2A1C) Mandatory
Temperature type (UUID: 2A1D) Optional
Intermediate temperature (UUID: 2A1E) Optional
Measurement interval (UUID: 2A21) Optional
Temperature measurement
Handle : 1
UUID: 2803 (declaration)
Data:
Data UUID: 2A1D
Data Handle: 2
Data Properties: Indicate
Handle : 2
UUID: 2A1C (attribute value)
Data:
Flags: <uint8>
Measurement value:<float>
Time stamp: <date_time>
Temperature type: <temperature_type>
Handle: 3
UUID: 2903
Client characteristics configuration
Properties: Read, write

In this example, the profile above is created with three services:

- GAP Service, which is always mandatory to be setup
- Health thermometer service
- Device information service

Each service consists of a set of characteristics which define the service and the type of data it provides as part of the service. In the above example, the health thermometer service contains the following characteristics:

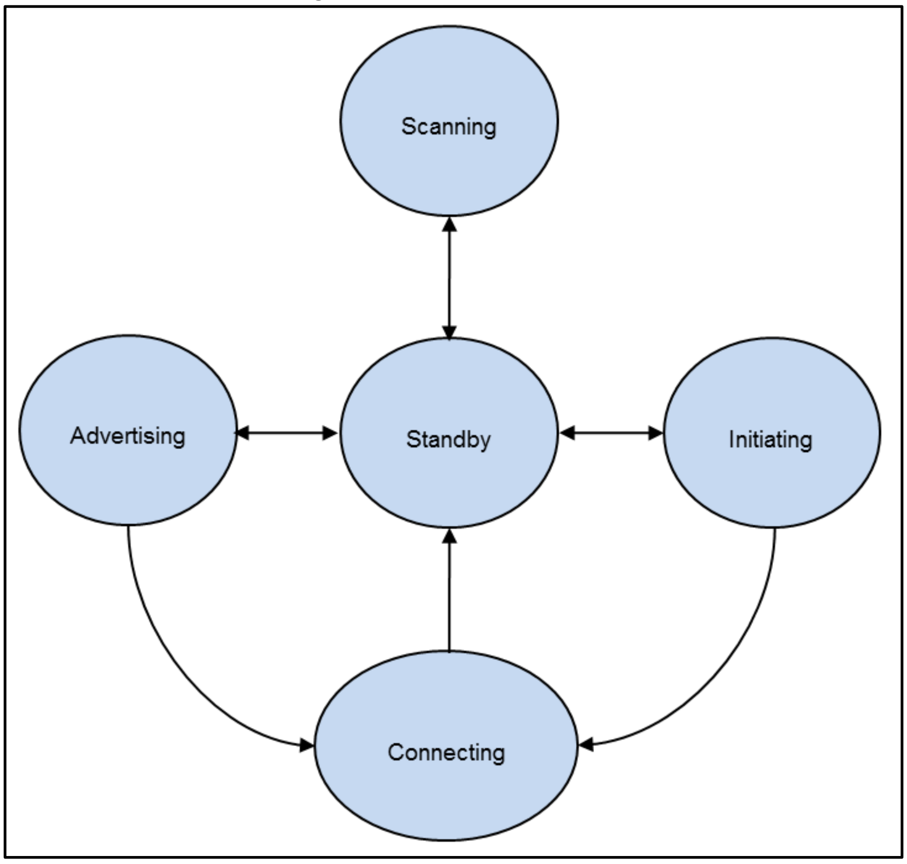
- Temperature measurement
- Temperature type
- Intermediate temperature
- Measurement interval

Each of the above characteristics details a type of data and the value of the data. The characteristics are defined by "attributes" which define the value of that characteristic.

Each characteristic has at least two attributes: the main attribute (0x2803) and a value attribute that actually contains the data. The main attribute defines the value attribute's handle and UUID which allows any client reading the attribute to know which handle to read to access the value attribute.

3.1.5 Bluetooth Low Energy state machine

Figure 4: BLE state machine



The above diagram describes the state machine during BLE operations. The following provides an explanation of each of the states:

- Standby: Does not transmit or receive packets.
- Advertising: Broadcasts advertisements in advertising channels. The device is transmitting advertising channel packets and possibly listening to and responding to responses triggered by these advertising channel packets.
- Scanning: Looks for advertisers. The device is listening for advertising channel packets from devices that are advertising.
- Initiating: The device initiates connection to the advertiser and is listening for advertising channel packets from a specific device(s) and responding to these packets to initiate a connection with another device.
- Connection: Connection has been made and the device is transmitting or receiving.
  - Initiator device will be in master role: it communicates with the device in the slave role, defines timings of transmissions.
  - Advertiser device will be in slave role: it communicates with single device in master role.

3.2 Architecture

This software is an expansion for STM32Cube, as such it fully complies with the architecture of STM32Cube and it expands it in order to enable development of applications accessing and using BlueNRG stack. Please see the previous section for an introduction to the STM32Cube architecture.

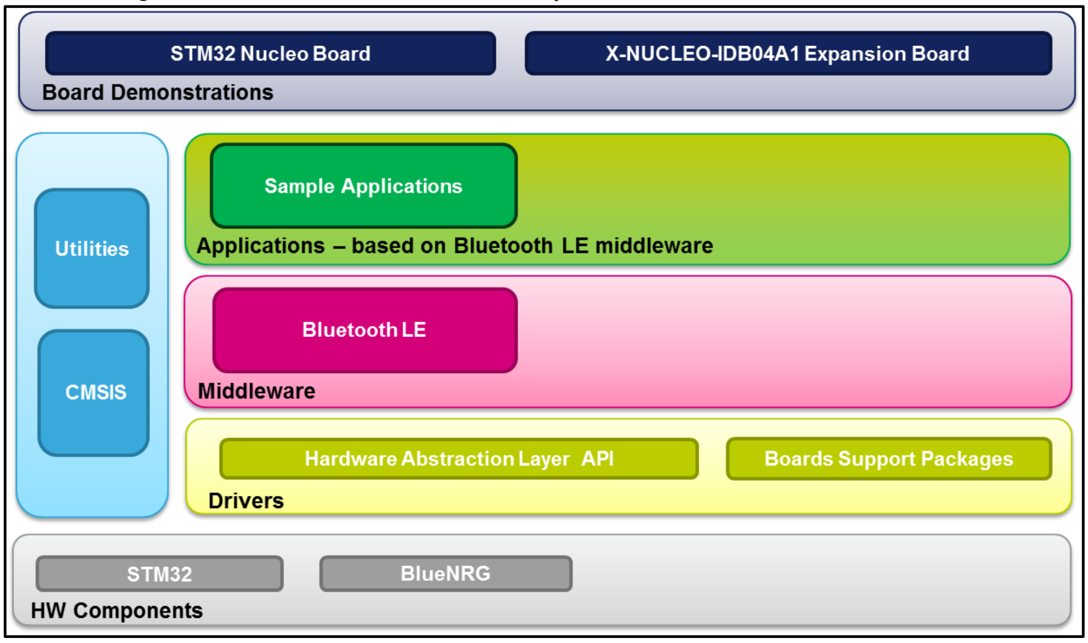
The software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the BlueNRG expansion board and some middleware components for serial communication with a PC.

The software layers used by the application software to access and use the BlueNRG expansion board are the following:

- STM32Cube HAL layer: The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers that are built upon, such as the middleware layer, to implement their functionalities without dependencies on the specific hardware configuration for a given microcontroller unit (MCU). This structure improves the library code reusability and guarantees an easy portability on other devices.
- Board support package (BSP) layer: The software package needs to support the peripherals on the STM32 Nucleo board apart from the MCU. This software is included in the board support package (BSP). This is a limited set of APIs which provides a programming interface for certain board specific peripherals, e.g. the LED, the user button, etc. This interface also helps in identifying the specific board version. For the BlueNRG expansion board, it provides provides support for Bluetooth Low Energy connectivity.

The figure below outlines the software architecture of the package:

Figure 5: STM32 Nucleo + BlueNRG expansion board software architecture

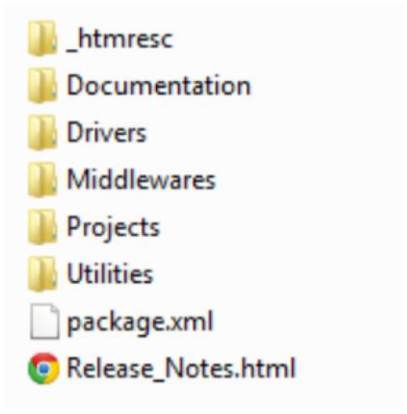


3.3 Folder structure

This section provides an overview of the package folder structure.

The figure below shows the architecture of the package.

Figure 6: X-CUBE-BLE1 package folder structure


_htmresc
Documentation
Drivers
Middlewares
Projects
Utilities
package.xml
Release_Notes.html

The following folders are included in the software package:

- The **Documentation** folder contains a compiled HTML file generated from the source code and documentating in details the software components and APIs.
- The **Drivers** folder contains the HAL drivers, the board specific drivers for each supported board or hardware platform, including the on-board components ones and the CMSIS layer which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.
- The **Middlewares** folder contains libraries and protocols related to host software and applications to interface the BlueNRG controller.
- The **Projects** folder contains a sub-folder called "Multi" including the following set of BLE applications (in the Applications folder):
  - **DMA_LowPower_App**
  - **SampleApp**
  - **SensorDemo**
  - **SensorDemo_GATTServer**
  - **Virtual_COM_Port**
- The **Utilities** folder contains a "FlashUpdaterTool" subfolder with a Java-based graphical tool allowing the user to upgrade the firmware of the BlueNRG expansion board from version 6.3 to version 6.4. Notice that this is a preliminary and required step to test the Applications listed above. The relevant readme file provides more in depth details about the tool and its usage.

All the applications in the **Projects** folder are provided for the NUCLEO-L053R8 and NUCLEO-F401RE platforms with three development environments (IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), Atollic TrueSTUDIO® for ARM).

3.4 Guide for writing applications

This section describes how to write a BLE application based on STM32 Nucleo board equipped with BlueNRG expansion board, and add GATT services and characteristics to it. Please refer to UM1686 "BlueNRG development kits" for seeing more details about the ACI API referenced in this section.

3.4.1 APIs

This section describes generic initialization and setup while writing BLE applications.

In this setup, the STM32 Nucleo board acts as GATT server (and as a peripheral device) and the PC as a GATT client (and as a central device).

Detailed technical information about the APIs available to the user can be found in a compiled HTML file located inside the "Documentation" folder of the software package where all the functions and parameters are fully described.

3.4.2 Initialization

Every application must to perform the basic initialization steps in order to configure and set up the STM32 Nucleo with the BlueNRG expansion board hardware and the software stack for correct operation. This section describes the initialization steps required.

3.4.2.1 Initializing STM32 Cube HAL

The STM32Cube HAL library must be initialized so that the necessary hardware components are correctly configured.

- HAL_Init();

This API initializes the HAL library. It configures Flash prefetch, Flash preread and Buffer cache. It also configures the time base source, vectored interrupt controller and low-level hardware.

3.4.2.2 Initializing Nucleo board peripherals

Some of the Nucleo on-board peripherals and hardware need to be configured before using them (if they are used). The functions to do this are:

- BSP_LED_Init(Led_TypeDef Led);

This API configures the LED on the Nucleo.

- BSP_PB_Init(Button_TypeDef Button, ButtonMode_TypeDef Button_Mode);

This API configures the user button in GPIO mode or in external interrupt (EXTI) mode.

- BSP_JOY_Init();

This API configures the joystick if the board is equipped with one.

3.4.2.3 Initializing BlueNRG HAL and HCI

The BlueNRG HAL provides the API and the functionality for performing operations related to the BlueNRG expansion board. This layer must be initialized so that STM32 CUBE HAL is configured properly for use with the BlueNRG expansion board.

- BNRG_SPI_Init();

This API is used to initialize the SPI communication with the BlueNRG expansion board.

- HCI_Init();

This API initializes the host controller interface (HCI).

- BlueNRG_RST();

This API resets the BlueNRG expansion board.

3.4.2.4 Initialization and services characteristics

BlueNRG's stack must be correctly initialized before establishing a connection with another BLE device. This is done with the following two commands.

- aci_gap_init(uint8_t role, uint16_t* service_handle, uint16_t* dev_name_char_handle, uint16_t* appearance_char_handle);

This API initializes BLE device for a particular role (peripheral, broadcaster, central device etc.). The role is passed as first parameter to this API.

- aci_gatt_add_serv(UUID_TYPE_128, service_uuid, PRIMARY_SERVICE, 7, &servHandle);

This API adds a service on the GATT server device. Here service_uuid is the 128-bit private service UUID allocated for the service (primary service). This API returns the service handle in servHandle.

