



STMicroelectronics

APPLICATION NOTE

2014-07-04

Developer's Guide to create Bluetooth® Low Energy Applications using STM32 Nucleo and BlueNRG

Table of Contents

1	List of Figures	4
2	References	5
3	Acronyms and Abbreviations	6
4	Getting Started	8
4.1	Hardware Description	8
4.1.1	STM32L0 Nucleo	8
4.1.2	BlueNRG expansion board	9
4.1.3	BlueNRG USB Dongle	9
4.2	Software Description	10
4.2.1	STM32Cube	10
4.2.2	Firmware for STM32 Nucleo with X-NUCLEO-IDB04A1 expansion board	11
4.3	Hardware and Software Setup	11
4.3.1	Hardware Setup	11
4.3.2	Software Setup	11
4.3.2.1	Development Tool-chains and Compilers	11
4.3.2.2	GUI Utility	11
4.3.3	System Setup Guide	12
4.3.3.1	BlueNRG USB Dongle setup	12
4.3.3.2	BlueNRG GUI setup	12
4.3.3.3	STM32 Nucleo and BlueNRG expansion boards setup	12
5	Bluetooth low energy (BLE)	14
5.1	Bluetooth Operating modes	14
5.2	Bluetooth low energy Software Partitioning	14
5.3	Profiles and Services	16
5.4	Bluetooth low energy state machine	17
6	Software Overview	19
6.1	Software Architecture	19
6.2	STM32Cube HAL	19
6.3	Board Support Package (BSP)	20

6.4	BlueNRG HAL	20
6.5	ACI	20

7 Guide for writing application 21

7.1	Relevant APIs for BlueNRG application	21
7.1.1	Initialization	21
7.1.1.1	Initializing STM32 Cube HAL	21
7.1.1.2	Initializing Nucleo board peripherals	21
7.1.1.3	Initializing BlueNRG HAL and HCI	22
7.1.1.4	Initialization and Services characteristics	22
7.1.2	Security Requirements	22
7.1.3	Connectable mode	22
7.1.4	Connection with central device	23
7.2	SensorDemo application description	23
7.3	Time service	24
7.3.1	Adding Time service	24
7.3.2	Update and Notify characteristic value	27
7.4	LED service	30
7.4.1	Adding GATT service and characteristics	30
7.4.2	Obtaining characteristic value	32

8 Testing the sample application 35

8.1	Testing SensorDemo application using BlueNRG GUI	35
8.1.1	Initializing the USB Dongle	36
8.1.2	Scanning for BLE peripheral device	36
8.1.3	Connecting to BLE peripheral device	37
8.1.4	Get Services supported by BLE peripheral device	38
8.1.5	Get Characteristics supported BLE peripheral device	39
8.1.6	Read characteristic value	40
8.1.7	Write characteristic value	41
8.1.8	Obtain notification for a characteristic	42
8.1.9	Disconnecting from remote device	44

1 List of Figures

Figure 1 STM32 Nucleo Board	8
Figure 2 BlueNRG expansion board connected to STM32 Nucleo board.....	9
Figure 3 STEVAL-IDB003V1 BlueNRG USB Dongle	10
Figure 4 Bluetooth LE Protocol Stack.....	14
Figure 5 Structure of a GATT based Profile	16
Figure 6 BLE State Machine.....	17
Figure 7 STM32 Nucleo + BlueNRG expansion board Software architecture.....	19
Figure 8 Hardware components setup connected	35
Figure 9: Initializing the BlueNRG Dongle	36
Figure 10: Scanning for devices	37
Figure 11: Create Connection.....	38
Figure 12 Discover all supported Services	39
Figure 13: discover all supported characteristics	40
Figure 14: Reading Data from Time Characteristic	41
Figure 15: Writing Data to LED button characteristic.....	42
Figure 16: Enabling notifications from server	43
Figure 17: Disconnecting the peripheral device	44

2 References

- [1] “BlueNRG Bluetooth LE stack application command interface (ACI)”, User manual UM1755, available from st.com at http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00114498.pdf
- [2] “BlueNRG development kits”, User manual UM1686, available from st.com at http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00099259.pdf

DRAFT

3 Acronyms and Abbreviations

Acronym	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
IDE	Integrated Development Environment
L2CAP	Logical link control and adaptation protocol
LED	Light emitting diode
LL	Link layer
MCU	Micro controller unit
PHY	Physical layer
SIG	Special interest group
SM	Security manager

Application Note

SPI	Serial Peripheral Interface
UUID	Universally unique identifier

DRAFT

4 Getting Started

This document describes how to develop a Bluetooth low energy application using the STM32 Nucleo platform and the BlueNRG expansion board (X-NUCLEO-IDB04A1), within the STM32Cube software environment. The BlueNRG is a very low power Bluetooth Low Energy (BLE) single-mode Network Processor, compliant with Bluetooth specifications core 4.0.

4.1 Hardware Description

This section describes the hardware components needed for developing a BLE application.

The following sub-sections describe the individual components.

4.1.1 STM32L0 Nucleo

The STM32 L0 Nucleo board is a development platform for low-power applications that belongs to the STM32 Nucleo family. It provides an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller lines. The Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the Nucleo open development platform with a wide choice of specialized expansion boards. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples.

The STM32 Nucleo L0 firmware and related documentation is available on st.com at

<http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847/PF260001>.

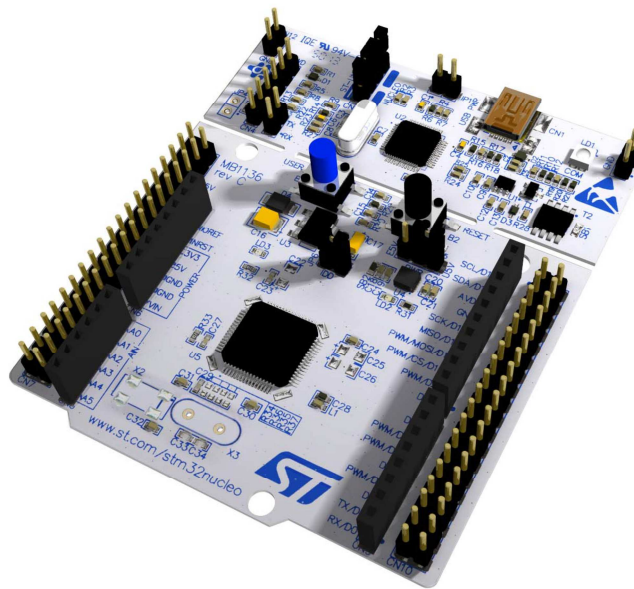


Figure 1 STM32 Nucleo Board

4.1.2 BlueNRG expansion board

The X-NUCLEO-IDB04A1 expansion board is compatible with the STM32 Nucleo boards. It contains a BlueNRG component, which is a BLE single-mode network processor, compliant with Bluetooth specification v4.0. 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 2 BlueNRG expansion board connected to STM32 Nucleo board

The X-NUCLEO-IDB04A1 firmware and related documentation is available on st.com at <http://www.st.com/web/en/catalog/tools/FM116/SC1075/PF260517>.

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.

