MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025

Table of contents

1 Middleware **3**
- 1.1 Motor Control . . . 3
  - 1.1.1 FreeMASTER . . . 3
- 1.2 Wireless . . . 40
  - 1.2.1 NXP Wireless Framework and Stacks . . . 40

2 RTOS **765**
- 2.1 FreeRTOS . . . 765
  - 2.1.1 FreeRTOS kernel . . . 765
  - 2.1.2 FreeRTOS drivers . . . 765
  - 2.1.3 backoffalgorithm . . . 765
  - 2.1.4 corehttp . . . 765
  - 2.1.5 corejson . . . 765
  - 2.1.6 coremqtt . . . 766
  - 2.1.7 corepkcs11 . . . 766
  - 2.1.8 freertos-plus-tcp . . . 766

Index **767**

This documentation contains information specific to the lpc845breakout board.

Chapter 1

Middleware

1.1 Motor Control

1.1.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.

- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory

block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
  - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

  - *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
  - *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
  - *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
  - *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
  - *freemaster_serial.h* - defines the low-level character-oriented Serial API.
  - *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
  - *freemaster_can.h* - defines the low-level message-oriented CAN API.
  - *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the "True" single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all *.c files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */
```

(continues on next page)

(continued from previous page)

```
/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*

- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TII	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS.	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P(	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

1.2 Wireless

1.2.1 NXP Wireless Framework and Stacks

NXP Bluetooth LE Host and Sample Applications

Changelog All notable changes to NXP Bluetooth LE Host will be documented in this file.

NXP Bluetooth LE Host Stack is certified **Bluetooth 6.0**

[1.10.13] - mcux v2025-12-00-pvw2

Added

- Threshold for the invalid number of Anchor Monitor events received by the target anchor
- **Experimental Monitoring Advertisers** feature in Bluetooth LE Host
- **Experimental Randomized RPA** feature in Bluetooth LE Host
- Application defines for default connection and default advertising tx power

Improved

- Miscellaneous applications updates
- Central applications now wait for status of Encrypt procedure in case of bonded device
- Logging data on localization applications
- NBU Low Power Mode enabled by default for Localization Applications
- PCT rotation calibration added to localization apps
- Configured CS Reflector to start the CS procedure with the tdm command and updated the documentation
- Populated the optionalSubfeaturesSupported field correctly
- Prevented CORE 0 from entering deep sleep while LCE is computing by setting the low power mode constraint to PWR_WFI during LCE computation and releasing it afterwards
- Implemented in CCC_CS, Channel Sounding data transfer from the anchor to the device
- Documentation miscellaneous updates
- Updated all kw47 and kw45 armgcc ld linker files to take gUseInternalStorageLink_d flag value into consideration

Fixed

- Memory issue when setting scan response data would return an error status from the LL
- Set advertises with the public address, overwritten by a previously used random address on receiving the Advertising Set Terminated event

Changed

- Updated memory configuration: replaced the extended heap area in the available SMU2 memory with a 24KB array in the data1
- Removed redundant cached remote capabilities write on reflectors, as the initiator will always trigger a capabilities exchange and trying to write cached capabilities afterwards results in an HCI error

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.12] - mcux v2025-12-00-pvw1

Added

- **Gap_LoadCustomBondedDeviceInformation API** to retrieve custom peer information using NVM index

Improved

- Miscellaneous application updates

Fixed

- **Updated privacy timeout** mechanism affected by LE Set Extended Advertising Enable Command

Changed

- Merged **Gap_SetPeriodicAdvParameters** and Gap_SetPeriodicAdvParametersV2 into Gap_SetPeriodicAdvParameters

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.11] - mcux v2025-09-00

Added

- CCC sample applications updated to **CCC Digital Key v4.0.0** Specification
- **RAS/RAP PTS** 8.7.4 test support added in Localization Sample applications
- Support for CS start procedure while the previous procedure is not completed; old procedure replaced with the new one
- Support for arm gcc for ncp_loc_reader__cm33_core1

Improved

- Localization Sample Applications Ram partition
- **RAS/RAP** profile and service
- Various sample applications have been updated

Fixed

- Privacy setting issue on ncp_loc_reader
- Extended NBU FSCI message handling issue
- Ble_shell updated to set the Random Static Address properly
- Always set the Advertising Legacy Set handle if the legacy API was used
- fsci_bridge and w_uart_host memory leak
- PAWR parameters in PeriodicSyncTransferReceived are now parsed correctly
- Ensure an RPA/NRPA is properly set from the application to enable a central using Controller Privacy to connect to unbonded peripherals

- CS algorithm buffer overwrite issue during Connection Handover application
- Various sample applications bug fixes applied
- Pass correct Codded PHY (S2) to Channel Sounding Set Procedure Parameters

Changed

- Merged Gap_SetExtAdvertisingParameters and Gap_SetExtAdvertisingParametersV2 into **Gap_SetExtAdvertisingParameters**
- ce_status_buffer type changed to int32_t
- BLE_Shell prints Random Static address as identity address instead of the Public Device Address

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.10] - mcux v2025-09-00-pvw2

Improved

- **CS Event Handling**: CS (Channel Sounding) events are now sent to the application task for processing, rather than being handled directly in the Host task
- Various sample applications have been updated

Fixed

- Bluetooth Advertising Sets: Now supports **4 advertising** sets in the Bluetooth host libraries
- Various sample applications bug fixes applied

Changed

- Bluetooth Address Type: The default address type has been changed from **Public to Random Static**

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.9] - mcux v2025-09-00-pvw1

Added

- **IDS event** trigger when incoming ATT packets exceed agreed MTU
- **IDS event** trigger when Unexpected SMP Messages received in idle states (before pairing starts)
- Average RSSI reporting in Anchor Monitor event
- Support for gAppDeferAlgoRun_d in btcs_client.c
- **Multi-connection monitoring** in Handover/Monitor Mode

Improved

- Updated **NBU channel sounding** applications to support **64MHz** clock speed
- Cleanup of commComplete structures that only contain status from hci_interface.h

Fixed

- CCC Application handover monitoring RSSI issue
- Intrusion Detection System not reporting event type
- Extended **NBU armgcc projects** stability
- Advertising Extended Applications when Gap_PeriodicAdvCreateSync fails

Changed

- Updated digital_key_car_anchor applications to configure coding scheme via Host API
- Enhanced RAS handling of ACK Ranging Data in invalid conditions

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.8] - mcux v2025-06-00

Added

- **Gap_SetBondedDeviceName()** to set device name using NVM index
- **RAS** queue for GATT indications sent
- **gHciStatusBase_c** to **csError** status
- Option to use statically allocated **memory** for dynamic **GATT database** (prevents heap fragmentation)
- Checks for **controller** supported features and setting **PAST bits** accordingly
- **Anchor** support to **export** device data via RAS using gAppHciDataLogExport_d = 2
- **Anchor** support to **export** device local HCI data using gAppHciDataLogExport_d = 1
- **Shell commands** to list peer devices and trigger connection handover
- Define for **enabling** optional **CCC** LE Coded **PHY** advertising
- **cs_sync_phy** parameter to mDefaultRangeSettings (**renamed** from outdated RTTPhy)

Improved

- **Stack Host** now saves the most recently set **random address** after successful controller response
- Miscellaneous **minor** application **updates**

Fixed

- Compilation issue in **loc_reader app** with real-time RAS transfer
- **CCC** application **handover state machine** race condition
- **CCC** resets **gCurrentAdvHandle** upon connection
- **RAS** uses correct bit for data overwrite preference

Changed

- Updated **Bluetooth LE Host Documentation**.
- **BLE_Shell** Tx timer interval adjusted for **max throughput** on 1M PHY
- **CS_ConfigVendorCommand** updated with **Inline Phase Return** field
- Renamed **tx_pwr_phy** to **phy** and removed obsolete rtt_phy field
- Updated **documentation** to clarify **Controller Privacy** restrictions

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.7] - mcux v2025-06-00-pvw2

Added

- Support for **Bluetooth LE Debug Keys**
- Support for **pairing failure** reason 0x10 (Busy)
- Extended NBU **Wireless_uart_host battery service**
- **Channel Sounding RSSI Information** at application level
- Application support for **4-antenna configuration**

Improved

- **Clean** include directives
- Allow central-only devices to use **Gap_EncryptAdvertisingData**
- **CSTACK** size optimization for **RADE** using baremetal apps

Fixed

- Extended NBU **memory leak** issues
- Issues when **gAppOtaASyncFlashTransactions_c** is set to 0
- Issue with **CS procedure** affected by the CS data export
- Extended NBU **Wireless_uart_host privacy**

Changed

- Updated **Bluetooth LE Host Documentation**.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.6] - mcux v2025-06-00-pvw1

Added

- **Encypted Advertising Data** support in Extendded Advertising Applications.
- Support for **disable UART** for **CS applications** for **low power measurements**.
- Support for **LCE (DSPV) non-blocking API** integration to **RADE**.
- **Intrusion Detection System** as **Experimental**.