3.4.3 Security requirements

The BlueNRG stack exposes an API that the GATT client application can use to specify its security requirements. If a characteristic has security restrictions, a pairing procedure must be initiated by the central device in order to access that characteristic. In the provided BLE SensorDemo a fixed pin (123456) is used as follow:

- aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED, OOB_AUTH_DATA_ABSENT, NULL, 7, 16, USE_FIXED_PIN_FOR_PAIRING, 123456, BONDING);

3.4.4 Connectable mode

On the GATT server device the following GAP ACI command is used to enter on general discoverable mode:

- aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR, NO_WHITE_LIST_USE,8, local_name, 0, NULL, 0, 0);

3.4.5 Connection with central device

Once the device used as GATT server is put in a discoverable mode, it can be seen by the GATT client role device in order to create a BLE connection.

On GATT Client device the following GAP ACI command is used to connect with the GATT server device in advertising mode:

- aci_gap_create_connection(0x4000, 0x4000, PUBLIC_ADDR, bdaddr, PUBLIC_ADDR, 9, 9, 0, 60, 1000 , 1000), where bdaddr is the peer address of the GATT Client role device.

Once the two devices are connected the BLE communication will work as follows:

- On GATT server role device the following API should be invoked for updating characteristic value:
  - aci_gatt_update_char_value(chatServHandle, TXCharHandle, 0, len, (tHalUint8 *)data)

where data contains the value by which the attribute pointed to by the characteristic handle TxCharHandle contained within the service chatServHandle, will be updated.

- On GATT client device, following API should be invoked for writing to a characteristic handle:
  - aci_gatt_write_without_response(connection_handle, RX_HANDLE+1, len, (tHalUint8 *)data)

where data is the value of the attribute pointed to by the attribute handle RX_HANDLE contained in the connection handle connection_handle. connection_handle is the handle returned on connection creation as parameter of the EVT_LE_CONN_COMPLETE event.

3.5 Sample application description

This section describes various services and characteristics in the SensorDemo_GATTServer application.

The project files for the SensorDemo application can be found here:

$BASE_DIR\Projects\Multi\Applications\SensorDemo_GATTServer

In the SensorDemo_GATTServer application, the STM32 Nucleo device creates two services:

- Accelerometer service with the free-fall characteristic data and the directional acceleration value characteristic in three directions (x, y, and z axis)
- Environmental Service with the following characteristics:
  - Temperature data characteristic
  - Pressure data characteristic
  - Humidity data characteristic

Please note that there is no "real" environment sensor and accelerometer on the STM32 Nucleo board and the data being generated is "simulated" data.

The application creates services and characteristics using ACI APIs described in *Section 3.4.1: "APIs"* and then waits for a client (central device) to connect to it. It advertises its services and characteristics to the listening client devices while waiting for a connection to be made. After the connection is created by the central device, data is periodically updated.

3.5.1 Time service

Time service is the new service that will be added to the SensorDemo application. Time service has the following two characteristics:

- Seconds characteristic: exposes the number of seconds passed since system boot. This is a read only characteristic.
- Minutes characteristic: exposes the number of minutes passed since system boot. This characteristic can be read by GATT server, and a "notify" event is generated for this characteristic at one minute intervals.

3.5.1.1 Adding Time service

The following piece of code in sensor_service.c is adding "Time service" and its corresponding characteristics to the SensorDemo application. As explained in *Section 3.4.2.4: "Initialization and services characteristics"* of this document, the aci_gatt_add_serv() API is used to add a service to the application, and the aci_gatt_add_char()is used to add the characteristics. Please refer to UM1686: BlueNRG development kits for details about these APIs. Please note that while adding "seconds characteristic", it is marked as a characteristic supporting read operation by using CHAR_PROP_READ argument. Similarly "minute characteristic" is marked as readable and notifiable by using the CHAR_PROP_NOTIFY|CHAR_PROP_READ argument.

```
/**
 * @brief  Add an time service using a vendor specific profile
 * @param  None
 * @retval Status
 */
tBleStatus Add_Time_Service(void)
{
  tBleStatus ret;
  uint8_t uuid[16];

  /* copy "Timer service UUID" defined above to 'uuid' local variable */
  COPY_TIME_SERVICE_UUID(uuid);

  /*
   * now add "Time service" to GATT server, service handle is returned
   * via 'timeServHandle' parameter of aci_gatt_add_serv() API.
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
  */
  ret = aci_gatt_add_serv(UUID_TYPE_128,  uuid, PRIMARY_SERVICE, 7,
                          &timeServHandle);
  if (ret != BLE_STATUS_SUCCESS) goto fail;

  /*
   * now add "Seconds characteristic" to Time service, characteristic handle
   * is returned via 'secondsCharHandle' parameter of aci_gatt_add_char() API.
   * This characteristic is read only, as specified by CHAR_PROP_READ parameter.
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
  */
  COPY_TIME_UUID(uuid);
  ret =  aci_gatt_add_char(timeServHandle, UUID_TYPE_128, uuid, 4,
                           CHAR_PROP_READ, ATTR_PERMISSION_NONE, 0,
                           16, 0, &secondsCharHandle);
                           if (ret != BLE_STATUS_SUCCESS) goto fail;

  COPY_MINUTE_UUID(uuid);
  /*
   * Add "Minutes characteristic" to "Time service".
   * This characteristic is readable as well as notifiable only, as specified
   * by CHAR_PROP_NOTIFY|CHAR_PROP_READ parameter below.
   */
  ret =  aci_gatt_add_char(timeServHandle, UUID_TYPE_128, uuid, 4,
                           CHAR_PROP_NOTIFY|CHAR_PROP_READ, ATTR_PERMISSION_NONE, 0,
                           16, 0, &minuteCharHandle);
  if (ret != BLE_STATUS_SUCCESS) goto fail;

  PRINTF("Service TIME added. Handle 0x%04X, TIME Charac handle:
0x%04X\n",timeServHandle, secondsCharHandle);
  return BLE_STATUS_SUCCESS;

  /* return BLE_STATUS_ERROR if we reach this tag */
fail:
  PRINTF("Error while adding Time service.\n");
  return BLE_STATUS_ERROR ;
}
```

Finally, Add_Time_Service()function should be called from main() function defined in main.c. The following code performs this task.

```
    /* instantiate timer service with 2 characteristics:-
   * 1. seconds characteristic: Readable only
   * 2. Minutes characteristics: Readable and Notifiable
   */
  ret = Add_Time_Service();

  if(ret == BLE_STATUS_SUCCESS)
    PRINTF("Time service added successfully.\n");
  else
    PRINTF("Error while adding Time service.\n");
```

3.5.1.2 Update and notify characteristic value

Time service has "seconds characteristic" as a "readable" characteristic. Support for updating this characteristic must be provided in this application. Seconds_Update() function, defined below, performs this task.

```
/**
 * @brief  Update seconds characteristic value of Time service
 * @param  AxesRaw_t structure containing acceleration value in mg
 * @retval Status
 */
tBleStatus Seconds_Update(void)
{
  tHalUint32 val;
  tBleStatus ret;

  /* Obtain system tick value in milliseconds, and convert it to seconds. */
  val = HAL_GetTick();
  val = val/1000;

  /* create a time[] array to pass as last argument of aci_gatt_update_char_value()
API*/
  const tHalUint8 time[4] = {(val >> 24)&0xFF, (val >> 16)&0xFF, (val >> 8)&0xFF,
(val)&0xFF};
/*
   * Update value of "Seconds characteristic" using aci_gatt_update_char_value() API
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
   */
  ret = aci_gatt_update_char_value(timeServHandle, secondsCharHandle, 0, 4,
                                   time);

  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating TIME characteristic.\n") ;
    return BLE_STATUS_ERROR ;
  }
  return BLE_STATUS_SUCCESS;
}
```

Similarly, the value of "minutes characteristics" should also be updated. The Minutes_Notify() function as described below performs this operation. This function updates the value of the "minutes characteristic" exactly once in a one minute interval.

```
    /**
 * @brief  Send a notification for a minute characteristic of time service
 * @param  None
 * @retval Status
 */
tBleStatus Minutes_Notify(void)
{
  tHalUint32 val;
  tHalUint32 minuteValue;
  tBleStatus ret;

  /* Obtain system tick value in milliseconds */
  val = HAL_GetTick();
/* update "Minutes characteristic" value iff it has changed w.r.t. previous
   * "minute" value.
   */
  if((minuteValue=val/(60*1000))!=previousMinuteValue) {
    /* memmorize this "minute" value for future usage */
    previousMinuteValue = minuteValue;

    /* create a time[] array to pass as last argument of
aci_gatt_update_char_value() API*/
    const tHalUint8 time[4] = {(minuteValue >> 24)&0xFF, (minuteValue >> 16)&0xFF,
(minuteValue >> 8)&0xFF, (minuteValue)&0xFF};

  /*
   * Update value of "Minutes characteristic" using aci_gatt_update_char_value() API
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
```

```
    */
     ret = aci_gatt_update_char_value(timeServHandle, minuteCharHandle, 0, 4,
                                      time);
   if (ret != BLE_STATUS_SUCCESS){
       PRINTF("Error while updating TIME characteristic.\n") ;
       return BLE_STATUS_ERROR ;
     }
   }
   return BLE_STATUS_SUCCESS;
}
```

Finally, Seconds_Update() and Minutes_Notify() must be invoked from main() function. Update_Time_Characteristics() described below performs this task.

```
    /**
 * @brief  Updates "Seconds and Minutes characteristics" values
 * @param  None
 * @retval None
 */
void Update_Time_Characteristics() {
  /* update "seconds and minutes characteristics" of time service  */
  Seconds_Update();
  Minutes_Notify();
}
```

Please note that main() function invokes Update_Time_Characteristics(), which in turn invokes Seconds_Update() and Minutes_Notify().

3.5.2 LED service

LED service can be used to control state of LED2 present on STM32 Nucleo board. This service has a writable "LED button characteristic", which controls the state of the LED2. When the GATT client application modifies value of this characteristic, LED2 is toggled.

3.5.2.1 Adding GATT service and characteristics

The following code in sensor_service.c adds the LED service and its corresponding "LED button" characteristic to SensorDemo application.

```
    /*
 * @brief  Add LED button service using a vendor specific profile
 * @param  None
 * @retval Status
 */

tBleStatus Add_LED_Service(void)
{
  tBleStatus ret;
  uint8_t uuid[16];

  /* copy "LED service UUID" defined above to 'uuid' local variable */
  COPY_LED_SERVICE_UUID(uuid);
  /*
   * now add "LED service" to GATT server, service handle is returned
   * via 'ledServHandle' parameter of aci_gatt_add_serv() API.
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
  */
  ret = aci_gatt_add_serv(UUID_TYPE_128,  uuid, PRIMARY_SERVICE, 7,
                          &ledServHandle);
  if (ret != BLE_STATUS_SUCCESS) goto fail;
  /* copy "LED button characteristic UUID" defined above to 'uuid' local variable */
  COPY_LED_UUID(uuid);
  /*
```

```
   * now add "LED button characteristic" to LED service, characteristic handle
   * is returned via 'ledButtonCharHandle' parameter of aci_gatt_add_char() API.
   * This characteristic is writable, as specified by 'CHAR_PROP_WRITE' parameter.
   * Please refer to 'BlueNRG Application Command Interface.pdf' for detailed
   * API description
  */
  ret =  aci_gatt_add_char(ledServHandle, UUID_TYPE_128, uuid, 4,
                           CHAR_PROP_WRITE | CHAR_PROP_WRITE_WITHOUT_RESP,
ATTR_PERMISSION_NONE, GATT_SERVER_ATTR_WRITE,
                           16, 1, &ledButtonCharHandle);

  if (ret != BLE_STATUS_SUCCESS) goto fail;
PRINTF("Service LED BUTTON added. Handle 0x%04X, LED button Charac handle:
0x%04X\n",ledServHandle, ledButtonCharHandle);
  return BLE_STATUS_SUCCESS;

fail:
  PRINTF("Error while adding LED service.\n");
  return BLE_STATUS_ERROR ;
}
```