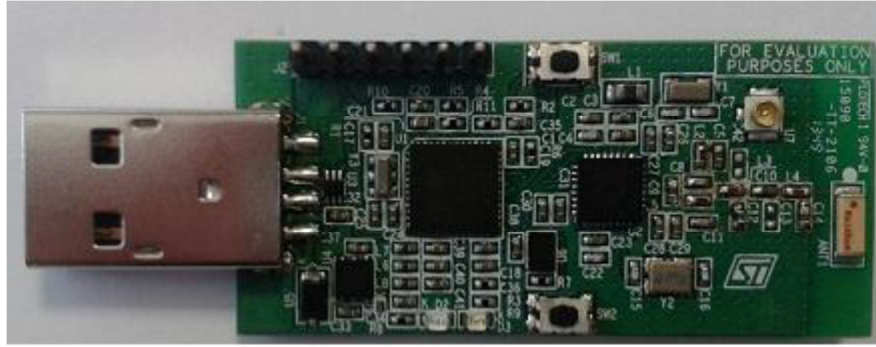


Figure 3 STEVAL-IDB003V1 BlueNRG USB Dongle

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

<http://www.st.com/web/en/catalog/tools/PF260386>

4.2 Software Description

The following software components are needed in order to setup the suitable development environment for creating Bluetooth low energy applications:

- STM32Cube environment and related firmware for STM32Nucleo and BlueNRG expansion board
- Development tool-chain and Compiler. IAR Embedded Workbench is the development environment used for the applications described in this document.

4.2.1 STM32Cube

STM32Cube is a development framework which provides tools and libraries to develop C applications on STM32 series platforms.

STM32Cube comprises the STM32CubeL0 platform which includes the STM32Cube HAL (an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio), plus a consistent set of middleware components (such as RTOS, USB, FatFS and STM32 touch sensing). All embedded software utilities come with a full set of examples.

STM32CubeL0 gathers in one single package all the generic embedded software components required to develop an application on STM32L0 microcontrollers.

STM32CubeL0 is fully compatible with STM32CubeMX code generator that allows generating initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for developer convenience.

The STM32CubeL0 firmware and related documentation is available on st.com at

<http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1743/PF260508>.

4.2.2 Firmware for STM32 Nucleo with X-NUCLEO-IDB04A1 expansion board

STSW-IDB04A1 is a package that contains a firmware with the STM32 Cube framework compatible with the STM32 Nucleo and the X-NUCLEO-IDB0401 boards. This package contains some sample applications that are used to explain how to write Bluetooth low energy applications. The same sample applications are referenced in further sections of this document.

The firmware and related documentation is available on st.com at

<http://www.st.com/web/en/catalog/tools/FM116/SC1075/PF260517>.

4.3 Hardware and Software Setup

This section describes hardware and software setup procedure for writing BLE applications. It also describes the system setup needed for the above.

4.3.1 Hardware Setup

For developing a BLE application following hardware is needed:

1. One STM32 Nucleo Development platform (suggested order code: NUCLEO- L053R8)
2. One BlueNRG expansion board (see Figure 1, order code: X-NUCLEO-IDB04A1)
3. One BlueNRG USB Dongle (see Figure 4, order code: STEVAL-IDB003V1)
4. One USB type A to Mini-B USB cable to connect the Nucleo to the PC

The BlueNRG USB dongle is not strictly mandatory but is useful for testing BLE applications running on the Nucleo platform, as outlined in Chapter 8.

4.3.2 Software Setup

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

4.3.2.1 Development Tool-chains and Compilers

IAR Embedded Workbench for ARM (EWARM) tool-chain V7.20

The IAR tool-chain has the following minimum requirements:

- PC with Intel® or AMD® processor running one of the following Microsoft® operating systems
 - Windows XP SP3
 - Windows Vista
 - Windows7

4.3.2.2 GUI Utility

The BlueNRG GUI utility has following minimum requirements:

- PC with Intel or AMD processor running one of following Microsoft operating system:
 - Windows XP SP3

- Windows Vista
- Windows 7
- At least 128 MBs of RAM
- 2 X USB ports
- 40 MB of hard disk space

4.3.3 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.

4.3.3.1 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 <http://www.st.com/web/catalog/tools/FM116/SC1075/PF260386> and related documentation (UM1686) for more details.

4.3.3.2 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, section 3.1.

4.3.3.3 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-SW-LINK008 or STSW-LINK009 on www.st.com (according to the MS Windows OS).

The BlueNRG expansion board X-NUCLEO-IDB04A1 can be easily connected to the Nucleo motherboard through the Arduino UNO R3 extension connector, see Figure 2. The BlueNRG

Application Note

expansion board is capable of interfacing with the external STM32 microcontroller on Nucleo 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 st.com at

<http://www.st.com/web/en/catalog/tools/FM116/SC1075/PF260517>.

DRAFT

5 Bluetooth low energy (BLE)

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 the fitness, security, healthcare etc. using devices which run on coin cell batteries, and can remain operative for “months or years” without draining out battery.

5.1 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 BLE protocol. Single mode devices are called “Bluetooth Smart”.

5.2 Bluetooth low energy Software Partitioning

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

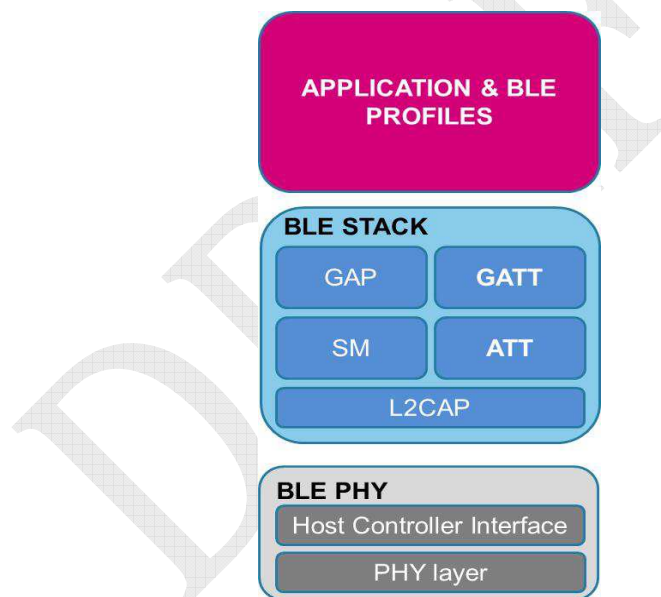


Figure 4 Bluetooth LE Protocol Stack

A typical BLE system consists of a 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 a 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.

Application Note

PHY layer insures communication with stack & 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.

5.3 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.