Improved

- **L2CAP command length validation** to cover all signaling commands.
- **Extended NBU Wireless_uart_host multiple connections** support.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.5] - mcux v2025-03-00

Added

- MCXW72 **Extended NBU** support and **w_uart_host**, **fsci_bridge** and **ncp_fscibb** sample applications.
- **L2CAP support** for Channel Sounding **IQ Sample Transfer** in CCC CS sample applications.
- Bluetooth LE Sample applications for **MCX-W71-EVK** board.

Changed

- Updated **FSCI XML file**.
- Updated **Bluetooth LE Host Documentation**.

Fixed

- Cleared the **mpRemoteCachedCaps** entry when the peer disconnects (CS sample applications).
- Transfer **RAS subscription** data during connection handover (CCC CS sample applications).
- **EAD** - Updated advertising data length check to ensure encrypted data fits inside one AD.
- Updated **digital_key_car_anchor** and **digital_key_car_anchor_cs** applications to manage **Random Static address** from the application layer.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.4] - mcux v2025-03-00-pvw2

Added

- PAWR support in **BLE Shell** sample application.
- PAWR support in **adv_ext_peripheral** and **adv_ext_central** sample applications.
- New sample applications for **FRDM-MCXW72**.
- **Gap_SetScanningCallback** API.
- Support for handover connection interval update command.

Changed

- Updated HID Device for **Windows 11** compatibility.
- Updated CCC demos to **Digital Key R4 spec version 1.0.0**.
- Improved RPA resolution at the **Host level**, now performed synchronously.
- Enhanced parsing of the **CS procedure** in Ranging Service.

Fixed

- Corrected parsing of the **PAST command** in FSCI GAP.
- Fixed **scan event reporting** in PAST scenario.
- Added an error case for Gap_SetChannelMap in the generic event handler.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

Bluetooth Low Energy Application Developer's Guide

Introduction This document explains how to integrate the NXP Bluetooth Low Energy Host Stack in an application and provides detailed explanation of the most commonly used APIs and code examples.

- *Introduction*: This section outlines the document structure.
- *Prerequisites*: The document sets out the prerequisites.
- *Bluetooth LE Host Stack Initialization and APIs*: This section describes the Bluetooth Low Energy Host Stack initialization. It also presents the APIs categorized according to the layer and by application role.
- *Generic Access Profile (GAP) Layer*: The Generic Access Profile (GAP) layer is divided into two sections according to the GAP role of the device: Central and Peripheral. The basic setup of two such devices is explained with code examples, such as how to prepare the devices for connections, how to connect them together, and pairing and bonding processes.
- *Generic Attribute Profile (GATT) Layer*: This section describes the Generic Attribute Profile (GATT) layer and introduces the APIs required for data transfer between the two connected devices. This section is divided into two subsections according to the GATT role of the device: Client and Server.

- *GATT database application interface*: The document further describes the usage of the GATT database APIs that allow the application to manipulate data stored in the GATT Server database.
- *Creating GATT database*: This section describes a user-friendly method to build a GATT database statically. The method involves the use of a predefined set of macros that the application can include to build the database at application compile time.
- *Creating a Custom Profile*: This section contains instructions on how to build a custom profile.
- *Application Structure*: The section describes the structure of the typical application.
- *Low-Power Management*: This section describes low-power management and how an application can use the low-power modes of the hardware and software.
- *Over the Air Programming (OTAP)*: This section describes the Over The Air Programming (OTAP) capabilities that the Host Stack offers via a dedicated Service/Profile. The section also describes how to use the OTAP capabilities in an application and also contains a detailed description of the SDK components involved in the OTAP process.
- *Creating a Bluetooth LE application when the Host Stack runs on another processor*: This section describes how to build a Bluetooth Low Energy application when the Host Stack is running on a separate processor.
- *References*: This section lists the documents that can be referred to for more information.
- *Acronyms and abbreviations*: This section lists the acronyms used in this document.

Prerequisites The Bluetooth Low Energy Host Stack library contains several external references that the application must define to enable full functionality.

Attention: Application developers must ensure to define these references to prevent linkage errors when trying to build the application binary.

Task and event queues The task queues are declared in the *ble_host_tasks.h* as follows:

```
/*! App to Host message queue for the Host Task */
   extern messaging_t   gApp2Host_TaskQueue;
/*! HCI to Host message queue for the Host Task */
   extern messaging_t   gHci2Host_TaskQueue;
/*! Event for the Host Task Queue */
   extern OSA_EVENT_HANDLE_DEFINE(gHost_TaskEvent);
```

See Initialization for more details about the RTOS tasks required by the Bluetooth LE Host Stack.

Parent topic:*Prerequisites*

GATT database The application must define and populate the database according to its requirements and constraints either statically, at application compile time, or dynamically.

Regardless of how the application creates the GATT database, the following two external references from *gatt_database.h* must be defined:

```
/*! The number of attributes in the GATT Database. */
extern uint16_t gGattDbAttributeCount_c;
/*! Reference to the GATT database */
extern gattDbAttribute_t* gattDatabase;
```

The attribute template is defined as shown here:

```
typedef struct{
    uint16_t handle ;
/*!< Attribute handle - cannot be 0x0000; attribute handles need not be consecutive, but must be strictly␣
↪increasing. */
    uint16_t permissions ;
/*!< Attribute permissions as defined by ATT. */
    uint32_t uuid ;
/*!< The UUID should be read according to the gattDbAttribute_t.uuidType member: for 2-byte and 4-byte␣
↪UUIDs, this contains the value of the UUID; for 16-byte UUIDs, this is a pointer to the allocated 16-byte␣
↪array containing the UUID. */
    uint8_t * pValue ;
/*!< Pointer to allocated value array. */
    uint16_t valueLength ;
/*!< Size of the value array. */
    uint16_t uuidType : 2;
/*!< Identifies the length of the UUID; the 2-bit values are interpreted according to the bleUuidType_t␣
↪enumeration. */
    uint16_t maxVariableValueLength : 10;
/*!< Maximum length of the attribute value array; if this is set to 0, then the attribute's length␣
↪(valueLength) is fixed and cannot be changed. */
} gattDbAttribute_t ;
```

Parent topic:*Prerequisites*

Non-Volatile Memory (NVM) access The Bluetooth LE Host Stack implements an internal module responsible for managing device information. This module relies on accessing a Non-Volatile Memory module for storing and loading bonded devices data.

The application developers determine the NVM access mechanism through the definition of three functions and one variable. The functions must first pre-process the information and then perform standard NVM operations (erase, write, read). The declarations are as follows:

```
bleResult_t App_NvmErase
(
    uint8_t mEntryIdx
);
bleResult_t App_NvmRead
(
    uint8_t  mEntryIdx,
    void*    pBondHeader,
    void*    pBondDataDynamic,
    void*    pBondDataStatic,
    void*    pBondDataLegacy,
    void*    pBondDataDeviceInfo,
    void*    pBondDataDescriptor,
    uint8_t  mDescriptorIndex
);
bleResult_t App_NvmWrite
(
    uint8_t  mEntryIdx,
    void*    pBondHeader,
    void*    pBondDataDynamic,
    void*    pBondDataStatic,
    void*    pBondDataLegacy,
    void*    pBondDataDeviceInfo,
    void*    pBondDataDescriptor,
    uint8_t  mDescriptorIndex
);
```

The device information is divided into several components to ensure that even software wear leveling mechanisms can be used optimally. The components sizes are fixed (defined in

ble_constants.h) and have the following meaning:

API pointer to bond component	Component size (*ble_constants.h*)	Description
pBondHeader: points to a bleBondIdentityHeaderBlob_t element	gBleBondIdentityHeaderSize_c	Bonding information which is sufficient to identify a bonded device.
pBondDataDynamic: points to a bleBondDataDynamicBlob_t element	gBleBondDataDynamicSize_c	Bonding information that might change frequently.
pBondDataStatic: points to a bleBondDataStaticBlob_t element	gBleBondDataStaticSize_c	Bonding information that is unlikely to change frequently.
pBondDataLegacy: points to a bleBondDataLegacyBlob_t element	gBleBondDataLegacySize_c	Stores legacy pairing and Connection Signature Resolving Key (CSRK) bond information.
pBondDataDeviceInfo: points to a bleBondDataDeviceInfoBlob_t element	gBleBondDataDeviceInfoSize_c	Additional bonding information that can be accessed using the host stack API.
pBondDataDescriptor: points to a bleBondDataDescriptorBlob_t element	gBleBondDataDescriptorSize_c	Bonding information used to store one Client Characteristic Configuration Descriptor (CCCD).