3.5.2.2 Obtaining characteristics value

When an ACI event is detected by BlueNRG BLE stack, it invokes HCI_Event_CB()function. In HCI_Event_CB() we can analyze value of the received event packet and take suitable action. HCI_Event_CB() function is described below:

```
    /**
 * @brief  This function is called whenever there is an ACI event to be
processed.
 * @note   Inside this function each event must be identified and correctly
 *         parsed.
* @param  pckt  Pointer to the ACI packet
* @retval None
*/
void HCI_Event_CB(void *pckt)
{
  hci_uart_pckt *hci_pckt = pckt;
  /* obtain event packet */
  hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;

  if(hci_pckt->type != HCI_EVENT_PKT)
    return;
switch(event_pckt->evt){
    .
    .
    .
  case EVT_VENDOR:
    {
      evt_blue_aci *blue_evt = (void*)event_pckt->data;
      switch(blue_evt->ecode){

       case EVT_BLUE_GATT_ATTRIBUTE_MODIFIED:
        {
          /* this callback is invoked when a GATT attribute is modified
             extract callback data and pass to suitable handler function */
          evt_gatt_attr_modified *evt = (evt_gatt_attr_modified*)blue_evt->data;

          Attribute_Modified_CB(evt->attr_handle, evt->data_length, evt->att_data);
        }
        break;
    .
    .
      }
    }
    break;
  }
}
```

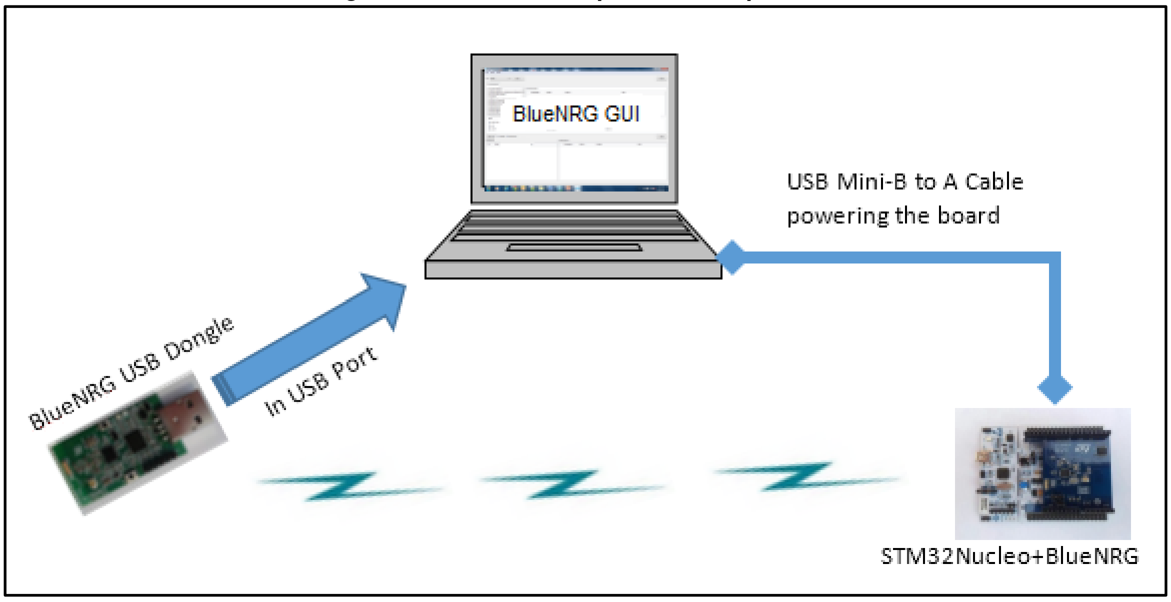
Attribute_Modified_CB() performs the event handling for LED service. It toggles the LED present on STM32 Nucleo board when the value of "LED button characteristic" is modified by the GATT client. Attribute_Modified_CB() is described below:

```
    /**
 * @brief  This function is called attribute value corresponding to
 *          ledButtonCharHandle characteristic gets modified
 * @param  handle : handle of the attribute
 * @param  data_length : size of the modified attribute data
 * @param  att_data : pointer to the modified attribute data
 * @retval None
 */
void Attribute_Modified_CB(tHalUint16 handle, tHalUint8 data_length, tHalUint8
*att_data)
{
  /* If GATT client has modified 'LED button characteristic' value, toggle LED2 */
  if(handle == ledButtonCharHandle + 1){
      BSP_LED_Toggle(LED2);
  }
}
```

3.5.3 Testing the sample application

In this section the BlueNRG GUI will be used for testing the SensorDemo application developed in the previous section. Please download the BlueNRG GUI installer provided in STSW-BlueNRG-DK. Detailed instructions regarding its use can be found in UM1686: BlueNRG development kits. For testing purposes, hardware components described in *Section 4.1.2: "BlueNRG expansion board"* are needed. The following diagram shows the interconnections among these components.

Figure 7: Hardware components setup



In subsequent sections, the Nucleo board equipped with BlueNRG expansion board will be referred to as "peripheral device" and the USB dongle as the "central device" throughout the document.

3.5.3.1 Testing SensorDemo_GATTServer application using BlueNRG GUI

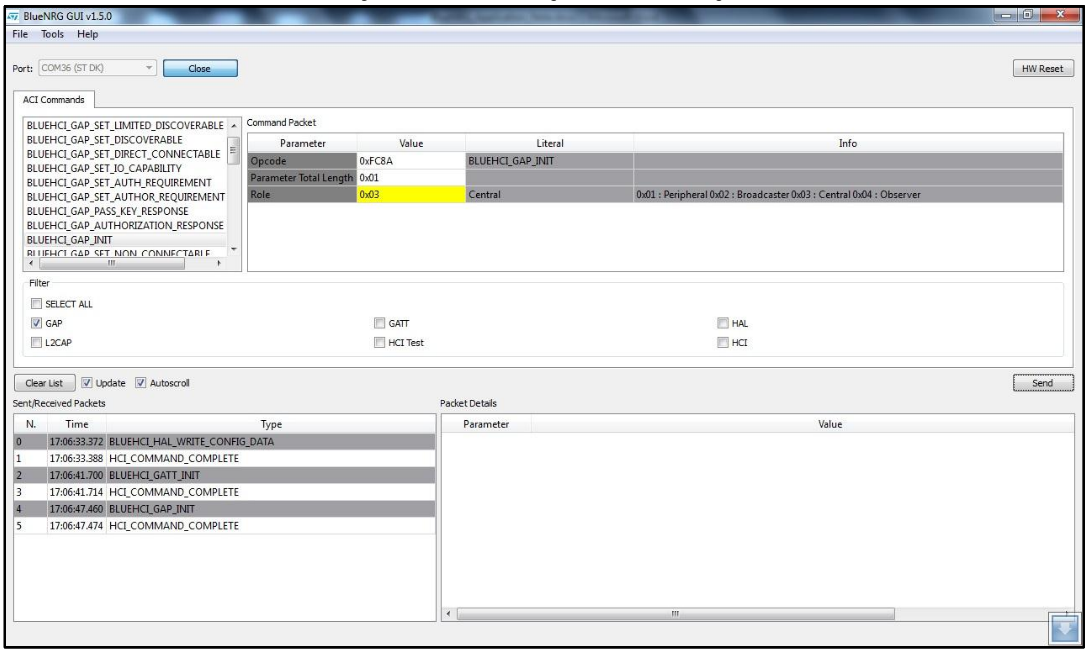
This section describes how BlueNRG GUI can be used to initialize and configure BlueNRG USB dongle properly so that it can be used to test the BLE application running on STM32Nucleo board equipped with a BlueNRG expansion board. In this example, the PC connected with BlueNRG USB dongle will be configured as "GAP central device", and the STM32Nucleo board equipped with a BlueNRG expansion board is "GAP peripheral device". Once the BlueNRG USB dongle is configured correctly, it can be used to scan remote devices and send ACI commands described in UM1686 "BlueNRG development kits". Various useful operations are described below.

3.5.3.2 Initializing the USB dongle

The BlueNRG USB dongle must be initialized so that it can communicate with the "GAP peripheral device". The following commands are used for this initialization:

1. BLUEHCI_HAL_WRITE_CONFIG_DATA
2. BLUEHCI_GATT_INIT
3. BLUEHCI_GAP_INIT

Figure 8: Initializing the USB dongle



Port: COM36 (ST DK)

BLUEHCI_GAP_SET_LIMITED_DISCOVERABLE
BLUEHCI_GAP_SET_DISCOVERABLE
BLUEHCI_GAP_SET_DIRECT_CONNECTABLE
BLUEHCI_GAP_SET_IO_CAPABILITY
BLUEHCI_GAP_SET_AUTH_REQUIREMENT
BLUEHCI_GAP_SET_AUTHOR_REQUIREMENT
BLUEHCI_GAP_PASS_KEY_RESPONSE
BLUEHCI_GAP_AUTHORIZATION_RESPONSE
BLUEHCI_GAP_INIT
BLUEHCI_GAP_SET_NON_CONNECTABLE

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC8A	BLUEHCI_GAP_INIT	
Parameter Total Length	0x01		
Role	0x03	Central	0x01 : Peripheral 0x02 : Broadcaster 0x03 : Central 0x04 : Observer

☐ SELECT ALL
☑ GAP ☐ GATT ☐ HAL
☐ L2CAP ☐ HCI Test ☐ HCI

☑ Update ☑ Autoscroll

Sent/Received Packets

N.	Time	Type
0	17:06:33.372	BLUEHCI_HAL_WRITE_CONFIG_DATA
1	17:06:33.388	HCI_COMMAND_COMPLETE
2	17:06:41.700	BLUEHCI_GATT_INIT
3	17:06:41.714	HCI_COMMAND_COMPLETE
4	17:06:47.460	BLUEHCI_GAP_INIT
5	17:06:47.474	HCI_COMMAND_COMPLETE

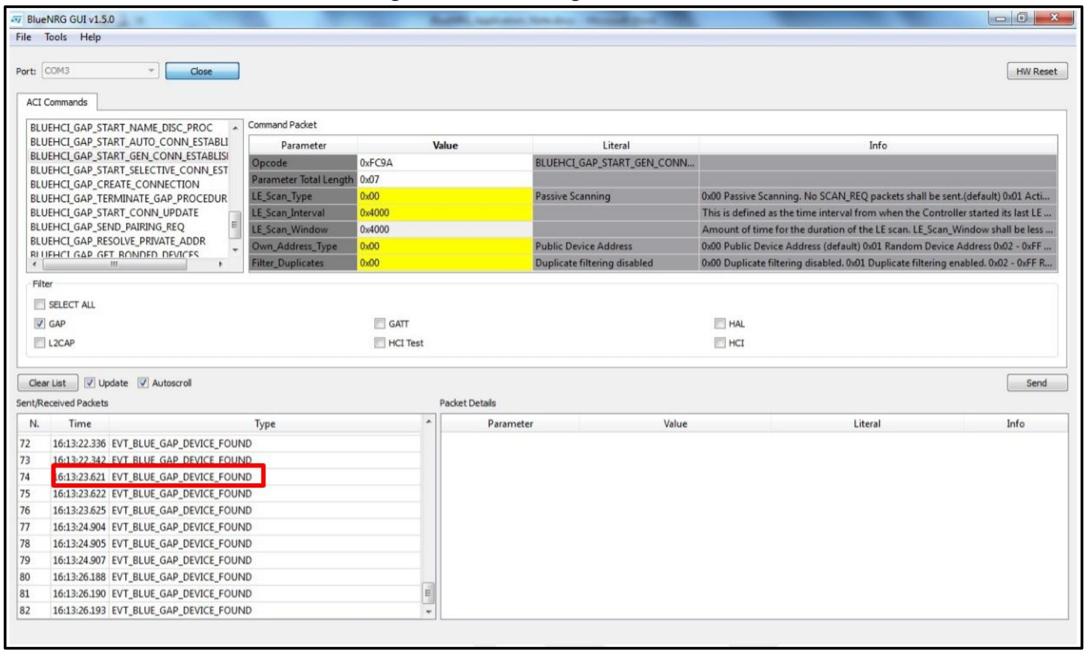
Packet Details

Parameter	Value

3.5.3.3 Scanning for BLE peripheral device

The command BLUEHCI_GAP_START_GEN_DISC_PROC discovers the "GAP peripheral device" and the following outcome is generated in the GUI window. EVT_BLUE_GAP_DEVICE_FOUND confirms that the device has been discovered by the BlueNRG dongle.