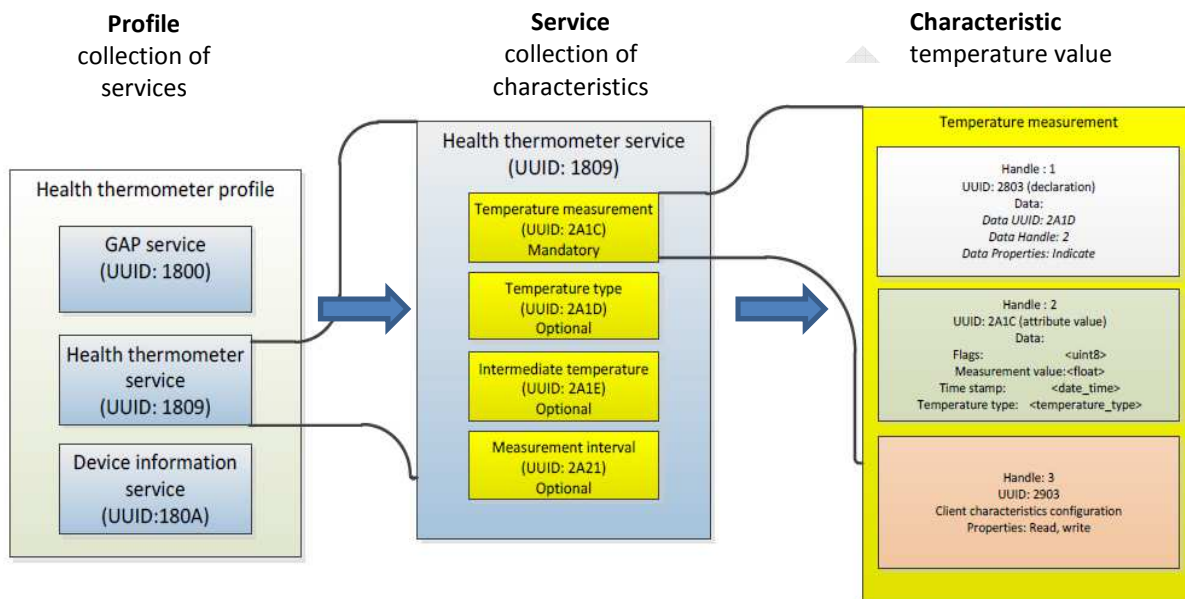


Figure 5 Structure of a GATT based Profile

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.

5.4 Bluetooth low energy state machine

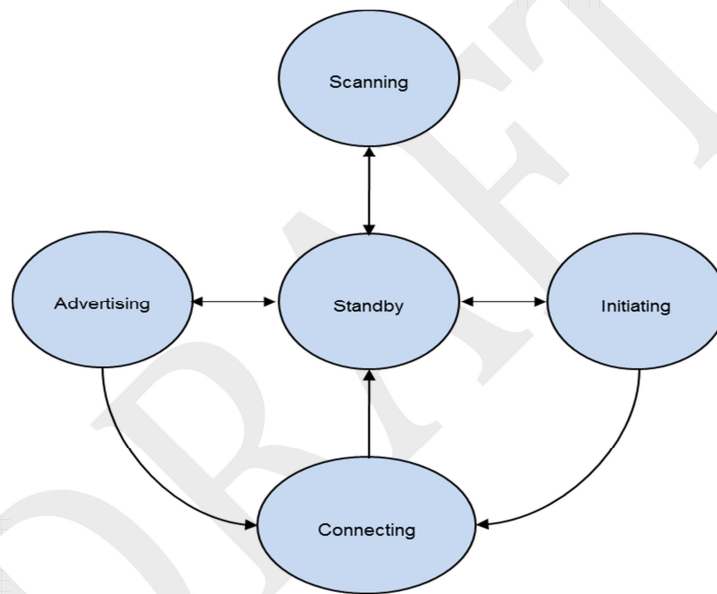


Figure 6 BLE State Machine

The above diagram describes the state machine during BLE operations. Here is 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.

6 Software Overview

6.1 Software Architecture

This chapter describes various software layers which are used by the application software to access and use the BlueNRG stack. These layers are the following:

- STM32Cube HAL Layer
- Board Support Package (BSP) Layer
- BlueNRG HAL Layer
- Application Command Interface (ACI) Layer

The next figure outlines the layering of the software architecture that comprises the STM32 Nucleo and the BlueNRG expansion board:

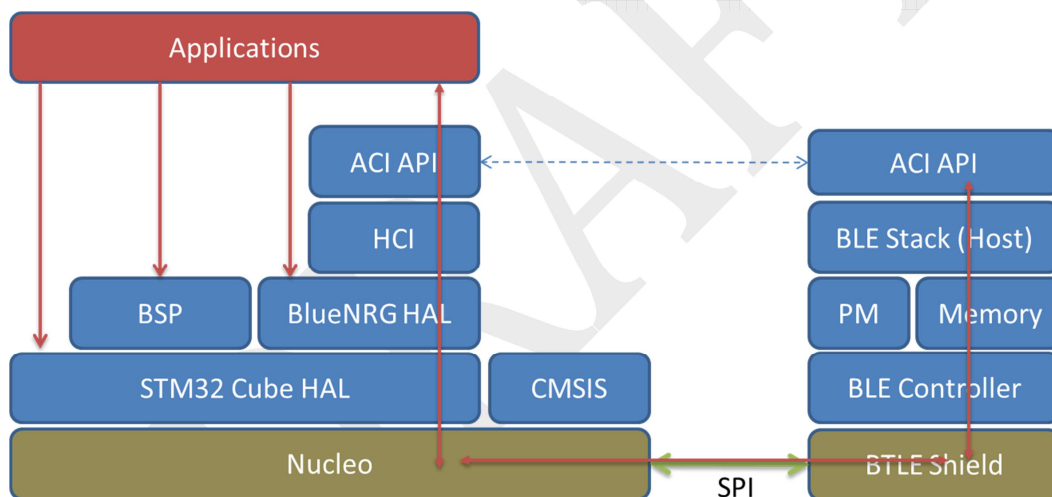


Figure 7 STM32 Nucleo + BlueNRG expansion board Software architecture

6.2 STM32Cube HAL

The STM32Cube HAL is the hardware abstraction layer for the STM32 microcontroller. The STM32Cube HAL ensures maximal portability across STM32 platforms.

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.

For an in-depth understanding of the STM32Cube HAL drivers API, please refer to the document “Description of STM32L0xx HAL drivers”, User manual UM1749, available from st.com at

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00113898.pdf.

6.3 Board Support Package (BSP)

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.

6.4 BlueNRG HAL

The BLE stack is running on the BlueNRG expansion board that is connected to the Nucleo board through the Arduino connectors using the SPI interface. Although the STM32Cube HAL Layer abstracts the SPI interface, there is very little high-level logic applied at the STM32Cube layer. The BlueNRG HAL encapsulates the high-level logic required to interact with the SPI interface and abstracts an API for the HCI to use.

Operations like reading and writing to the SPI interface, enabling and disabling the SPI interrupt, initializing the SPI interface, resetting the BlueNRG expansion board are some of the high-level logic implemented by the BlueNRG HAL layer and presented to the HCI layer as APIs.

6.5 ACI

The Application Command Interface (ACI) layer encapsulates the software logic which uses the Host Controller Interface which in turn uses the BlueNRG HAL layer to access the SPI interface.

The BlueNRG is implemented using a combined Host and Controller solution. An Application Command Interface (ACI) is provided for accessing the BlueNRG Host and Controller. User applications, i.e. programs running on the STM32 host processor, can send ACI commands to control the BlueNRG. The ACI commands are sent over a SPI connection.

For a deeper understanding of the ACI API, please refer the document “DM00114498.pdf” available on st.com at

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00114498.pdf

7 Guide for writing application

This chapter 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 [2] for seeing more details about the ACI API referenced in this chapter.

7.1 Relevant APIs for BlueNRG application

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).

7.1.1 Initialization

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

7.1.1.1 Initializing STM32 Cube HAL

The STM32Cube HAL library needs to 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.