The Bluetooth LE Host Stack handles the format of the bonding information. Therefore, application developers need not to take care of this aspect.

Each bonding data slot must contain one bonding header blob, one dynamic data blob, one static data blob, one data legacy blob, one device information blob, and an array of descriptor blobs equal to *gcGapMaximumSavedCccds_c.*

Note: *The application must define the* gcGapMaximumSavedCccds_c. *macro according to its requirement. The default value can be found in the* ble_constants.h *file.*)

A slot is uniquely identified by the *mEntryIdx* parameter.

A descriptor is uniquely identified by the pair *mEntryIdx - mDescriptorIndex*.

If one or more pointers passed as parameters are NULL, the read from or write to the corresponding blob of the bonding slot must be ignored. The erase function must clear the entire bonding data slot specified by the entry index.

Note: When Advanced Secure Mode is chosen (gAppSecureMode_d is defined as 1 in app_preinclude.h), two additional application NVM functions are defined to handle local keys encrypted blob storage. Their declaration is:

```
bleResult_t App_NvmWriteLocalKeys
(
    uint8_t mEntryIdx,
    void* pLocalKey
)
bleResult_t App_NvmReadLocalKeys
(
    uint8_t mEntryIdx,
    void* pLocalKey
)
```

The functions write/read a structure of type bleLocalKeysBlob_t into/from NVM using a dedicated data set. The parameter mEntryIdx can be 0 (local IRK is handled) or 1 (local CSRK is handled).

The format of the local keys blob nor about the generation and storage of the local keys is automatically handled in **BLE Connection Manager** (*BleConnManager_GenericEvent*). Therefore the application developer need not manage this aspect.

The current implementation of the aforementioned functions uses either the framework NVM module or a RAM buffer. Additional details about the NVM configuration and functionality can be found in the *Connectivity Framework Reference Manual*. See *References*.

To enable the NVM mechanism, ensure the following points:

- gAppUseNvm_d (in *app_preinclude.h*) is set to 1 and
- gUseNVMLink_d is set to 1 in the linker options of the toolchain.

Note:

- If gAppUseNvm_d is set to 0, then all bonding data is stored in the RAM and is accessible until reset or power cycle.
- If gAppUseNvm_d is set to 1, the default NVM module configurations are applied in the *app_preinclude.h* file.

Parent topic:*Prerequisites*

Bluetooth LE Host Stack Initialization and APIs

Initialization The application developer is required to configure the Host Task as part of the Host Stack requirement. The task is the context for running all the Host layers (GAP, GATT, ATT, L2CAP, SM, GATTDB)

The prototype of the task function is located in the *ble_host_tasks.h* file:

```
void Host_TaskHandler(void * args);
```

It should be called with *NULL* as an argument in the task code from the application.

Application developers are required to define task events and queues as explained in RTOS Task Queues and Events.

If the Controller software runs on the same chip as the Host, the Controller task always has a higher priority than the Host task. The priority value of the Host Task can be configured through the *gHost_TaskPriority_c define* (by default set in *ble_host_task_config.h*). Note that changing this value can have a significant impact on the Bluetooth Low Energy stack.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Main function to initialize the Bluetooth LE Host Stack This Figure provides an overview of Bluetooth Low Energy Host Stack. When using the existing application common files, the startup task uses *BluetoothLEHost_AppInit()*, which is defined in *app_conn.h*. The function initializes all components related to the Bluetooth Low Energy application. It has the following prototype:

```
void BluetoothLEHost_AppInit(void);
```

The *BluetoothLEHost_AppInit()* function must be implemented by each application. It should register its generic event callback using *BluetoothLEHost_SetGenericCallback()* and initialize the Bluetooth LE Host Stack layer by calling *BluetoothLEHost_Init()*. The prototype for the *BluetoothLEHost_Init()* function is found in *app_conn.h* and is implemented in *app_conn.c*.

```
void BluetoothLEHost_Init
(
    appBluetoothLEInitCompleteCallback_t pCallback
);
```

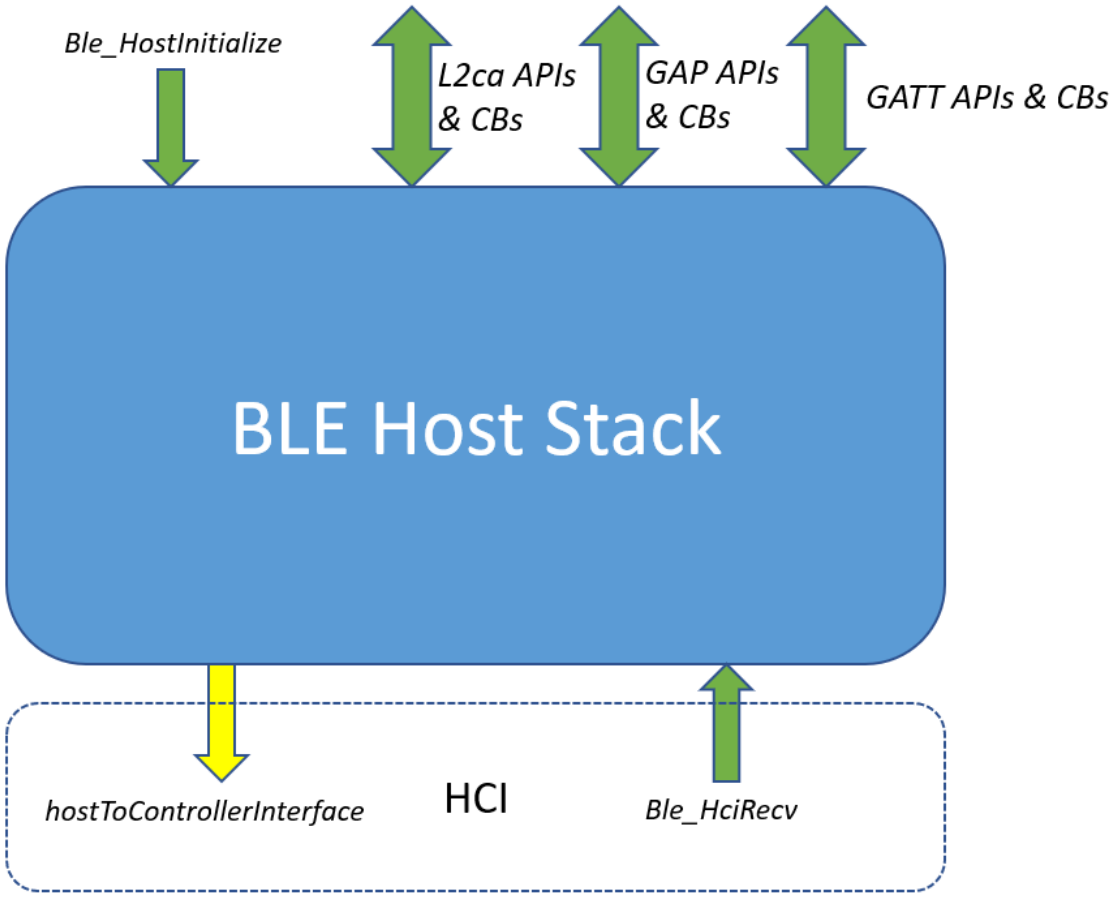
BluetoothLEHost_Init takes as parameter a function to be called at the end of Bluetooth LE Host Stack initialization. In this callback, the application can register its callbacks with the Host layer, allocate timers, start services, and perform similar tasks. The callback should have a prototype as follows:

```
static void BluetoothLEHost_Initialized(void);
```

BluetoothLEHost_Init() is responsible for initializing the Host.

Initialize the Bluetooth LE Host Stack after platform setup is complete and all RTOS tasks have been started. The function that should be called for this purpose is located in the *ble_general.h* file and has the following prototype:

```
bleResult_t Ble_HostInitialize
(
    gapGenericCallback_t            genericCallback,
    hciHostToControllerInterface_t  hostToControllerInterface
);
```



Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

HCI entry and exit points The HCI entry point of the Host Stack is the second function located in the *ble_general.h* file:

```
void Ble_HciRecv
(
    hciPacketType_t packetType,
```

(continues on next page)

(continued from previous page)

```
    void* pHciPacket,
    uint16_t packetSize
);
```

This is the function that the application must call to insert an HCI message into the Host.

An equivalent exists, to be used in ISR context:

```
bleResult_t Ble_HciRecvFromIsr
(
    hciPacketType_t     packetType,
    void*               pHciPacket,
    uint16_t            packetSize
);
```

Therefore, the *Ble_HciRecv* function and the *hostToControllerInterface* parameter of the *Ble_HostInitialize* function represent the two points that need to be connected to the LE Controller (see Bluetooth Low Energy Host Stack overview), either directly (if the Controller software runs on the same chip as the Host) or through a physical interface (for example, UART).