Figure 9: Scanning for devices



BLUEHCI_GAP_START_NAME_DISC_PROC
BLUEHCI_GAP_START_AUTO_CONN_ESTABLI
BLUEHCI_GAP_START_GEN_CONN_ESTABLISI
BLUEHCI_GAP_START_SELECTIVE_CONN_EST
BLUEHCI_GAP_CREATE_CONNECTION
BLUEHCI_GAP_TERMINATE_GAP_PROCEDUR
BLUEHCI_GAP_START_CONN_UPDATE
BLUEHCI_GAP_SEND_PAIRING_REQ
BLUEHCI_GAP_RESOLVE_PRIVATE_ADDR
BLUEHCI_GAP_GET_BONDED_DEVICES

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC9A	BLUEHCI_GAP_START_GEN_CONN...	
Parameter Total Length	0x07		
LE_Scan_Type	0x00	Passive Scanning	0x00 Passive Scanning. No SCAN_REQ packets shall be sent.(default) 0x01 Acti...
LE_Scan_Interval	0x4000		This is defined as the time interval from when the Controller started its last LE ...
LE_Scan_Window	0x4000		Amount of time for the duration of the LE scan. LE_Scan_Window shall be less ...
Own_Address_Type	0x00	Public Device Address	0x00 Public Device Address (default) 0x01 Random Device Address 0x02 - 0xFF ...
Filter_Duplicates	0x00	Duplicate filtering disabled	0x00 Duplicate filtering disabled. 0x01 Duplicate filtering enabled. 0x02 - 0xFF R...

Sent/Received Packets

N.	Time	Type
72	16:13:22.336	EVT_BLUE_GAP_DEVICE_FOUND
73	16:13:22.342	EVT_BLUE_GAP_DEVICE_FOUND
74	16:13:23.621	EVT_BLUE_GAP_DEVICE_FOUND
75	16:13:23.622	EVT_BLUE_GAP_DEVICE_FOUND
76	16:13:23.625	EVT_BLUE_GAP_DEVICE_FOUND
77	16:13:24.904	EVT_BLUE_GAP_DEVICE_FOUND
78	16:13:24.905	EVT_BLUE_GAP_DEVICE_FOUND
79	16:13:24.907	EVT_BLUE_GAP_DEVICE_FOUND
80	16:13:26.188	EVT_BLUE_GAP_DEVICE_FOUND
81	16:13:26.190	EVT_BLUE_GAP_DEVICE_FOUND
82	16:13:26.193	EVT_BLUE_GAP_DEVICE_FOUND

3.5.3.4 Connecting to BLE peripheral device

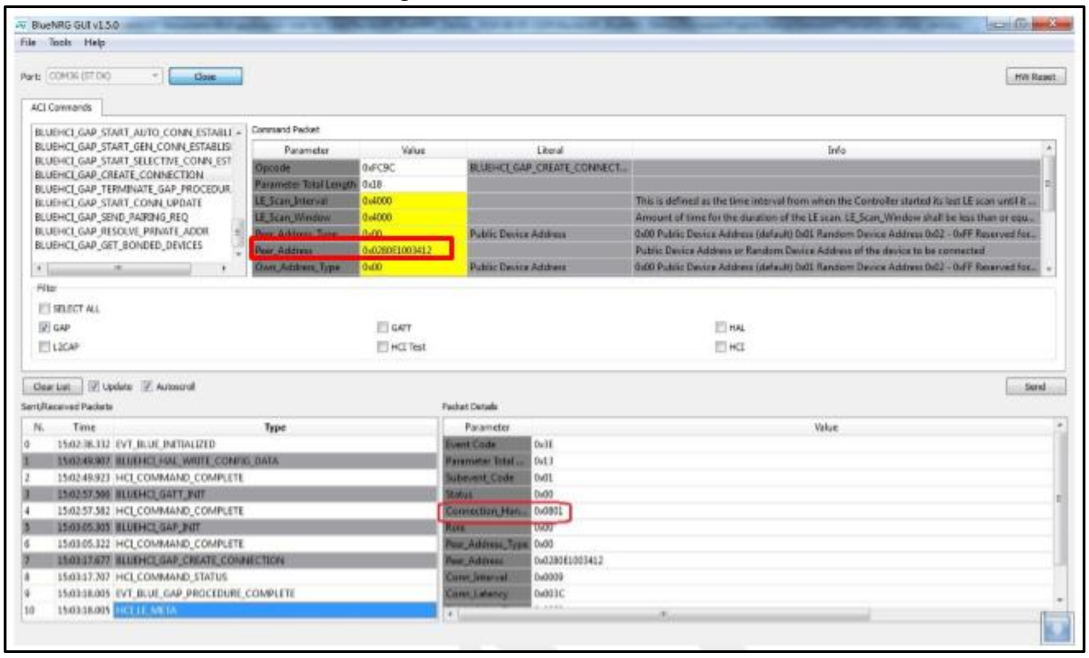
To connect "GAP peripheral device" with the dongle, we need to issue the command BLUEHCI_GAP_CREATE_CONNECTION from the BlueNRG GUI.

The peer address required for the connection is the address of the server as mentioned in the source code:

```
tHalUint8 SERVER_BDADDR[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
```

This command will make the connection and return the connection handle. This connection handle would be required for subsequent commands to retrieve services and characteristics.

Figure 10: Create connection



BLUEHCI_GAP_START_AUTO_CONN_ESTABLI
BLUEHCI_GAP_START_GEN_CONN_ESTABLIS
BLUEHCI_GAP_START_SELECTIVE_CONN_EST
BLUEHCI_GAP_CREATE_CONNECTION
BLUEHCI_GAP_TERMINATE_GAP_PROCEDUR
BLUEHCI_GAP_START_CONN_UPDATE
BLUEHCI_GAP_SEND_PAIRING_REQ
BLUEHCI_GAP_RESOLVE_PRIVATE_ADDR
BLUEHCI_GAP_GET_BONDED_DEVICES

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC9C	BLUEHCI_GAP_CREATE_CONNECT...	
Parameter Total Length	0x18		
LE_Scan_Interval	0x4000		This is defined as the time interval from when the Controller started its last LE scan until it ...
LE_Scan_Window	0x4000		Amount of time for the duration of the LE scan. LE_Scan_Window shall be less than or equ...
Peer_Address_Type	0x00	Public Device Address	0x00 Public Device Address (default) 0x01 Random Device Address 0x02 - 0xFF Reserved for...
Peer_Address	0x0280E1003412		Public Device Address or Random Device Address of the device to be connected
Own_Address_Type	0x00	Public Device Address	0x00 Public Device Address (default) 0x01 Random Device Address 0x02 - 0xFF Reserved for...

Filter: SELECT ALL [✓ GAP] GATT HAL L2CAP HCI Test HCI

[✓ Update] [✓ Autoscroll]

Sent/Received Packets

N.	Time	Type
0	15:02:38.332	EVT_BLUE_INITIALIZED
1	15:02:49.907	BLUEHCI_HAL_WRITE_CONFIG_DATA
2	15:02:49.923	HCI_COMMAND_COMPLETE
3	15:02:57.566	BLUEHCI_GATT_INIT
4	15:02:57.582	HCI_COMMAND_COMPLETE
5	15:03:05.305	BLUEHCI_GAP_INIT
6	15:03:05.322	HCI_COMMAND_COMPLETE
7	15:03:17.677	BLUEHCI_GAP_CREATE_CONNECTION
8	15:03:17.707	HCI_COMMAND_STATUS
9	15:03:18.005	EVT_BLUE_GAP_PROCEDURE_COMPLETE
10	15:03:18.005	HCI_LE_META

Packet Details

Parameter	Value
Event Code	0x3E
Parameter Total ...	0x13
Subevent_Code	0x01
Status	0x00
Connection_Han...	0x0801
Role	0x00
Peer_Address_Type	0x00
Peer_Address	0x0280E1003412
Conn_Interval	0x0009
Conn_Latency	0x003C

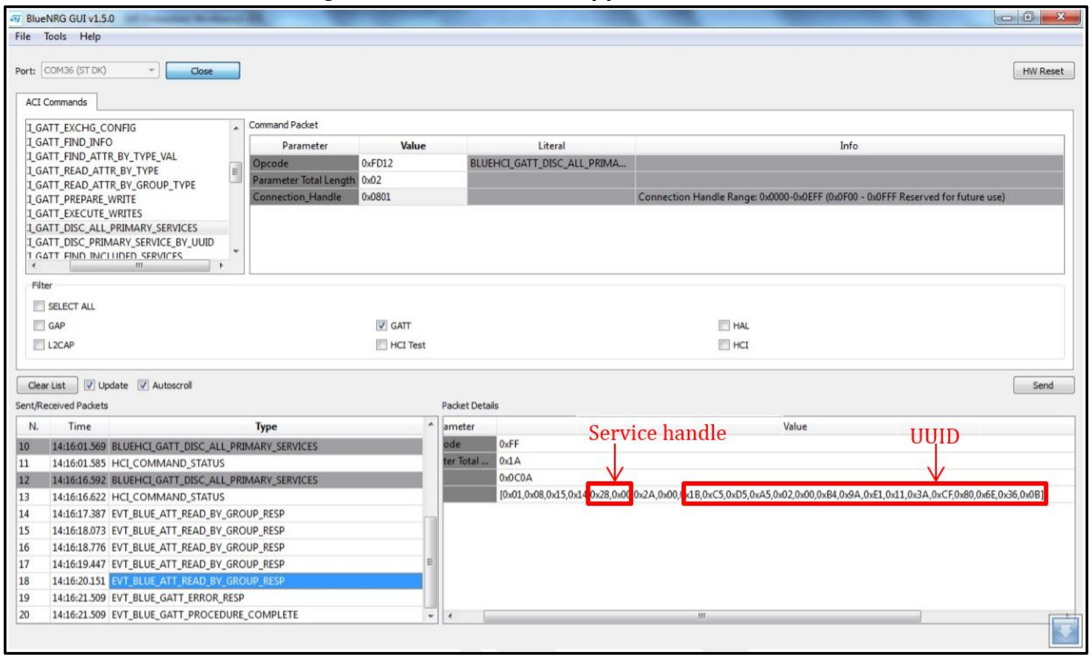
The connection handle can be determined from the HCI_LE_META response of the server. It is indicated in the figure above.

3.5.3.5 Get services supported by BLE peripheral device

The server device supports a number of GATT services and the BlueNRG GUI can obtain this information by issuing the command BLUEHCI_GATT_DISC_ALL_PRIMARY_SERVICES. Once the command is issued the server responds with EVT_BLUE_ATT_READ_BY_GROUP_RESP for each service supported by the Server.

Each response includes the connection handle, the length of the response, the data length, the handle-value pair and the UUID of the service. They are indicated in the figure below.

Figure 11: Discover all supported services



ACI Commands list:

```
I_GATT_EXCHG_CONFIG
I_GATT_FIND_INFO
I_GATT_FIND_ATTR_BY_TYPE_VAL
I_GATT_READ_ATTR_BY_TYPE
I_GATT_READ_ATTR_BY_GROUP_TYPE
I_GATT_PREPARE_WRITE
I_GATT_EXECUTE_WRITES
I_GATT_DISC_ALL_PRIMARY_SERVICES
I_GATT_DISC_PRIMARY_SERVICE_BY_UUID
I_GATT_FIND_INCLUDED_SERVICES
```

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFD12	BLUEHCI_GATT_DISC_ALL_PRIMA...	
Parameter Total Length	0x02		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Sent/Received Packets

N.	Time	Type
10	14:16:01.569	BLUEHCI_GATT_DISC_ALL_PRIMARY_SERVICES
11	14:16:01.585	HCI_COMMAND_STATUS
12	14:16:16.592	BLUEHCI_GATT_DISC_ALL_PRIMARY_SERVICES
13	14:16:16.622	HCI_COMMAND_STATUS
14	14:16:17.387	EVT_BLUE_ATT_READ_BY_GROUP_RESP
15	14:16:18.073	EVT_BLUE_ATT_READ_BY_GROUP_RESP
16	14:16:18.776	EVT_BLUE_ATT_READ_BY_GROUP_RESP
17	14:16:19.447	EVT_BLUE_ATT_READ_BY_GROUP_RESP
18	14:16:20.151	EVT_BLUE_ATT_READ_BY_GROUP_RESP
19	14:16:21.509	EVT_BLUE_GATT_ERROR_RESP
20	14:16:21.509	EVT_BLUE_GATT_PROCEDURE_COMPLETE

Packet Details

ameter	Value
ode	0xFF
ter Total ...	0x1A
	0x0C0A
	[0x01,0x08,0x15,0x14 0x28,0x00 0x2A,0x00, x1B,0xC5,0xD5,0xA5,0x02,0x00,0xB4,0x9A,0xE1,0x11,0x3A,0xCF,0x80,0x6E,0x36,0x0B]