7.1.1.2 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 function 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.

7.1.1.3 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.

7.1.1.4 Initialization and Services characteristics

BlueNRG's stack must be correctly initialized before establishing a connection with another BLE device. This is done with 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.). 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`

7.1.2 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);`

7.1.3 Connectable mode

On 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);`

7.1.4 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:

1. On GATT Server role device the following API should be invoked for updating characteristic value:
 - o `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.
2. On GATT Client device, following API should be invoked for writing to a characteristic handle:
 - o `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.

7.2 SensorDemo application description

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

The IAR project file for the SensorDemo application can be found here:

```
$BASE_DIR\Projects\STM32L053R8-Nucleo\Applications\Bluetooth_LE\SensorDemo\STM32L0xx_EWARM\SensorDemoProject.eww
```

In the SensorDemo 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 7.1 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.

7.3 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
This characteristic exposes the number of seconds passed since system boot. This is a read only characteristic.
- Minutes characteristic
This 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.

7.3.1 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 7.1.1.4 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 [2] 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.

Application Note

```
/**
 * @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);
}
```

Application Note

```
    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`. 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");
```

7.3.2 Update and Notify characteristic value

Time service has “seconds characteristic” as a “readable” characteristic. We must provide support for updating this characteristic 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, value of “minutes characteristics” should also be updated. Minutes_Notify() function as described below performs this operation. This function updates value of “minutes characteristic” exactly once in 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();
```

Application Note

```
/* 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()`.

7.4 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 GATT client application modifies value of this characteristic, LED2 is toggled.

7.4.1 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.

Application Note

```
/*
 * @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 ;
}
```

7.4.2 Obtaining characteristic 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 pkt Pointer to the ACI packet
 * @retval None
 */
void HCI_Event_CB(void *pkt)
{
    hci_uart_pkt *hci_pkt = pkt;
    /* obtain event packet */
    hci_event_pkt *event_pkt = (hci_event_pkt*)hci_pkt->data;

    if(hci_pkt->type != HCI_EVENT_PKT)
        return;
```



```
switch(event_pkt->evt){
    .
    .
    .
case EVT_VENDOR:
{
    evt_blue_aci *blue_evt = (void*)event_pkt->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 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:-

Application Note

```
/**
 * @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);
    }
}
```

The IAR project file for the modified SensorDemo application, in which the new services described in this chapter have been added, can be found here:

\$BASE_DIR\Projects\STM32L053R8-Nucleo\Applications\Bluetooth_LE\
SensorDemo_GATTServer\STM32L0xx_EWARM\SensorDemoProject.eww

8 Testing the sample application

In this chapter the BlueNRG GUI will be used for testing the SensorDemo application developed in the previous chapter. Please download BlueNRG GUI installer provided in STSW-IDB002V1 available [here](#). Detailed instructions regarding its usage can be found in [2]. For testing purpose, hardware components described in section 4.3.1 are needed. Following diagram shows interconnections among these components.

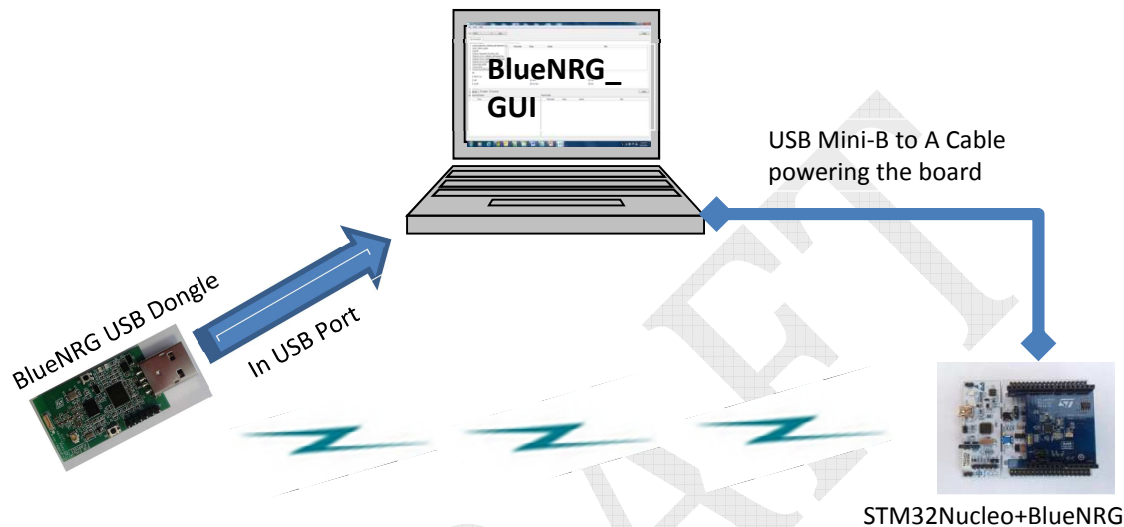


Figure 8 Hardware components setup connected

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.

8.1 Testing SensorDemo 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, PC connected with BlueNRG USB dongle will be configured as “GAP central device”, and STM32Nucleo board equipped with a BlueNRG expansion board is “GAP peripheral device”. Once BlueNRG USB dongle is configured correctly, it can be used to scan remote devices and send ACI commands described in [2]. Various useful operations are described below:

8.1.1 Initializing the USB Dongle

BlueNRG USB dongle must be initialized so that it can communicate with the “GAP peripheral device”. Following commands are used for doing this initialization:

- a. BLUEHCI_HAL_WRITE_CONFIG_DATA
- b. BLUEHCI_GATT_INIT
- c. BLUEHCI_GAP_INIT

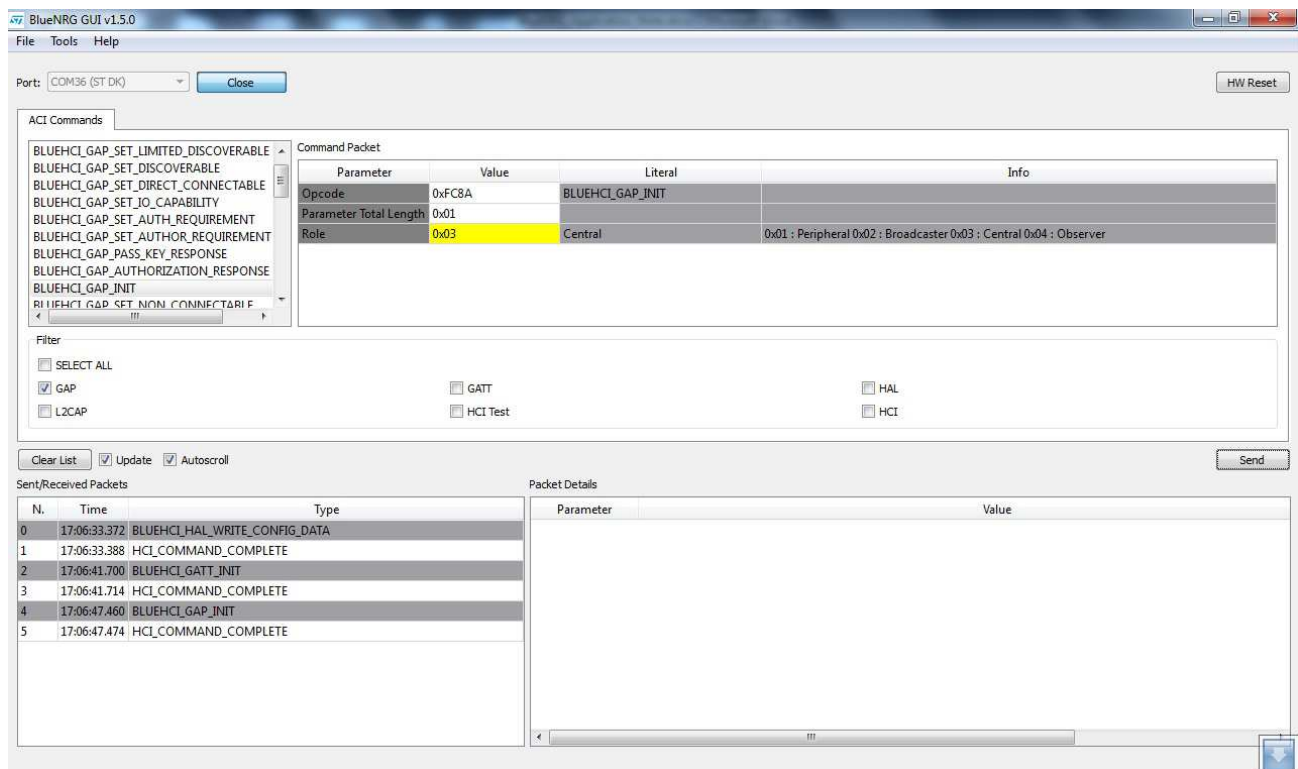


Figure 9: Initializing the BlueNRG Dongle

8.1.2 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.

Application Note

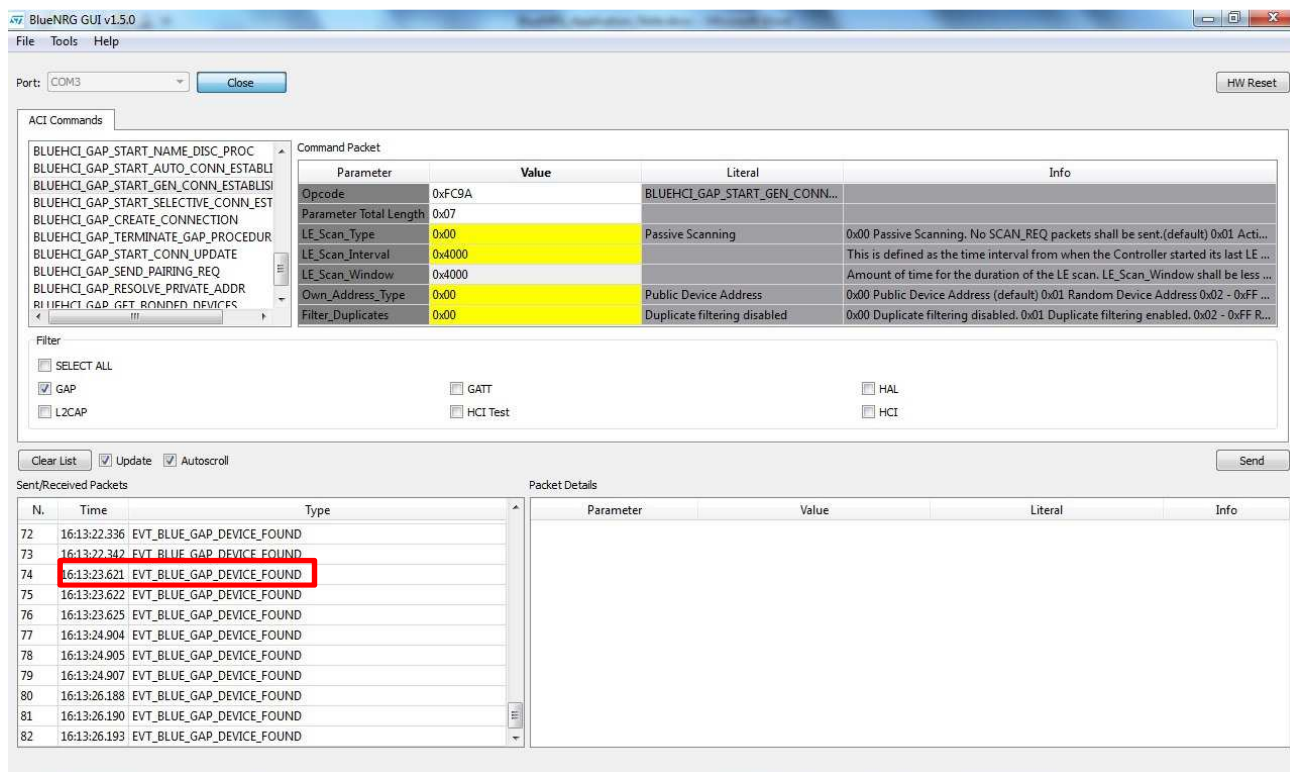


Figure 10: Scanning for devices

8.1.3 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.