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Bluetooth LE Host Stack libraries and API availability All the APIs referenced in this document are available in the Central and Peripheral libraries. The support for Bluetooth 5.3 optional features such as Advertising, Advertising Extensions, GATT Caching, and EATT are provided in separate host libraries. They are distributed in a similar process as the legacy ones using GAP/GATT role support, which is described as follows.

For example, below are listed the full-featured libraries with complete support for both Central and Peripheral APIs, at GAP level.

- *lib_ble_OPT_host_cm33_iar.a* (for IAR projects)
- *lib_ble_OPT_host_cm33_gcc.a* (for MCUX projects)

These libraries include optional features implemented by the Bluetooth LE Host. For applications that need to use only the mandatory 5.3 Bluetooth LE and below features, the *lib_ble_host_cm33_iar.a* or *lib_ble_host_cm33_gcc.a* libraries can be used instead.

However, some applications may be targeted to memory-constrained devices and do not need the full support. In the interest of reducing code size and RAM utilization, optimized libraries are provided:

- *lib_ble_host_peripheral_cm33_iar.a/ lib_ble_host_peripheral_cm33_gcc.a* and
- *lib_ble_OPT_host_peripheral_cm33_iar.a / lib_ble_OPT_host_peripheral_cm33_gcc.a.*
  - Support only APIs for the GAP Peripheral and GAP Broadcaster roles
  - Support only APIs for the GATT Server role
- *lib_ble_host_central_cm33_iar.a* and *lib_ble_OPT_host_central_cm33_iar.a*
- *lib_ble_host_central_cm33_gcc.a* and *lib_ble_OPT_host_central_cm33_gcc.a*
  - Support only APIs for the GAP Central and GAP Observer roles
  - Support only APIs for the GATT Client role

If one attempts to use an API that is not supported (for instance, calling *Gap_Connect* with the *lib_ble_host_peripheral_cm33_iar.a* and *lib_ble_host_peripheral_cm33_gcc.a*), then the API returns the *gBleFeatureNotSupported_c* error code.

Similarly, if the API for OPT is used with a host library that does not have support for optional features, then *gBleFeatureNotSupported_c* is returned. For instance, calling *Gap_SetExtAdvertising*

parameters with the *lib_ble_host_peripheral_cm33_iar.a* and *lib_ble_host_peripheral_cm33_gcc.a*) returns the exit code *gBleFeatureNotSupported_c*.

Note: See the *Bluetooth Low Energy Host Stack API Reference Manual* for explicit information regarding API support. Each function documentation contains this information in the Remarks section.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Synchronous and asynchronous functions The vast majority of the GAP and GATT APIs are executed **asynchronously**. Calling these functions generates a message and places it in the Host Task message queue.

Therefore, the actual result of these APIs is signaled in **events** triggered by specific callbacks installed by the application. See the *Bluetooth Low Energy Host Stack API Reference Manual* for specific information about the events that are triggered by each API.

However, there are a few APIs which are executed immediately (**synchronously**). This is explicitly mentioned in the *Bluetooth Low Energy Host Stack API Reference Manual* in the *Remarks* section of each function documentation.

If nothing is mentioned, then the API is asynchronous.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Radio TX Power level The controller interface includes APIs that can be used to set the Radio TX Power to a different level than the default one.

The power level can be set differently for advertising and connection channels by calling the function *Controller_SetTxPowerLevelDbm()* with the channel parameter set to *gAdvTxChannel_c* or *gConnTxChannel_c*.

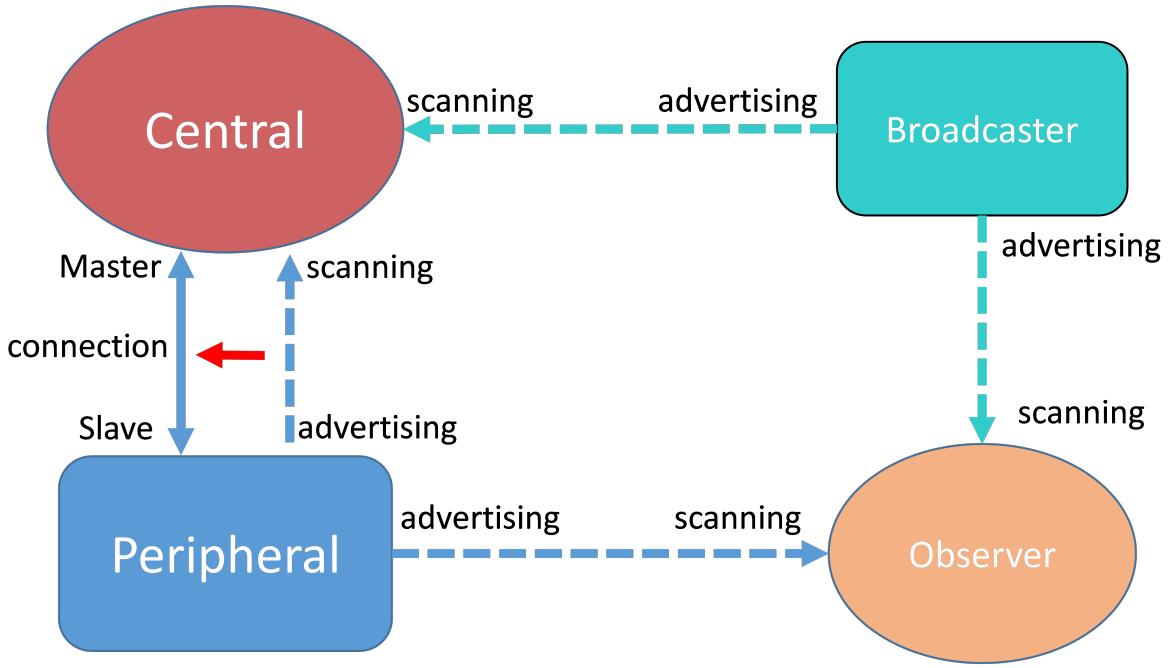
Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Generic Access Profile (GAP) Layer The GAP layer manages connections, security, and bonded devices.

The GAP layer APIs are built on top of the Host-Controller Interface (HCI), the Security Manager Protocol (SMP), and the Device database.

GAP defines four possible roles that a Bluetooth Low Energy device may have in a Bluetooth Low Energy system:

- **Central**
  - Scans for advertisers (Peripherals and Broadcasters)
  - Initiates connection to Peripherals; Central at Link Layer (LL) level
  - Usually acts as a GATT Client, but can also contain a GATT Database itself
- **Peripheral**
  - Advertises and accepts connection requests from Central devices; LL Peripheral
  - Usually contains a GATT Database and acts as a GATT Server, but may also be a Client
- **Observer**
  - Scans for advertisers, but does not initiate connections; Transmit is optional
- **Broadcaster**
  - Advertises, but does not accept connection requests from Central devices; Receive is optional



The Figure illustrates the generic GAP topology.

Peripheral setup The Peripheral starts advertising and waits for scan and connection requests from other Central devices.

Advertising Before starting advertising, the advertising parameters should be configured. Otherwise, the following defaults are used.

```
#define gGapDefaultAdvertisingParameters_d \
{
   /* minInterval */      gGapAdvertisingIntervalDefault_c, \
   /* maxInterval */      gGapAdvertisingIntervalDefault_c, \
   /* advertisingType */ gConnectableUndirectedAdv_c, \
   /* addressType */      gBleAddrTypePublic_c, \
   /* peerAddressType */  gBleAddrTypePublic_c, \
   /* peerAddress */      {0U, 0U, 0U, 0U, 0U, 0U}, \
   /* channelMap */       (gapAdvertisingChannelMapFlags_t)gGapAdvertisingChannelMapDefault_c,  \
   /* filterPolicy */    gProcessAll_c \
}
```

To set different advertising parameters, a *gapAdvertisingParameters_t* structure should be allocated and initialized with defaults. Then, the necessary fields may be modified.

After that, the following function should be called:

```
bleResult_t Gap_SetAdvertisingParameters
(
    const gapAdvertisingParameters_t *      pAdvertisingParameters
);
```

The application should listen to the *gAdvertisingParametersSetupComplete_c* generic event.

Next, the advertising data should be configured and, if the advertising type supports active scanning, the scan response data should also be configured. If either of these is not configured, they are defaulted to empty data.

The function used to configure the advertising and/or scan response data is shown here:

```
bleResult_t Gap_SetAdvertisingData
(
  const gapAdvertisingData_t *      pAdvertisingData,
  const gapScanResponseData_t *     pScanResponseData
);
```

Either of the two pointers may be *NULL*, in which case they are ignored (the corresponding data is left as it was previously configured, or empty if it has never been set), but not both at the same time.