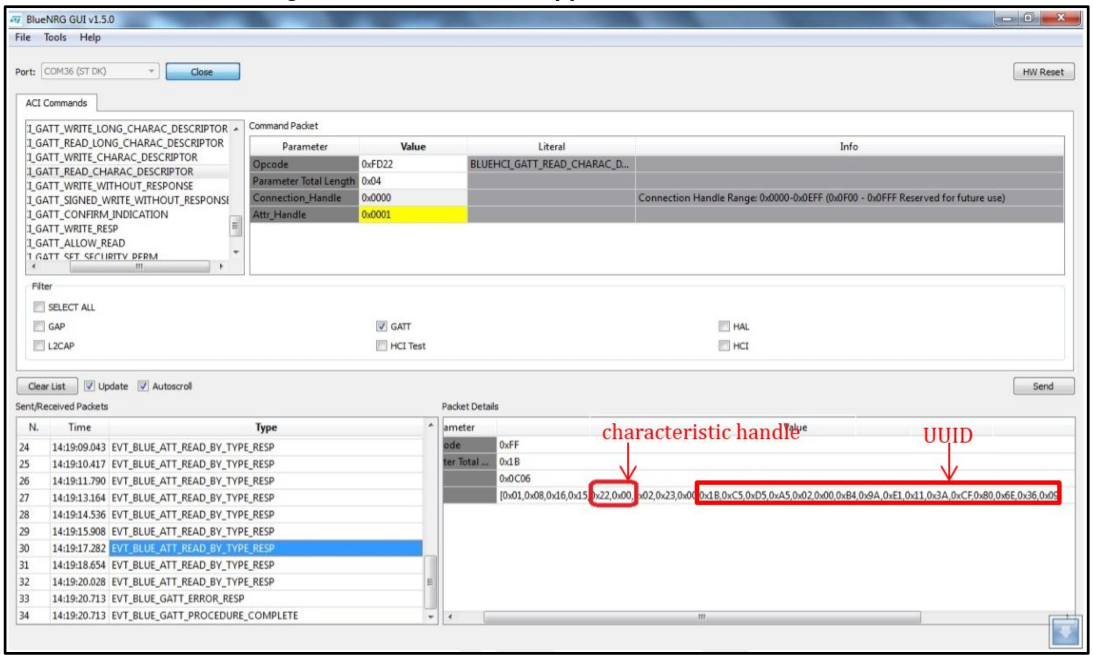
To find the handle of a service the UUID of that particular service (determined from the server code given) has to be matched with the response payload (EVT_BLUE_ATT_READ_BY_GROUP_RESP). The last 16 bytes of the payload is the UUID of a service.

3.5.3.6 Get characteristics supported by BLE peripheral device

The server device supports a number of GATT characteristics and the BlueNRG GUI can obtain this information by issuing the command BLUEHCI_GATT_DISC_ALL_CHARAC_OF_A_SERVICE. Once the command is issued the server responds with EVT_BLUE_ATT_READ_BY_TYPE_RESP for each characteristic supported by the server.

Each response includes the connection handle, the length of the response, the data length, the handle-value pair and the UUID of the characteristic. They are indicated below in the figure below.

Figure 12: Discover all supported characteristics



Command Packet

Parameter	Value	Literal	Info
Opcode	0xFD22	BLUEHCI_GATT_READ_CHARAC_D...	
Parameter Total Length	0x04		
Connection_Handle	0x0000		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Attr_Handle	0x0001		

Sent/Received Packets

N.	Time	Type
24	14:19:09.043	EVT_BLUE_ATT_READ_BY_TYPE_RESP
25	14:19:10.417	EVT_BLUE_ATT_READ_BY_TYPE_RESP
26	14:19:11.790	EVT_BLUE_ATT_READ_BY_TYPE_RESP
27	14:19:13.164	EVT_BLUE_ATT_READ_BY_TYPE_RESP
28	14:19:14.536	EVT_BLUE_ATT_READ_BY_TYPE_RESP
29	14:19:15.908	EVT_BLUE_ATT_READ_BY_TYPE_RESP
30	14:19:17.282	EVT_BLUE_ATT_READ_BY_TYPE_RESP
31	14:19:18.654	EVT_BLUE_ATT_READ_BY_TYPE_RESP
32	14:19:20.028	EVT_BLUE_ATT_READ_BY_TYPE_RESP
33	14:19:20.713	EVT_BLUE_GATT_ERROR_RESP
34	14:19:20.713	EVT_BLUE_GATT_PROCEDURE_COMPLETE

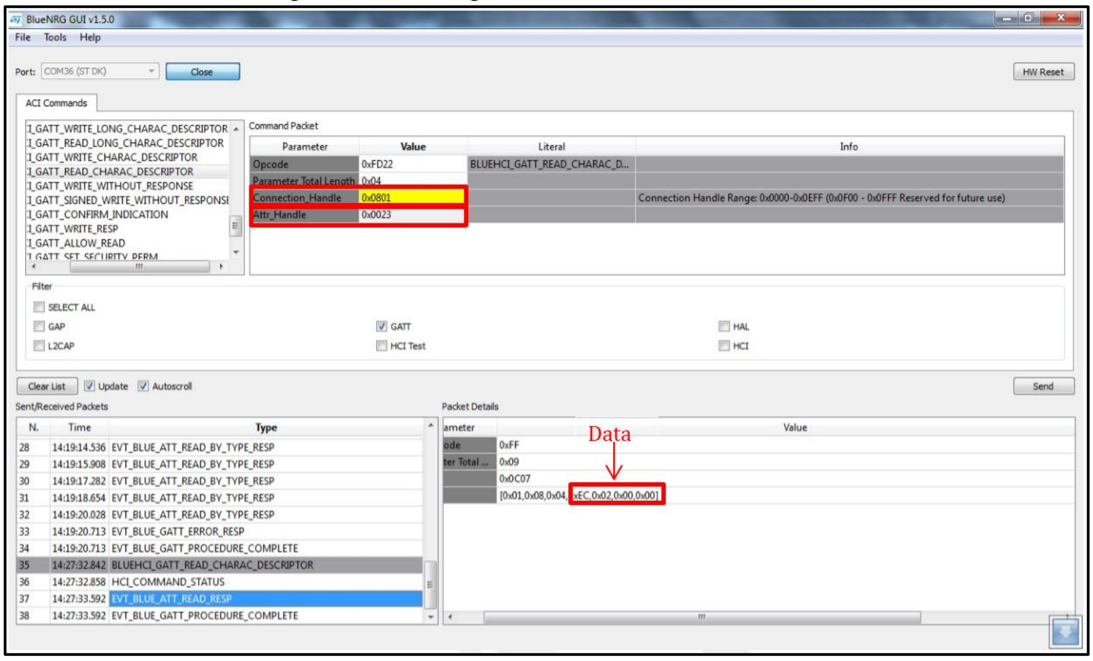
Packet Details

ameter	Value
ode	0xFF
ter Total ...	0x1B
	0x0C06
	[0x01,0x08,0x16,0x15,0x22,0x00,0x02,0x23,0x00,0x1B,0xC5,0xD5,0xA5,0x02,0x00,0xB4,0x9A,0xE1,0x11,0x3A,0xCF,0x80,0x6E,0x36,0x0

3.5.3.7 Read characteristic value

To read a particular characteristic on the server, the command BLUEHCI_GATT_READ_CHARACTERISTIC_VAL can be used. In this command we have to provide two parameters: the attribute handle of the characteristic we are reading and the connection handle. From the previous sections we already know the connection handle value which is “0x0801” in this case. The attribute handle would be the handle of the characteristic handle plus one since the "value" handle of this characteristic lies at offset one from the characteristic handle. Please note that the time characteristic has only one attribute which is a "readable" attribute of the time value. In this case this handle is "0x22+1" or 0x23.

Figure 13: Reading data from time characteristic



I_GATT_WRITE_LONG_CHARAC_DESCRIPTOR
I_GATT_READ_LONG_CHARAC_DESCRIPTOR
I_GATT_WRITE_CHARAC_DESCRIPTOR
I_GATT_READ_CHARAC_DESCRIPTOR
I_GATT_WRITE_WITHOUT_RESPONSE
I_GATT_SIGNED_WRITE_WITHOUT_RESPONSI
I_GATT_CONFIRM_INDICATION
I_GATT_WRITE_RESP
I_GATT_ALLOW_READ
I_GATT_SET_SECURITY_PERM

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFD22	BLUEHCI_GATT_READ_CHARAC_D...	
Parameter Total Length	0x04		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Attr_Handle	0x0023		

Sent/Received Packets

N.	Time	Type
28	14:19:14.536	EVT_BLUE_ATT_READ_BY_TYPE_RESP
29	14:19:15.908	EVT_BLUE_ATT_READ_BY_TYPE_RESP
30	14:19:17.282	EVT_BLUE_ATT_READ_BY_TYPE_RESP
31	14:19:18.654	EVT_BLUE_ATT_READ_BY_TYPE_RESP
32	14:19:20.028	EVT_BLUE_ATT_READ_BY_TYPE_RESP
33	14:19:20.713	EVT_BLUE_GATT_ERROR_RESP
34	14:19:20.713	EVT_BLUE_GATT_PROCEDURE_COMPLETE
35	14:27:32.842	BLUEHCI_GATT_READ_CHARAC_DESCRIPTOR
36	14:27:32.858	HCI_COMMAND_STATUS
37	14:27:33.592	EVT_BLUE_ATT_READ_RESP
38	14:27:33.592	EVT_BLUE_GATT_PROCEDURE_COMPLETE

Packet Details

ameter	Value
ode	0xFF
ter Total ...	0x09
	0x0C07
	[0x01,0x08,0x04,xEC,0x02,0x00,0x00]

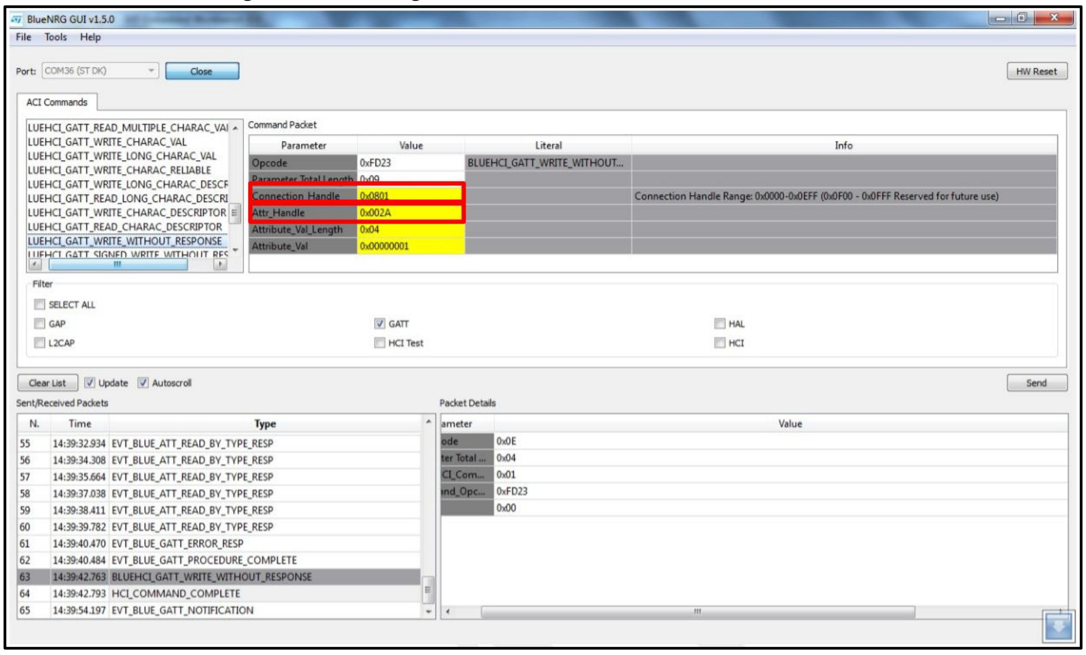
The value of the time characteristic is available in the response from the server EVT_BLUE_ATT_READ_RESP. The value in this case is the 4 bytes at the end of the payload.

3.5.3.8 Write characteristic value

To read a particular characteristic on the server, the command BLUEHCI_GATT_WRITE_WITHOUT_RESPONSE can be used. In this command we have to provide four parameters: the attribute handle of the characteristic we are reading, the connection handle, the data length and the data value to write. From the previous sections we already know that the connection handle value is "0x0801" in this case. The attribute handle would be the handle of the characteristic handle plus one since the "value" handle of this characteristic lies at offset one from the characteristic handle. Please note that the "LED button" characteristic has only one attribute which is a "writable" attribute of the "LED button" value.

As explained in *Section 3.5.1: "Time service"*, by writing data to this characteristic we can toggle the LED2 present on the "peripheral device" and hence when we perform the BLUEHCI_GATT_WRITE_WITHOUT_RESPONSE command, the LED2 should be switched ON/OFF alternately.