Application Note

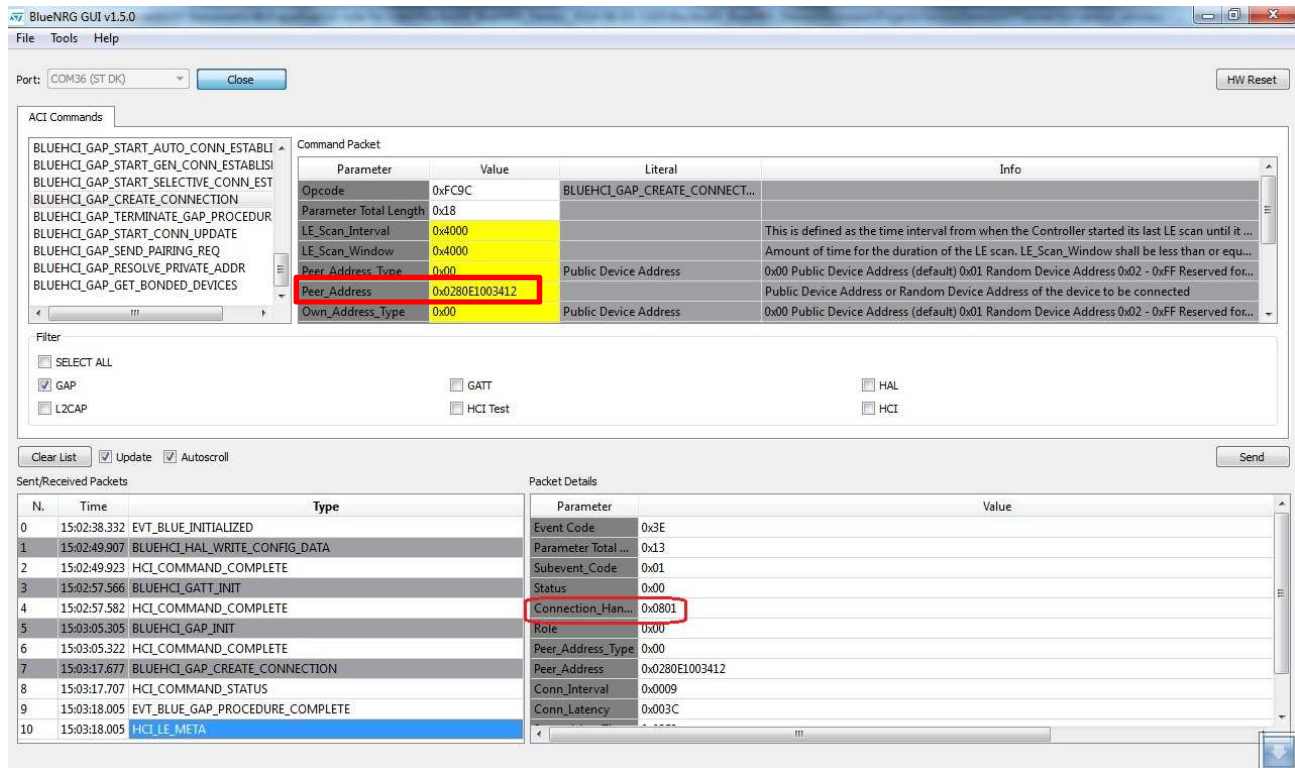


Figure 11: Create Connection

The connection handle can be found out from the HCI_LE_META response of the server. It is indicated in the above figure 11.

8.1.4 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 below in figure 12.

Application Note

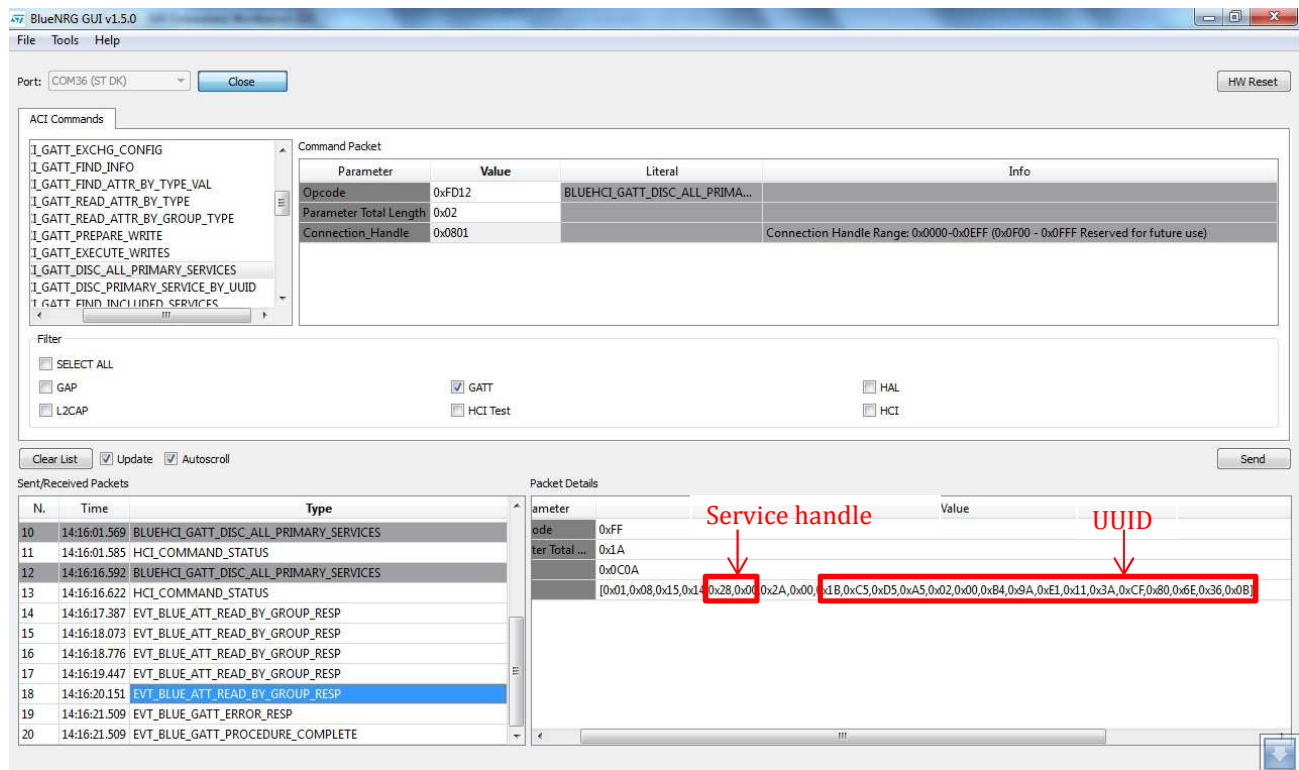


Figure 12 Discover all supported Services

To find the handle of a service the UUID of that particular service (known from given server code) 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.

8.1.5 Get Characteristics supported 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 figure 13.

Application Note

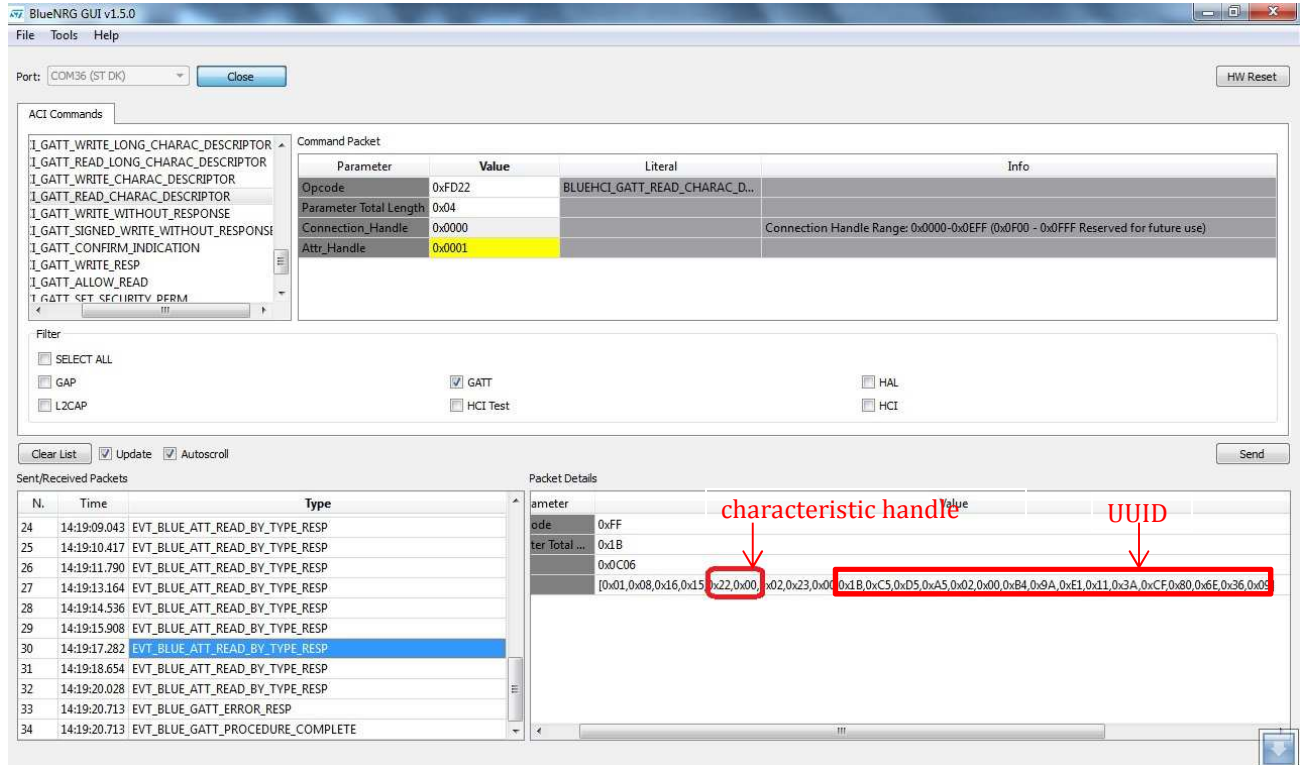


Figure 13: discover all supported characteristics

8.1.6 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 "readable" attribute of the time value. In this case this handle is "0x22+1" or 0x23.