The application should listen to the *gAdvertisingDataSetupComplete_c* generic event.

After all the necessary setup is done, advertising may be started with this function:

```
bleResult_t Gap_StartAdvertising
(
    gapAdvertisingCallback_t advertisingCallback,
    gapConnectionCallback_t connectionCallback
);
```

The advertising callback is used to receive advertising events (advertising state changed or advertising command failed), while the connection callback is only used if a connection is established during advertising.

The connection callback is the same as the callback used by the Central when calling the *Gap_Connect* function.

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartAdvertising
(
    appAdvertisingParams_t    *pAdvParams,
    gapAdvertisingCallback_t pfAdvertisingCallback,
    gapConnectionCallback_t   pfConnectionCallback
);
```

The API goes through the steps of setting the advertising data and parameters. Events from the Host task are treated in the *App_AdvertiserHandler()* function, implemented in *app_advertiser.c*. To set the advertising parameters and data *BluetoothLEHost_StartAdvertising* requires a parameter of the following type:

```
typedef struct
appAdvertisingParams_tag
{
 gapAdvertisingParameters_t *pGapAdvParams; /*!< Pointer to the GAP advertising parameters */
 const gapAdvertisingData_t *pGapAdvData; /*!< Pointer to the GAP advertising data */
 const gapScanResponseData_t *pScanResponseData;
 /*!< Pointer to the scan response data */ } appAdvertisingParams_t;
```

If a Central initiates a connection to this Peripheral, the *gConnEvtConnected_c* connection event is triggered.

To stop advertising while the Peripheral has not yet received any connection requests, use this function:

```
bleResult_t Gap_StopAdvertising (void);
```

This function should not be called after the Peripheral enters a connection, as the advertising automatically stops in this case.

Parent topic: Peripheral setup

Pairing and bonding (peripheral) After a connection has been established to a Central, the Peripheral's role regarding security is a passive one. It is the responsibility of the Central device to start the pairing process. In case, the devices have already bonded in the past, the Central encrypts the link using the shared LTK.

The Peripheral sends error responses (at ATT level) with proper error code if the Central attempts to access sensitive data without authenticating. Examples of such error responses are: Insufficient Authentication, Insufficient Encryption, Insufficient Authorization, and so on. Therefore, it indicates to the Central that it needs to perform security procedures.

All security checks are performed internally by the GAP module and the security error responses are sent automatically. All the application developer needs to do is register the security requirements.

First, when building the GATT Database (see *Creating GATT database*), the sensitive attributes should have the security built into their access permissions (for example, read-only / read with authentication / write with authentication / write with authorization, and so on.).

Second, if the GATT Database requires additional security besides that already specified in attribute permissions (for example, certain services require higher security in certain situations), the following function must be called:

```
bleResult_t Gap_RegisterDeviceSecurityRequirements
(
    const gapDeviceSecurityRequirements_t * pSecurity
);
```

The parameter is a pointer to a structure which contains a "device security setting" and service-specific security settings. All these security requirements are pointers to gapSecurityRequirements_t structures. The pointers that are to be ignored should be set to NULL.

Although the Peripheral does not initiate any kind of security procedure, it can inform the Central about its security requirements.

The informing is performed through the Peripheral Security Request packet at SMP level. To use it, the following GAP API is provided:

```
bleResult_t Gap_SendPeripheralSecurityRequest
(
    deviceId_t                  deviceId,
    const gapPairingParameters_t* pPairingParameters
);
```

The gapPairingParameters_t structure includes two important fields. The withBonding field indicates to the Central whether this Peripheral can bond and the securityModeAndLevel field informs about the required security mode and level that the Central should pair for. See Pairing and bonding (Central) for an explanation about security modes and levels, as defined by the GAP module.

This request expects no reply, nor any immediate action from the Central. The Central may easily choose to ignore the Peripheral Security Request.

If the two devices have bonded in the past, the Central proceeds directly to encrypting the link. If the bond was not made using LE Secure Connections, the Peripheral expects to receive a gConnEvtLongTermKeyRequest_c connection event. If the bond was made using LE Secure Connections, the Host provides the LTK automatically to the LE Controller.

When the devices have been previously pairing without using LE Secure Connections, along with the Peripheral's LTK, the EDIV (2 bytes) and RAND (8 bytes) values were also sent (their meaning is defined by the SMP). Therefore, before providing the key to the Controller, the application should check that the two values match with those received in the gConnEvtLongTermKeyRequest_c event. If they do, the application should reply with:

```
bleResult_t Gap_ProvideLongTermKey
(
    deviceId_t          deviceId,
    const uint8_t*      aLtk,
    uint8_t             ltkSize
);
```

The LTK size cannot exceed the maximum value of 16.

If the EDIV and RAND values do not match, or if the Peripheral does not recognize the bond, it can reject the encryption request with:

```
bleResult_t Gap_DenyLongTermKey
(
    deviceId_t deviceId
);
```

If LE SC Pairing was used then the LTK is generated internally by the Bluetooth LE Host Stack and it is not requested from the application during post-bonding link encryption. In this scenario, the application is only notified of the link encryption through the gConnEvtEncryptionChanged_c connection event.

If the devices are not bonded, the Peripheral should expect to receive the gConnEvtPairingRequest_c, indicating that the Central has initiated pairing.

If the application agrees with the pairing parameters (see Pairing and bonding (Central) for detailed explanations), it can reply with:

```
bleResult_t Gap_AcceptPairingRequest
(
    deviceId_t                      deviceId,
    const gapPairingParameters_t *    pPairingParameters
);
```

This time, the Peripheral sends its own pairing parameters, as defined by the SMP.

After sending this response, the application should expect to receive the same pairing events as the Central (see Pairing and bonding (Central)), with one exception: the gConnEvtPasskeyRequest_c event is not called if the application sets the Passkey (PIN) for pairing before the connection by calling the API:

```
bleResult_t Gap_SetLocalPasskey
(
    uint32_t passkey
);
```

This is done because, usually, the Peripheral has a static secret PIN that it distributes only to trusted devices. If, for any reason, the Peripheral must dynamically change the PIN, it can call the aforementioned function every time it wants to, before the pairing starts (for example, right before sending the pairing response with Gap_AcceptPairingRequest).

If the Peripheral application never calls Gap_SetLocalPasskey, then the gConnEvtPasskeyRequest_c event is sent to the application as usual.

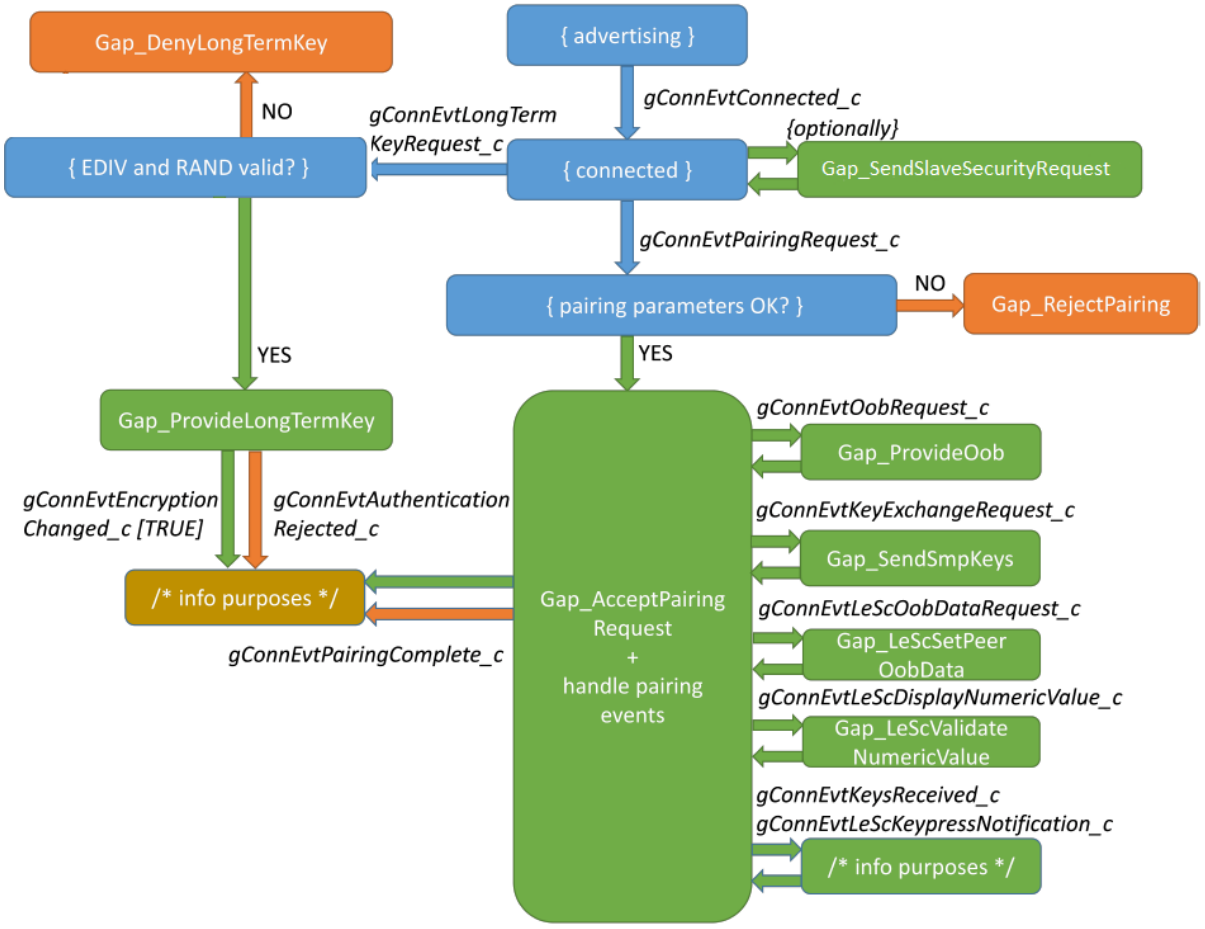
The Peripheral can use the following API to reject the pairing process:

```
bleResult_t Gap_RejectPairing
(
deviceId_t                      deviceId,
gapAuthenticationRejectReason_t    reason
);
```

The reason should indicate why the application rejects the pairing. The value gLinkEncryptionFailed_c is reserved for the gConnEvtAuthenticationRejected_c connection event to indicate the

link encryption failure rather than pairing failures. Therefore, it is not meant as a pairing reject reason.

The Gap_RejectPairing function may be called not only after the Pairing Request was received, but also during the pairing process. For example, when handling pairing events or asynchronously, if for any reason the Peripheral decides to abort the pairing, this function can be called. This also holds true for the Central. Figure illustrates the Peripheral pairing flow and lists the main APIs and events. Gap_RejectPairing can be called on any pairing event.



For both the Central and the Peripheral, bonding is performed internally and is not the application's concern. The gConnEvtPairingComplete_c event parameters inform the application if bonding has occurred.

Parent topic: Peripheral setup

Parent topic:*Generic Access Profile (GAP) Layer*

Central setup Usually, a Central must start scanning to find Peripherals. When the Central has scanned a Peripheral it wants to connect to, it stops scanning and initiates a connection to that Peripheral. After the connection has been established, it may start pairing, if the Peripheral requires it, or directly encrypt the link, if the two devices have already bonded in the past.

Scanning The most basic setup for a Central device begins with scanning, which is performed by the following function from *gap_interface.h*:

```
bleResult_t Gap_StartScanning
(
  const gapScanningParameters_t* pScanningParameters,
```

(continues on next page)

(continued from previous page)

```
  gapScanningCallback_t scanningCallback,
  gapFilterDuplicates_t enableFilterDuplicates,
  uint16_t duration,
  uint16_t period
);
```

If the *pScanningParameters* pointer is NULL, the currently set parameters are used. If no parameters have been set after a device power-up, the standard default values are used:

```
#define gGapDefaultScanningParameters_d \
{ \
    /* type */              gScanTypePassive_c, \
    /* interval */          gGapScanIntervalDefault_d, \
    /* window */              gGapScanWindowDefault_d, \
    /* ownAddressType */    gBleAddrTypePublic_c, \
    /* filterPolicy */      gScanAll_c \
    /* scanning PHY */      gLePhy1MFlag_c\
}
```

The easiest way to define non-default scanning parameters is to initialize a *gapScanningParameters_t* structure with the above default and change only the required fields.

For example, to perform active scanning and only scan for devices in the Filter Accept List, the following code can be used:

```
gapScanningParameters_t scanningParameters = gGapDefaultScanningParameters_d;
scanningParameters.type = gScanTypeActive_c;
scanningParameters.filterPolicy = gScanWithFilterAcceptList_c;
Gap_StartScanning(&scanningParamters, scanningCallback, enableFilterDuplicates, duration, period);
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartScanning
(
    appScanningParams_t    *pAppScanParams,
    gapScanningCallback_t pfCallback
);
```

The API uses the *appScanningParams_t* structure, which is defined as follows:

```
typedef struct appScanningParams_tag
{
    gapScanningParameters_t *pHostScanParams;         /*!< Pointer to host scan structure */
    gapFilterDuplicates_t enableDuplicateFiltering;   /*!< Duplicate filtering mode */
    uint16_t duration;                                /*!< scan duration  */
    uint16_t period;                                 /*!< scan period  */
} appScanningParams_t;
```

The *scanningCallback* is triggered by the GAP layer to signal events related to scanning.

The most important event is the *gDeviceScanned_c* event, which is triggered each time an advertising device is scanned. This event data contains information about the advertiser:

```
typedef struct
{
    bleAddressType_t                addressType ;
    bleDeviceAddress_t              aAddress ;
    int8_t                        rssi ;
    uint8_t                        dataLength ;
    uint8_t*                        data ;
```

(continues on next page)

(continued from previous page)

```
    bleAdvertisingReportEventType_t    advEventType ;
    bool_t                             directRpaUsed;
    bleDeviceAddress_t                     directRpa;
    bool_t                             advertisingAddressResolved;
} gapScannedDevice_t;
```

If this information signals a known Peripheral that the Central wants to connect to, the latter must stop scanning and connect to the Peripheral.

To stop scanning, call this function:

```
bleResult_t Gap_StopScanning (void);
```

By default, the GAP layer is configured to report all scanned devices to the application using the *gDeviceScanned_c* event type. However, some use cases might require to perform specific GAP Discovery Procedures. In such use cases the advertising reports might require the filtering of Flags AD value from the advertising data. Other use cases require the Bluetooth LE Host Stack to automatically initiate a connection when a specific device has been scanned.

To enable filtering based on the Flags AD value or to set device addresses for automatic connections, the following function must be called before the scanning is started:

```
bleResult_t Gap_SetScanMode
(
    gapScanMode_t            scanMode,
    gapAutoConnectParams_t* pAutoConnectParams,
    gapConnectionCallback_t connCallback
);
```

The default value for the scan mode is *gDefaultScan_c*, which reports all packets regardless of their content and does not perform any automatic connection.

To enable Limited Discovery, the *gLimitedDiscovery_c* value must be used, while the *gGeneralDiscovery_c* value activates General Discovery.

To enable automatic connection when specific devices are scanned, the *gAutoConnect_c* value must be set, in which case the *pAutoConnectParams* parameter must point to the structure that holds the target device addresses and the connection parameters to be used by the Host for these devices.

If scanMode is set to *gAutoConnect_c*, connCallback must be set and is triggered by GAP to send the events related to the connection.

Parent topic: Central setup

Initiating and closing a connection To connect to a scanned Peripheral, extract its address and address type from the *gDeviceScanned_c* event data, stop scanning, and call the following function:

```
bleResult_t Gap_Connect
(
const gapConnectionRequestParameters_t * pParameters,
gapConnectionCallback_t connCallback
);
```

When using the common application structure, the application can also use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_Connect
(
    gapConnectionRequestParameters_t*    pParameters,
```

(continues on next page)

(continued from previous page)

```
    gapConnectionCallback_t             connCallback
);
```

An easy way to create the connection parameter structure is to initialize it with the defaults, then change only the necessary fields. The default structure is defined as shown here:

```
#define gGapDefaultConnectionRequestParameters_d \
{ \
    /* scanInterval */       gGapScanIntervalDefault_d, \
    /* scanWindow */         gGapScanWindowDefault_d, \
    /* filterPolicy */       gUseDeviceAddress_c, \
    /* ownAddressType */     gBleAddrTypePublic_c, \
    /* peerAddressType */    gBleAddrTypePublic_c, \
    /* peerAddress */        { 0, 0, 0, 0, 0, 0 }, \
    /* connIntervalMin */    gGapDefaultMinConnectionInterval_d, \
    /* connIntervalMax */    gGapDefaultMaxConnectionInterval_d, \
    /* connLatency */        gGapDefaultConnectionLatency_d, \
    /* supervisionTimeout */ gGapDefaultSupervisionTimeout_d, \
    /* connEventLengthMin */ gGapConnEventLengthMin_d, \
    /* connEventLengthMax */ gGapConnEventLengthMax_d \
    /* initiatingPHYs     */ gLePhylMFlag_c \
}
```

In the following example, Central scans for a specific Heart Rate Sensor with a known address. When it finds it, it immediately connects to it.

```
static void BleApp_ScanningCallback
(
    gapScanningEvent_t *pScanningEvent
)
{
    switch (pScanningEvent->eventType)
    {
        case gDeviceScanned_c:
        {
            if (BleApp_CheckScanEvent(&pScanningEvent->eventData.scannedDevice))
            {
                gConnReqParams.peerAddressType = pScanningEvent->eventData.scannedDevice.addressType;
                FLib_MemCpy(gConnReqParams.peerAddress,
                        pScanningEvent->eventData.scannedDevice.aAddress,
                        sizeof(bleDeviceAddress_t));
                (void)Gap_StopScanning();
#if gAppUsePrivacy_d
                gConnReqParams.usePeerIdentityAddress =  pScanningEvent->eventData.scannedDevice.
↪advertisingAddressResolved;
#endif
                (void)BluetoothLEHost_Connect(&gConnReqParams, BleApp_ConnectionCallback);
            }
        }
        break;
}
```

The *connCallback* is triggered by GAP to send all events related to the active connection. It has the following prototype:

```
typedef void (* gapConnectionCallback_t )
(
    deviceId_t              deviceId,
    gapConnectionEvent_t * pConnectionEvent
);
```

The very first event that should be listened inside this callback is the *gConnEvtConnected_c* event.