Figure 14: Writing data to LED button time characteristic



LUEHCI_GATT_READ_MULTIPLE_CHARAC_VAI
LUEHCI_GATT_WRITE_CHARAC_VAL
LUEHCI_GATT_WRITE_LONG_CHARAC_VAL
LUEHCI_GATT_WRITE_CHARAC_RELIABLE
LUEHCI_GATT_WRITE_LONG_CHARAC_DESCF
LUEHCI_GATT_READ_LONG_CHARAC_DESCRI
LUEHCI_GATT_WRITE_CHARAC_DESCRIPTOR
LUEHCI_GATT_READ_CHARAC_DESCRIPTOR
LUEHCI_GATT_WRITE_WITHOUT_RESPONSE
LUEHCI_GATT_SIGNED_WRITE_WITHOUT_RES

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFD23	BLUEHCI_GATT_WRITE_WITHOUT...	
Parameter Total Length	0x09		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Attr_Handle	0x002A		
Attribute_Val_Length	0x04		
Attribute_Val	0x00000001		

Sent/Received Packets

N.	Time	Type
55	14:39:32.934	EVT_BLUE_ATT_READ_BY_TYPE_RESP
56	14:39:34.308	EVT_BLUE_ATT_READ_BY_TYPE_RESP
57	14:39:35.664	EVT_BLUE_ATT_READ_BY_TYPE_RESP
58	14:39:37.038	EVT_BLUE_ATT_READ_BY_TYPE_RESP
59	14:39:38.411	EVT_BLUE_ATT_READ_BY_TYPE_RESP
60	14:39:39.782	EVT_BLUE_ATT_READ_BY_TYPE_RESP
61	14:39:40.470	EVT_BLUE_GATT_ERROR_RESP
62	14:39:40.484	EVT_BLUE_GATT_PROCEDURE_COMPLETE
63	14:39:42.763	BLUEHCI_GATT_WRITE_WITHOUT_RESPONSE
64	14:39:42.793	HCI_COMMAND_COMPLETE
65	14:39:54.197	EVT_BLUE_GATT_NOTIFICATION

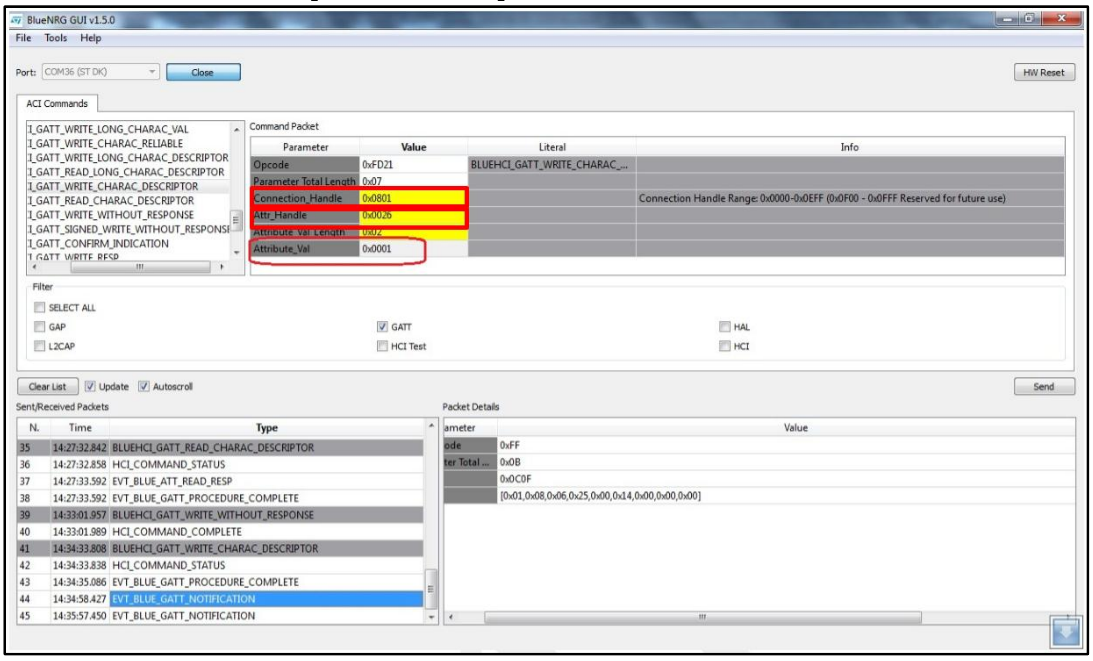
Packet Details

ameter	Value
ode	0x0E
ter Total ...	0x04
CI_Com...	0x01
nd_Opc...	0xFD23
	0x00

3.5.3.9 Obtain notification for a characteristic

To enable notifications for notifiable characteristics (i.e. characteristics which have CHAR_PROP_NOTIFY property), BLUEHCI_GATT_WRITE_CHARAC_DESCRIPTOR command can be used. This command can be used to write a descriptor to an attribute. We must set correct values of attribute handle, and configuration data while using this command. For enabling notification, the configuration data is {0x00, 0x01}. In this case, handle of the attribute for "Minutes characteristic" lies at offset two from the characteristic handle.

Figure 15: Enabling notifications from server



I_GATT_WRITE_LONG_CHARAC_VAL
I_GATT_WRITE_CHARAC_RELIABLE
I_GATT_WRITE_LONG_CHARAC_DESCRIPTOR
I_GATT_READ_LONG_CHARAC_DESCRIPTOR
I_GATT_WRITE_CHARAC_DESCRIPTOR
I_GATT_READ_CHARAC_DESCRIPTOR
I_GATT_WRITE_WITHOUT_RESPONSE
I_GATT_SIGNED_WRITE_WITHOUT_RESPONSE
I_GATT_CONFIRM_INDICATION
I_GATT_WRITE_RESP

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFD21	BLUEHCI_GATT_WRITE_CHARAC_...	
Parameter Total Length	0x07		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Attr_Handle	0x0026		
Attribute_Val_Length	0x02		
Attribute_Val	0x0001		

Sent/Received Packets

N.	Time	Type
35	14:27:32.842	BLUEHCI_GATT_READ_CHARAC_DESCRIPTOR
36	14:27:32.858	HCI_COMMAND_STATUS
37	14:27:33.592	EVT_BLUE_ATT_READ_RESP
38	14:27:33.592	EVT_BLUE_GATT_PROCEDURE_COMPLETE
39	14:33:01.957	BLUEHCI_GATT_WRITE_WITHOUT_RESPONSE
40	14:33:01.989	HCI_COMMAND_COMPLETE
41	14:34:33.808	BLUEHCI_GATT_WRITE_CHARAC_DESCRIPTOR
42	14:34:33.838	HCI_COMMAND_STATUS
43	14:34:35.086	EVT_BLUE_GATT_PROCEDURE_COMPLETE
44	14:34:58.427	EVT_BLUE_GATT_NOTIFICATION
45	14:35:57.450	EVT_BLUE_GATT_NOTIFICATION

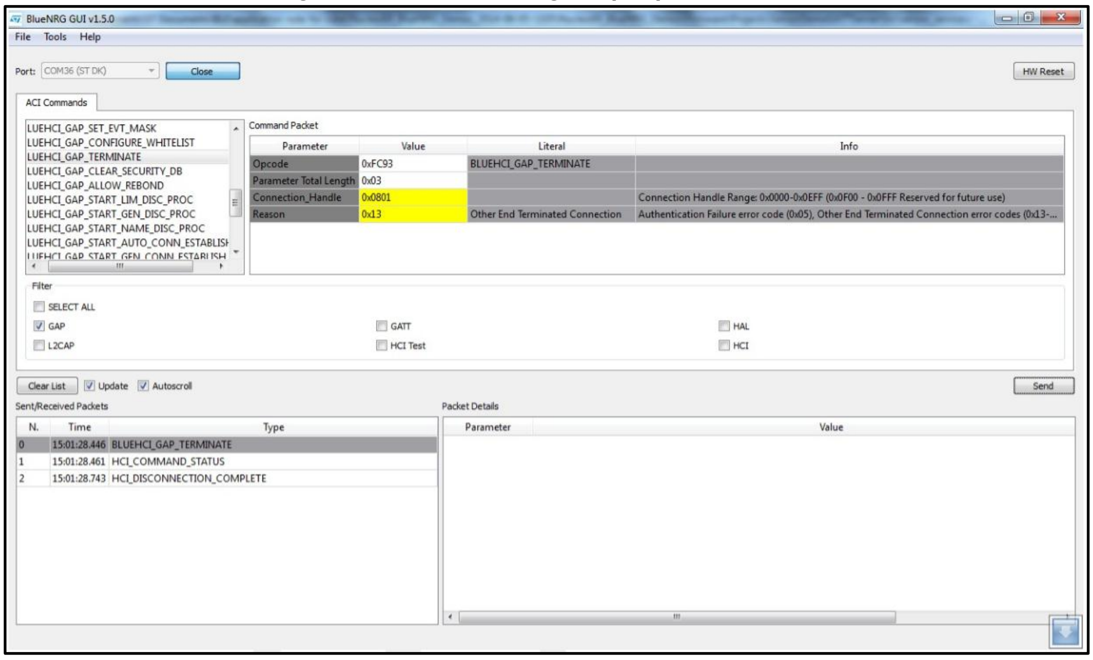
Packet Details

ameter	Value
ode	0xFF
ter Total ...	0x0B
	0x0C0F
	[0x01,0x08,0x06,0x25,0x00,0x14,0x00,0x00,0x00]

3.5.3.10 Disconnecting from remote device

To disconnect the peripheral device from the central device, BLUEHCI_GAP_TERMINATE command can be used as shown below:

Figure 16: Disconnecting the peripheral device



LUEHCI_GAP_SET_EVT_MASK
LUEHCI_GAP_CONFIGURE_WHITELIST
LUEHCI_GAP_TERMINATE
LUEHCI_GAP_CLEAR_SECURITY_DB
LUEHCI_GAP_ALLOW_REBOND
LUEHCI_GAP_START_LIM_DISC_PROC
LUEHCI_GAP_START_GEN_DISC_PROC
LUEHCI_GAP_START_NAME_DISC_PROC
LUEHCI_GAP_START_AUTO_CONN_ESTABLISH
LUEHCI_GAP_START_GEN_CONN_ESTABLISH

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC93	BLUEHCI_GAP_TERMINATE	
Parameter Total Length	0x03		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Reason	0x13	Other End Terminated Connection	Authentication Failure error code (0x05), Other End Terminated Connection error codes (0x13-...

Filter
SELECT ALL
[✓ GAP] GATT HAL
L2CAP HCI Test HCI

Clear List [✓ Update] [✓ Autoscroll] Send

Sent/Received Packets

N.	Time	Type
0	15:01:28.446	BLUEHCI_GAP_TERMINATE
1	15:01:28.461	HCI_COMMAND_STATUS
2	15:01:28.743	HCI_DISCONNECTION_COMPLETE

Packet Details

Parameter	Value

3.5.4 Adding security to sample application

In this section we show a slight modification of the sample application code in order to protect one characteristic and describe the proper way to perform the pairing from the BlueNRG GUI in order to gain the authorization to access it.

3.5.4.1 Protecting the characteristic

In order to add read protection to a characteristic it is enough to modify the secPermissions flag of the call to aci_gatt_add_char. The example below show how to set up the Time service seconds characteristic with a protection requiring an authenticated pairing and that will guarantee the data exchange will be encrypted.