Application Note

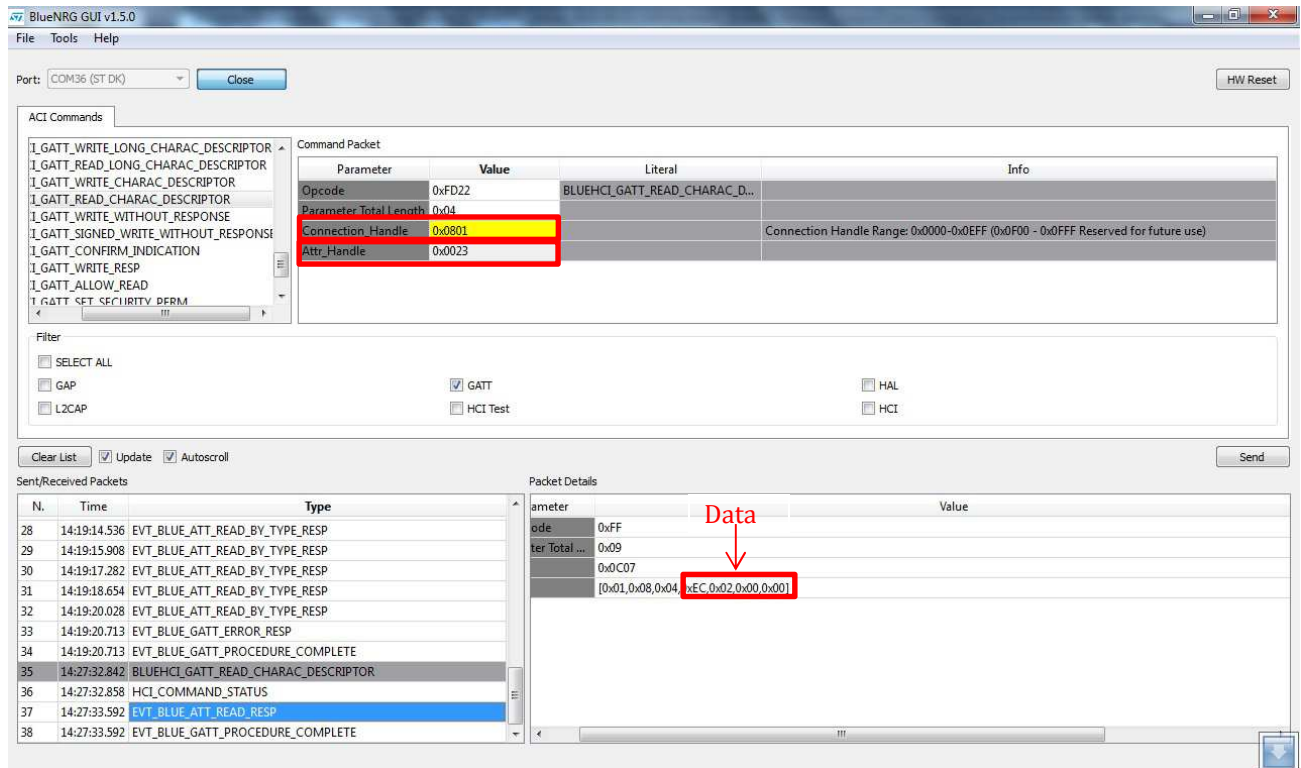


Figure 14: Reading Data from Time Characteristic

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.

8.1.7 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 "writable" attribute of the "LED button" value.

As explained in section 7.3, 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.

Application Note

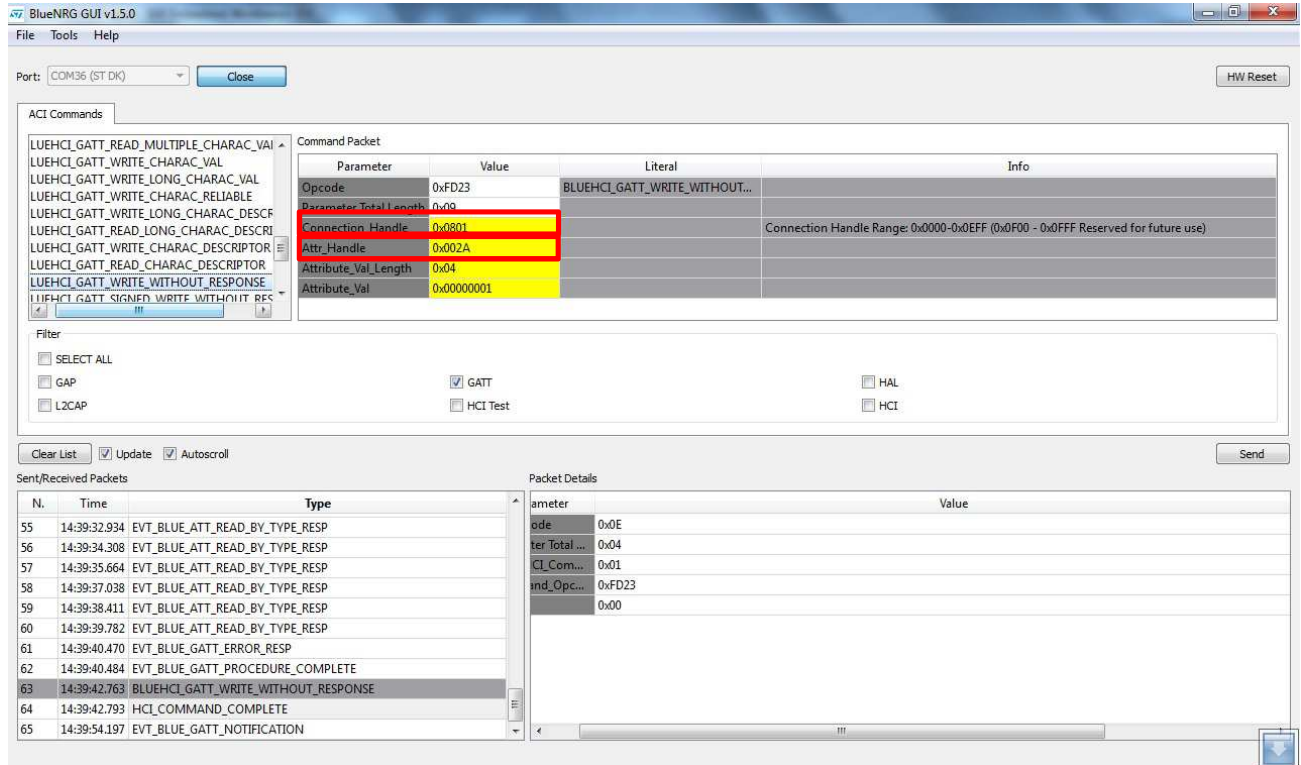


Figure 15: Writing Data to LED button characteristic

8.1.8 Obtain notification for a characteristic

To enable notifications for notifiable characteristics (i.e. characteristics which have CHAR_PROP_NOTIFY property), BLUEHCL_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.