If the application decides to drop the connection establishment before this event is generated, it should call the following macro:

```
#define Gap_CancelInitiatingConnection()\
    Gap_Disconnect(gCancelOngoingInitiatingConnection_d)
```

This is useful, for instance, when the application chooses to use an expiration timer for the connection request.

Upon receiving the *gConnEvtConnected_c* event, the application may proceed to extract the necessary parameters from the event data (*pConnectionEvent->event.connectedEvent*). The most important parameter to be saved is the *deviceId*.

The *deviceId* is a unique 8-bit, unsigned integer, used to identify an active connection for subsequent GAP and GATT API calls. All functions related to a certain connection require a *deviceId* parameter. For example, to disconnect, call this function:

```
bleResult_t Gap_Disconnect
(
    deviceId_t deviceId
);
```

Parent topic: Central setup

Pairing and bonding (Central) After the user has connected to a Peripheral, use the following function to check whether this device has bonded in the past:

```
bleResult_t Gap_CheckIfBonded
(
    deviceId_t    deviceId,
    bool_t *      pOutIsBonded,
    uint8_t*      pOutNvmIndex
);
```

If it has, link encryption can be requested with:

```
bleResult_t Gap_EncryptLink
(
    deviceId_t      deviceId
);
```

If the link encryption is successful, the *gConnEvtEncryptionChanged_c* connection event is triggered. Otherwise, a *gConnEvtAuthenticationRejected_c*event is received with the *rejectReason* event data parameter set to *gLinkEncryptionFailed_c*.

On the other hand, if this is a new device (not bonded), pairing may be started as shown here:

```
bleResult_t Gap_Pair
(
    deviceId_t                      deviceId,
    const gapPairingParameters_t *    pPairingParameters
);
```

The pairing parameters are shown here:

```
typedef struct gapPairingParameters_tag {
    bool_t                      withBonding ;
    gapSecurityModeAndLevel_t     securityModeAndLevel ;
    uint8_t                     maxEncryptionKeySize ;
    gapIoCapabilities_t           localIoCapabilities ;
    bool_t                      oobAvailable ;
    gapSmpKeyFlags_t              centralKeys ;
```

(continues on next page)

(continued from previous page)

```
    gapSmpKeyFlags_t            peripheralKeys ;
    bool_t                  leSecureConnectionSupported ;
    bool_t                  useKeypressNotifications ;
} gapPairingParameters_t;
```

The names of the parameters are self-explanatory. The *withBonding* flag should be set to *TRUE* if the Central must/wants to bond.

When Advanced Secure Mode is enabled, (*gAppSecureMode_d* id defined as *1* in *app_preinclude.h*), the security mode and level for pairing is automatically enforced as Mode 1 Level 4, and LE Secure Connection Supported is automatically enforced TRUE. Legacy pairing is not supported in this mode.

For the Security Mode and Level, the GAP layer defines them as follows:

- Security Mode 1 Level 1 stands for no security requirements.
- Except for Level 1 (which is only used with Mode 1), Security Mode 1 requires encryption, while Security Mode 2 requires data signing.
- Mode 1 Level 2 and Mode 2 Level 1 do not require authentication (in other words, they allow Just Works pairing, which has no MITM protection). Mode 1 Level 3 and Mode 2 Level 2 require authentication (must pair with PIN or OOB data, which provide MITM protection).
- Starting with Bluetooth specification 4.2, OOB pairing offers MITM protection only in certain conditions. The application must inform the stack if the OOB data exchange capabilities offer MITM protection via a dedicated API.
- Security Mode 1 Level 4 is reserved for authenticated pairing (with MITM protection) using a LE Secure Connections pairing method.
- If a pairing method is used but it does not offer MITM protection, then the pairing parameters must use Security Mode 1 level 2. If the requested pairing parameters are incompatible (for example, Security Mode 1 Level 4 without LE Secure Connections enabled), a gBleInvalidParameter_c status is returned by the security API functions: Gap_SetDefaultPairingParameters, Gap_SendPeripheralSecurityRequest, Gap_Pair and Gap_AcceptPairingRequest.

—	No security	No MITM protection	Legacy MITM protection	LE secure connections with MITM protection
Mode 1 (encryption) distributed LTK (EDIV+RAND) or generated LTK	**Level 1** no security	**Level 2** unauthenticated encryption	**Level 3** authenticated encryption	**Level 4** LE SC authenticated encryption
Mode 2 (data signing) distributed CSRK	—	**Level 1** unauthenticated data signing	**Level 2** authenticated data signing	—

The *centralKeys* should have the flags set for all the keys that are available in the application. The IRK is mandatory if the Central is using a Private Resolvable Address, while the CSRK is necessary if the Central wants to use data signing. The LTK is provided by the Peripheral and should only be included if the Central intends on becoming a Peripheral in future reconnections (GAP role change).

The *peripheralKeys* should follow the same guidelines. The LTK is mandatory if encryption is to be performed, while the peer's IRK should be requested if the Peripheral is using Private Resolvable Addresses.

See table below for detailed guidelines regarding key distribution.

The first three rows are both guidelines for Pairing Parameters (*centralKeys* and *peripheralKeys*) and for distribution of keys with *Gap_SendSmpKeys*.

If LE Secure Connections Pairing is performed (Bluetooth Low Energy 4.2 and above), then the LTK is generated internally, so the corresponding bits in the key distribution fields from the pairing parameters are ignored by the devices.

The Identity Address is distributed if the IRK is also distributed (its flag has been set in the Pairing Parameters). Therefore, it can be "asked" only by asking for IRK (it does not have a separate flag in a *gapSmpKeyFlags_t* structure). Therefore, it is N/A.

The negotiation of the distributed keys is as follows:

- In the SMP Pairing Request (started by *Gap_Pair*), the Central sets the flags for the keys it wants to distribute (*centralKeys*) and receive (*peripheralKeys*).

	Central keys (Central)	Peripheral keys (Central)	Peripheral () keys	Central keys (peripheral)
Long Term Key (LTK)+EDIV +RAND	If it wants to be a peripheral in a future reconnection	If it wants encryption	If it wants encryption	If it wants to become a central in a future reconnection
Identity Resolving Key (IRK)	If it uses or intends to use private resolvable addresses	If a peripheral is using a private resolvable address	If it uses or intends to use private resolvable addresses	If a central is using a private resolvable address
Connection Signature Resolving Key (CSRK)	If it wants to sign data as GATT Client	If it wants the peripheral to sign data as GATT Client	If it wants to sign data as GATT Client	If it wants the Central to sign data as GATT Client
Identity address	If it distributes the IRK	N/A	If it distributes the IRK	N/A

- The Peripheral examines the two distributions and must send an SMP Pairing Response (started by the *Gap_AcceptPairingRequest*) after performing any changes it deems necessary. The Peripheral is only allowed to set to 0 some flags that are set to 1 by the Central, but not the other way around. For example, it cannot request/distribute keys that were not offered/requested by the Central. If the Peripheral is adverse to the Central's distributions, it can reject the pairing by using the *Gap_RejectPairing* function.
- The Central examines the updated distributions from the Pairing Response. If it is adverse to the changes made by the Peripheral, it can reject the pairing (*Gap_RejectPairing*). Otherwise, the pairing continues and, during the key distribution phase (the *gConnEvtKeyExchangeRequest_c* event) only the final negotiated keys are included in the key structure sent with *Gap_SendSmpKeys*.
- For LE Secure Connections (both devices set the SC bit in the AuthReq field of the Pairing Request and Pairing Response packets), the LTK is not distributed. It is generated and the corresponding bit in the Initiator Key Distribution and Responder Key Distribution fields of the Pairing Response packet are set to 0.