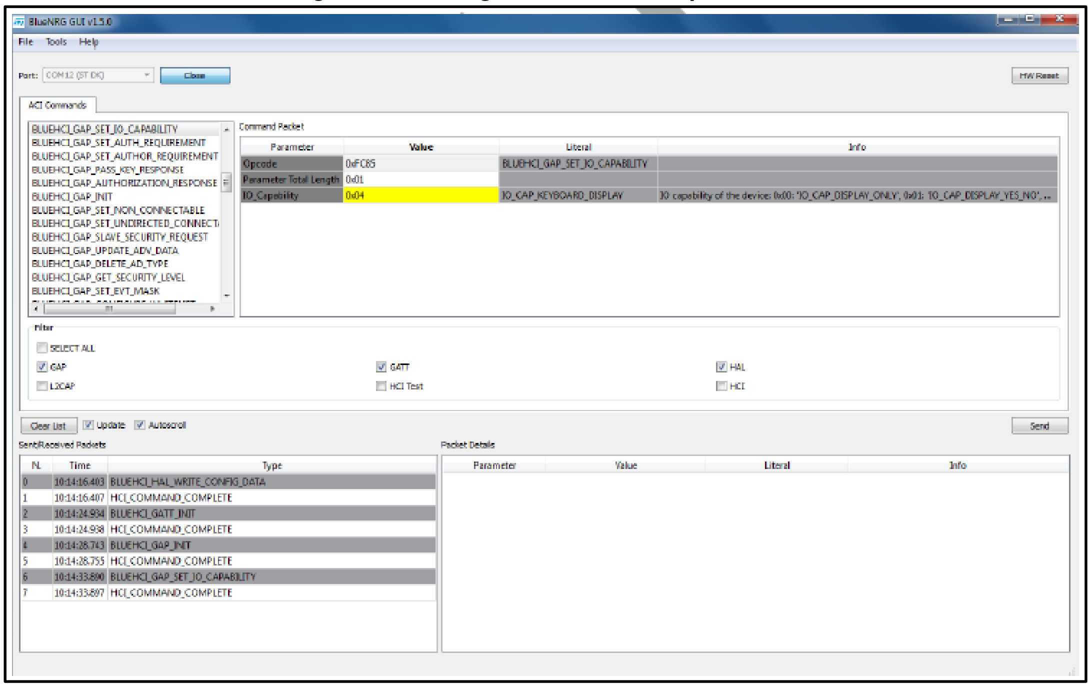
```
    ret = aci_gatt_add_char(timeServHandle, UUID_TYPE_128, uuid, 4,
CHAR_PROP_READ, ATTR_PERMISSION_ENCRY_READ | ATTR_PERMISSION_AUTHEN_READ, 0, 16, 0,
&secondsCharHandle);
```

If the GATT client will try to read this characteristic without having performed an authenticated pairing, the reading will return an error.

3.5.4.2 Performing the pairing

In order to perform the paring with the BlueNRG GUI the user should first declare its I/O capabilities, after having initialized the USB dongle as done in *Section 3.5.3.2: "Initializing the USB dongle"*, with the command BLUEHCI_GAP_SET_IO_CAPABILITY as depicted in the figure below, where the selected capability is IO_CAP_KEYBOARD_DISPLAY as on the PC running the BlueNRG GUI there is both a keyboard and a display.

Figure 17: Setting the device I/O capabilities



Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC85	BLUEHCI_GAP_SET_IO_CAPABILITY	
Parameter Total Length	0x01		
IO_Capability	0x04	IO_CAP_KEYBOARD_DISPLAY	IO capability of the device: 0x00: 'IO_CAP_DISPLAY_ONLY', 0x01: 'IO_CAP_DISPLAY_YES_NO', ...

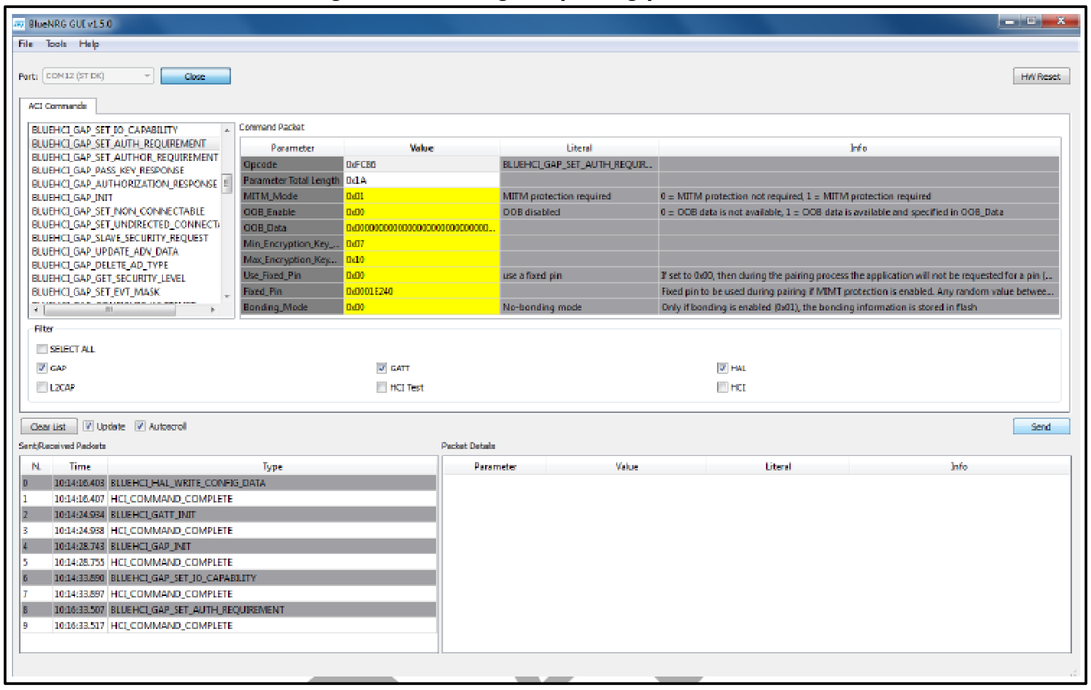
Sent/Received Packets

N.	Time	Type
0	10:14:16.403	BLUEHCI_HAL_WRITE_CONFIG_DATA
1	10:14:16.407	HCI_COMMAND_COMPLETE
2	10:14:24.934	BLUEHCI_GATT_INIT
3	10:14:24.938	HCI_COMMAND_COMPLETE
4	10:14:28.743	BLUEHCI_GAP_INIT
5	10:14:28.755	HCI_COMMAND_COMPLETE
6	10:14:33.890	BLUEHCI_GAP_SET_IO_CAPABILITY
7	10:14:33.897	HCI_COMMAND_COMPLETE

Apart from the I/O capabilities the user needs to configure the parameters for the pairing. This is achieved through the command BLUEHCI_GAP_SET_AUTH_REQUIREMENT as shown in the figure below.

To match with the code of the sample application we set man-in-the-middle protection, OOB disabled, and configured a fixed PIN of value 123456 (0x1E240).

Figure 18: Setting the pairing parameters



Command Packet

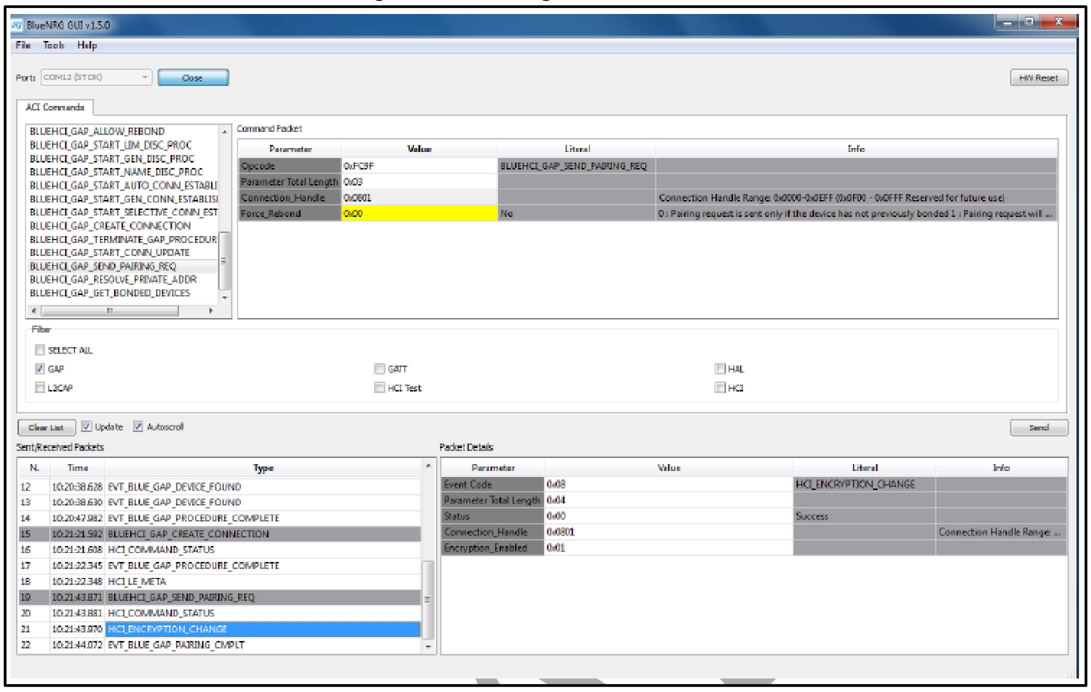
Parameter	Value	Literal	Info
Opcode	0xFC86	BLUEHCI_GAP_SET_AUTH_REQUIR...	
Parameter Total Length	0x1A		
MITM_Mode	0x01	MITM protection required	0 = MITM protection not required, 1 = MITM protection required
OOB_Enable	0x00	OOB disabled	0 = OOB data is not available, 1 = OOB data is available and specified in OOB_Data
OOB_Data	0x000000000000000000000000000000...		
Min_Encryption_Key_...	0x07		
Max_Encryption_Key...	0x10		
Use_Fixed_Pin	0x00	use a fixed pin	If set to 0x00, then during the pairing process the application will not be requested for a pin (...
Fixed_Pin	0x0001E240		Fixed pin to be used during pairing if MIMT protection is enabled. Any random value betwee...
Bonding_Mode	0x00	No-bonding mode	Only if bonding is enabled (0x01), the bonding information is stored in flash

Sent/Received Packets

N.	Time	Type
0	10:14:16.403	BLUEHCI_HAL_WRITE_CONFIG_DATA
1	10:14:16.407	HCI_COMMAND_COMPLETE
2	10:14:24.934	BLUEHCI_GATT_INIT
3	10:14:24.938	HCI_COMMAND_COMPLETE
4	10:14:28.743	BLUEHCI_GAP_INIT
5	10:14:28.755	HCI_COMMAND_COMPLETE
6	10:14:33.890	BLUEHCI_GAP_SET_IO_CAPABILITY
7	10:14:33.897	HCI_COMMAND_COMPLETE
8	10:16:33.507	BLUEHCI_GAP_SET_AUTH_REQUIREMENT
9	10:16:33.517	HCI_COMMAND_COMPLETE

After this operation, and after a connection has been set up as described in *Section 3.5.3.4: "Connecting to BLE peripheral device"*, the user can start the pairing procedure with the command BLUEHCI_GAP_SEND_PAIRING_REQUEST which will result in an HCI_ENCRYPTION_CHANGE and in an event EVT_BLUE_GAP_PAIRING_CMPL indicating the result of the pairing. This procedure is depicted in the figure below.

Figure 19: Pairing with the device



BLUEHCI_GAP_ALLOW_REBOND
BLUEHCI_GAP_START_LIM_DISC_PROC
BLUEHCI_GAP_START_GEN_DISC_PROC
BLUEHCI_GAP_START_NAME_DISC_PROC
BLUEHCI_GAP_START_AUTO_CONN_ESTABLI
BLUEHCI_GAP_START_GEN_CONN_ESTABLISI
BLUEHCI_GAP_START_SELECTIVE_CONN_EST
BLUEHCI_GAP_CREATE_CONNECTION
BLUEHCI_GAP_TERMINATE_GAP_PROCEDUR
BLUEHCI_GAP_START_CONN_UPDATE
BLUEHCI_GAP_SEND_PAIRING_REQ
BLUEHCI_GAP_RESOLVE_PRIVATE_ADDR
BLUEHCI_GAP_GET_BONDED_DEVICES

Command Packet

Parameter	Value	Literal	Info
Opcode	0xFC3F	BLUEHCI_GAP_SEND_PAIRING_REQ	
Parameter Total Length	0x03		
Connection_Handle	0x0801		Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)
Force_Rebond	0x00	No	0 : Pairing request is sent only if the device has not previously bonded 1 : Pairing request will ...

Filter: SELECT ALL [✓ GAP] GATT HAL L2CAP HCI Test HCI

[✓ Update] [✓ Autoscroll]

Sent/Received Packets

N.	Time	Type
12	10:20:38.628	EVT_BLUE_GAP_DEVICE_FOUND
13	10:20:38.630	EVT_BLUE_GAP_DEVICE_FOUND
14	10:20:47.982	EVT_BLUE_GAP_PROCEDURE_COMPLETE
15	10:21:21.592	BLUEHCI_GAP_CREATE_CONNECTION
16	10:21:21.608	HCI_COMMAND_STATUS
17	10:21:22.345	EVT_BLUE_GAP_PROCEDURE_COMPLETE
18	10:21:22.348	HCI_LE_META
19	10:21:43.871	BLUEHCI_GAP_SEND_PAIRING_REQ
20	10:21:43.881	HCI_COMMAND_STATUS
21	10:21:43.970	HCI_ENCRYPTION_CHANGE
22	10:21:44.072	EVT_BLUE_GAP_PAIRING_CMPLT

Packet Details

Parameter	Value	Literal	Info
Event Code	0x08	HCI_ENCRYPTION_CHANGE	
Parameter Total Length	0x04		
Status	0x00	Success	
Connection_Handle	0x0801		Connection Handle Range: ...
Encryption_Enabled	0x01		

After a successful pairing it is now possible to proceed to reading the protected characteristic with the normal procedure described in *Section 3.5.3.7: "Read characteristic value"*.