Application Note

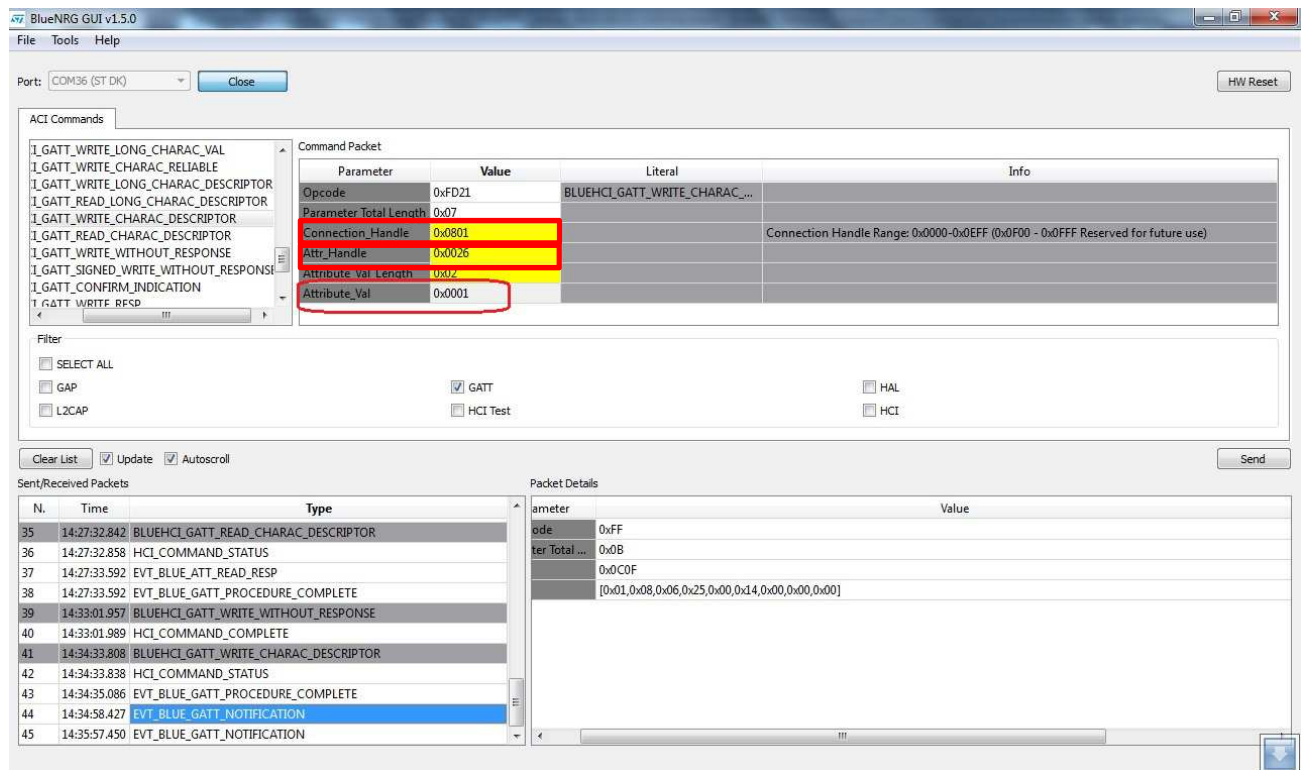


Figure 16: Enabling notifications from server

8.1.9 Disconnecting from remote device

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

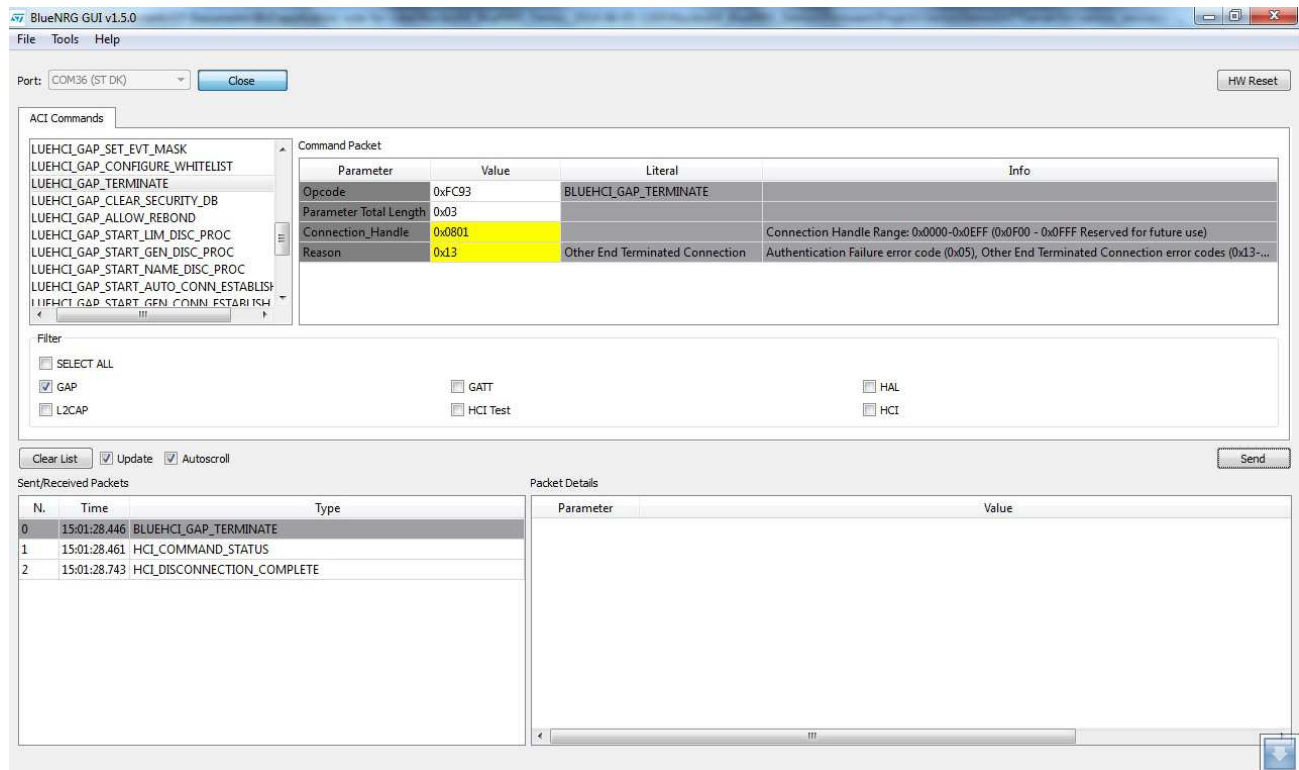


Figure 17: Disconnecting the peripheral device