4 System setup guide

4.1 Hardware description

This section describes the hardware components needed for developing a sensor-based application.

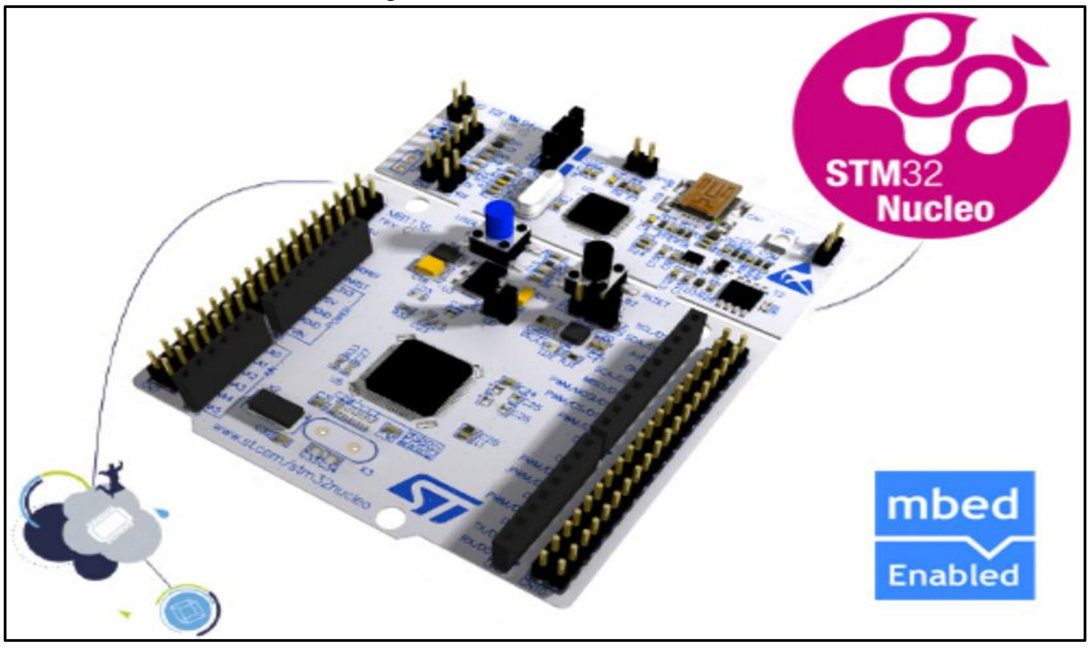
The following sub-sections describe the individual components.

4.1.1 STM32 Nucleo platform

The STM32 Nucleo boards provide an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller line. The Arduino^{TM} connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized expansion boards. The STM32 Nucleo board does not require any separate probes as it integrates the ST-LINK/V2-1 debugger/programmer. The board comes with the comprehensive STM32 HAL software library together with various packaged software examples.

Information regarding the STM32 Nucleo board is available on st.com at: http://www.st.com/stm32nucleo

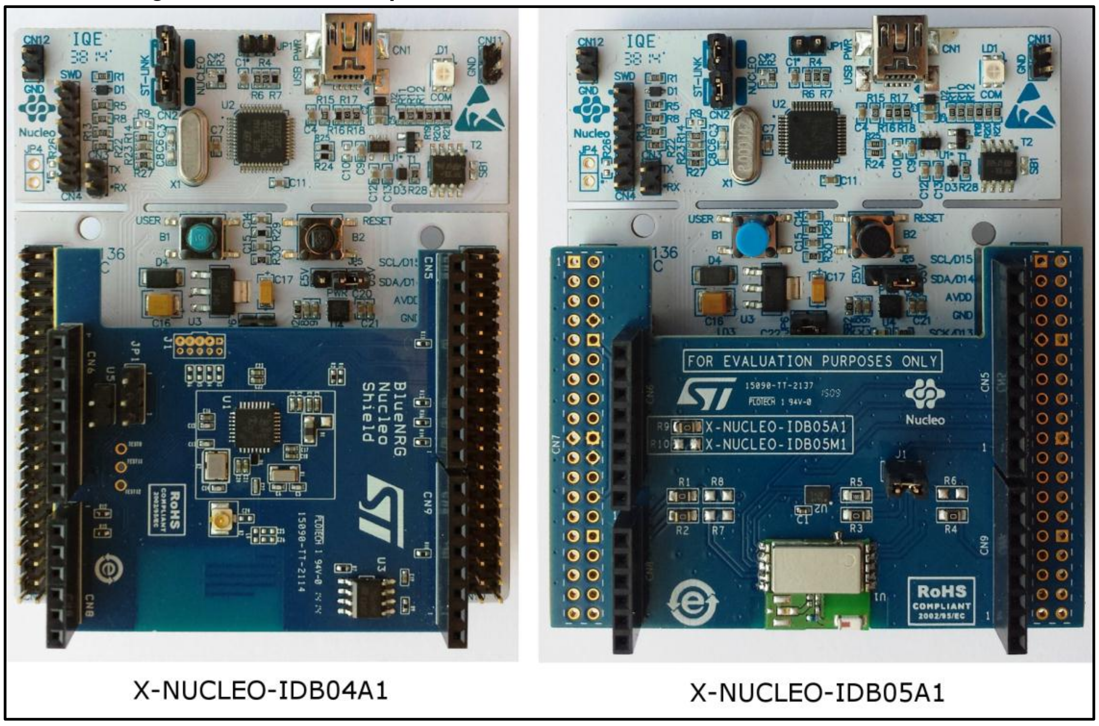
Figure 20: STM32 Nucleo board



4.1.2 BlueNRG expansion board

The X-NUCLEO-IDB04A1 and X-NUCLEO-IDB05A1 expansion boards are compatible with the STM32 Nucleo boards. They each contain a BlueNRG component, which is a BLE single-mode network processor, compliant with Bluetooth specification v4.0 and v4.1 respectively. The BlueNRG can act as master or slave. The entire Bluetooth Low Energy stack runs on the embedded Cortex M0 core. The BlueNRG offers the option of interfacing with external microcontrollers using SPI transport layer.

Figure 21: BlueNRG expansion board connected to STM32 Nucleo board



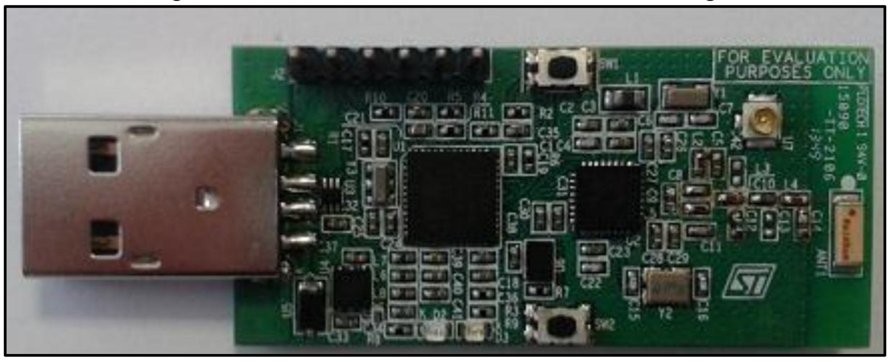
The documentation related to X-NUCLEO-IDB04A1 and X-NUCLEO-IDB05A1 is available on www.st.com.

4.1.3 BlueNRG USB dongle

The STEVAL-IDB003V1 is an evaluation board based on BlueNRG, a low power Bluetooth Smart IC, compliant with the Bluetooth 4.0 specifications and supporting both master and slave roles.

The STEVAL-IDB003V1 has a USB connector for PC GUI interaction and firmware update.

Figure 22: STEVAL-IDB003V1 BlueNRG USB dongle



The STEVAL-IDB003V1 evaluation board firmware and related documentation is available on www.st.com

4.1.4 Software description

The following software components are needed in order to setup the suitable development environment for creating applications for the STM32 Nucleo equipped with the BlueNRG expansion board:

- X-CUBE-BLE1: an expansion for STM32Cube dedicated to Bluetooth Low Energy applications development. The X-CUBE-BLE1 firmware and related documentation is available on www.st.com.
- Development toolchain and compiler: The STM32Cube expansion software supports the three following environments:
  – IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-Link
  – RealView Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK
  – Atollic TrueSTUDIO® for ARM® Pro + ST-LINK

4.2 Hardware and software setup

This section describes the hardware and software setup procedures. It also describes the system setup needed for the above.

4.2.1 Hardware setup

To develop a BLE application, the following hardware is needed:

1. One STM32 Nucleo development platform
2. One BlueNRG expansion board (order code: X-NUCLEO-IDB04A1 or X-NUCLEO-IDB05A1)
3. One USB type A to Mini-B USB cable to connect the Nucleo to the PC
4. One BlueNRG USB dongle (order code: STEVAL-IDB003V1)

The BlueNRG USB dongle is not mandatory but is useful for testing BLE applications running on the Nucleo platform. If you don't want to use the USB dongle, as an alternative, you can replace it with the following additional hardware:

1. One STM32 Nucleo Development platform
2. One BlueNRG expansion board (order code: X-NUCLEO-IDB04A1 or X-NUCLEO-IDB05A1)
3. One USB type A to Mini-B USB cable to connect the Nucleo to the PC
4. One USB type A to Mini-B USB cable to connect the Nucleo to the PC

4.2.2 Software setup

This section lists the minimum requirements for the developer to setup the software environment, run the sample testing scenario based on the GUI utility and customize applications.

- Development toolchains and compilers: Please select one of the integrated development environments supported by the STM32Cube expansion software. Please read the system requirements and setup information provided by the selected IDE provider.
- GUI utility: The BlueNRG GUI utility has following minimum requirements:
  – PC with Intel or AMD processor running one of following Microsoft operating system: Win XP SP3 Vista/7
  – At least 128 MBs of RAM
  – 2 X USB ports
  – 40 MB of hard disk space

4.2.3 System setup guide

This section describes how to setup different hardware parts before writing and executing an application on the STM32 Nucleo board with BlueNRG expansion board.

4.2.3.1 BlueNRG USB dongle setup

The BlueNRG USB dongle allows to easily add BLE functionalities to a user PC by just plugging it on a PC USB port. The USB dongle can be used as a simple interface between the BlueNRG device and a GUI application on the PC. The on board STM32L microcontroller can also be programmed, so the board can be used to develop applications that need to use BlueNRG. The board can be powered through the USB connector, which can also be used for I/O interaction with a USB Host. The board has also two buttons and two LEDs for user interaction.

The reader can refer to user manual UM1686, available on www.st.com, for more details.

4.2.3.2 BlueNRG GUI setup

The BlueNRG GUI included in the software package is a graphical user interface that can be used to interact and evaluate the capabilities of the remote BlueNRG network processor through the local BlueNRG USB dongle.

This utility can send standard and vendor-specific HCI commands to the controller and receive events from it. It lets the user configure each field of the HCI command packets to be sent and analyzes all received packets. In this way BlueNRG can be easily managed at low level.

In order to use the BlueNRG GUI, make sure you have correctly set up your hardware and software (BlueNRG GUI installed) according to the requirements described in UM1686.

4.2.3.3 STM32 Nucleo and BlueNRG expansion boards setup

The STM32 Nucleo development motherboard allows the exploitation of the BLE capabilities provided by the BlueNRG network processor.

The Nucleo board integrates the ST-LINK/V2-1 debugger/programmer. The developer can download the relevant version of the ST-LINK/V2-1 USB driver by looking ST-LINK008 or STSW-LINK009 on st.com (according to the MS Windows OS).

The X-NUCLEO-IDB04A1 or X-NUCLEO-IDB05A1 BlueNRG expansion boards can be easily connected to the Nucleo motherboard through the Arduino UNO R3 extension connector. The BlueNRG expansion board is capable of interfacing with the external STM32 microcontroller on the STM32 Nucleo board using the SPI transport layer.

Note: It is assumed that the BlueNRG expansion board is preprogrammed with BlueNRG firmware image version V6.3. Once the USB driver has been installed and the BlueNRG expansion board has been connected, the STM32 Nucleo board can be plugged on a PC USB port through a USB cable.

Getting started documentation is available on www.st.com

5 References

1. ***UM1755****: BlueNRG Bluetooth LE stack application command interface (ACI)*
2. ***UM1686****: BlueNRG development kits*

6 Revision history

Table 2: Document revision history

Date	Revision	Changes
27-Aug-2015	1	Initial release.