If LE Secure Connections Pairing (Bluetooth LE 4.2 and above) is used, and OOB data needs to be exchanged, the application must obtain the local LE SC OOB Data from the Bluetooth LE Host Stack by calling the *Gap_LeScGetLocalOobData* function. The data is contained by the generic *gLeScLocalOobData_c* event.

The local LE SC OOB Data is refreshed in the following situations:

- The *Gap_LeScRegeneratePublicKey* function is called (the *gLeScPublicKeyRegenerated_c* generic event is also generated as a result of this API).
- The device is reset (which also causes the Public Key to be regenerated).

If the pairing continues, the following connection events may occur:

- **Request events**
  - *gConnEvtPasskeyRequest_c*: a PIN is required for pairing; the application must respond with the *Gap_EnterPasskey(deviceId, passkey).*
  - *gConnEvtOobRequest_c*: if the pairing started with the *oobAvailable* set to *TRUE* by both sides; the application must respond with the *Gap_ProvideOob(deviceId, oob).*
  - *gConnEvtKeyExchangeRequest_c*: the pairing has reached the key exchange phase; the application must respond with the *Gap_SendSmpKeys(deviceId, smpKeys).*
  - *gConnEvtLeScOobDataRequest_c*: the stack requests the LE SC OOB Data received from the peer (r, Cr and Addr); the application must respond with *Gap_LeScSetPeerOobData(deviceId, leScOobData).*
  - *gConnEvtLeScDisplayNumericValue_c*: the stack requests the display and confirmation of the LE SC Numeric Comparison Value; the application must respond with *Gap_LeScValidateNumericValue(deviceId, ncvValidated).*
- **Informational events**
  - *gConnEvtKeysReceived_c*: the key exchange phase is complete; keys are automatically saved in the internal device database and are also provided to the application for immediate inspection; application does not have to save the keys in NVM storage because this is done internally if *withBonding*was set to TRUE by both sides.
  - *gConnEvtAuthenticationRejected_c*: the peer device rejected the pairing; the *rejectReason* parameter of the event data indicates the reason that the Peripheral does not agree with the pairing parameters (it cannot be *gLinkEncryptionFailed_c* because that reason is reserved for the link encryption failure).
  - *gConnEvtPairingComplete_c*: the pairing process is complete, either successfully, or an error may have occurred during the SMP packet exchanges; note that this is different from the *gConnEvtKeyExchangeRequest_c* event; the latter signals that the pairing was rejected by the peer, while the former is used for failures due to the SMP packet exchanges.
  - *gConnEvtLeScKeypressNotification_c*: the stack informs the application that a remote SMP Keypress Notification has been received during Passkey Entry Pairing Method.

After the link encryption or pairing is completed successfully, the Central may immediately start exchanging data using the GATT APIs. Gap_RejectPairing may be called on any pairing event.

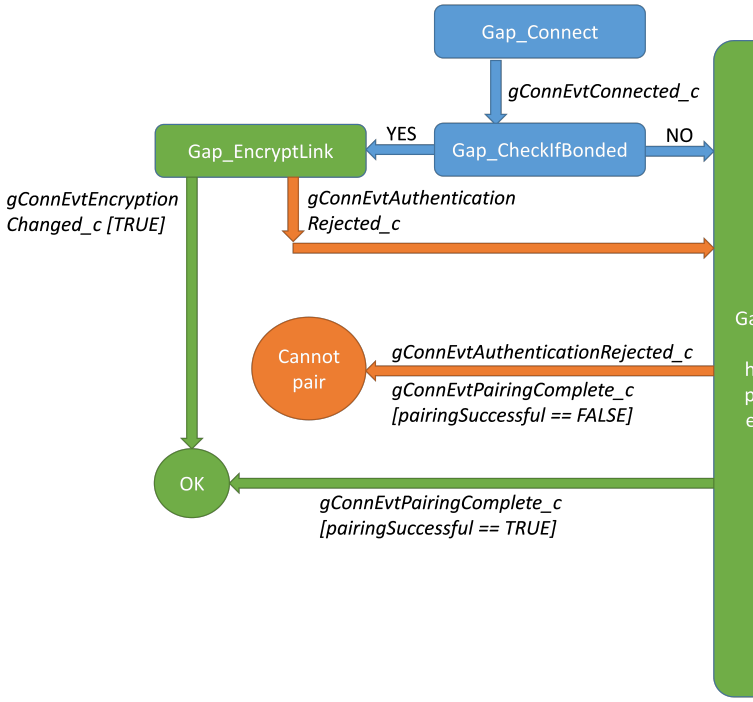


Figure 4. Central pairing flow – APIs and events.

Parent topic: Central setup

Parent topic:*Generic Access Profile (GAP) Layer*

LE data packet length extension This new feature extends the maximum data channel payload length from 27 to 251 octets.

The length management is done automatically by the link layer immediately after the connection is established. The stack passes the default values for maximum transmission number of payload octets and maximum packet transmission time that the application configures at compilation time in *ble_config.h*:

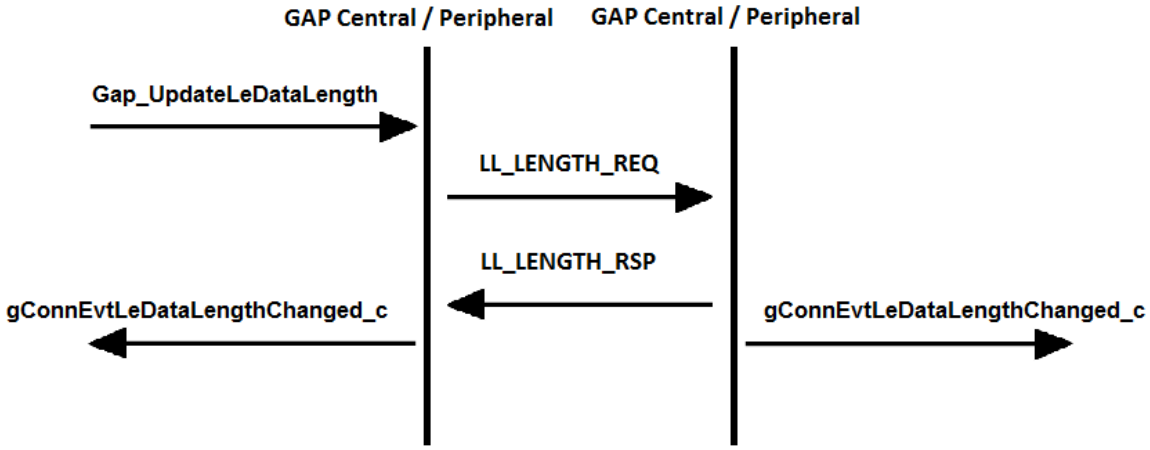
```
#ifndef gBleDefaultTxOctets_c
#define gBleDefaultTxOctets_c       0x00FB
#endif

#ifndef gBleDefaultTxTime_c
#define gBleDefaultTxTime_c         0x0848
#endif
```

The device can update the data length anytime, while in connection. The function that triggers this mechanism is the following:

```
bleResult_t Gap_UpdateLeDataLength
(
    deviceId_t          deviceId,
    uint16_t    txOctets,
    uint16_t    txTime
);
```

After the procedure executes, a *gConnEvtLeDataLengthChanged_c* connection event is triggered with the maximum values for number of payload octets and time to transmit and receive a link layer data channel PDU. The event is send event if the remote device initiates the procedure. This procedure is shown in Figure. **Figure 5. Data Length Update Procedure**



Parent topic:*Generic Access Profile (GAP) Layer*

Privacy feature

Introduction Starting with Bluetooth 4.2, Privacy can be enabled either in the Host or in the Controller:

- **Host Privacy** consists of two use cases that are described in detail in the following sections. These are:
  - Random address generation - Periodically regenerating a random address (Resolvable or Non-Resolvable Private Address) inside the Host and then applying it into the Controller.
  - Random address resolution - Trying to resolve incoming RPAs using the IRKs stored in the bonded devices list. The address resolution is performed when a connection is established with a device or for the autoconnect scan. The advertising packets that have an RPA are not resolved automatically due to the high MCU processing that is required.

    The random address resolution is performed by default by the Host whenever the Controller is not able to resolve an RPA. The Host performs random address generation only when Host Privacy is requested to be enabled. During random address generation, the advertising and scan operations, if active, are stopped and restarted. If errors occur during this process and the scan or advertising cannot be started, the application is notified through the corresponding event (gAdvertisingStateChanged_c, gExtAdvertisingStateChanged_c or gScanStateChanged_c)
- **Controller Privacy**, introduced by Bluetooth 4.2, consists of writing the local IRK in the Controller, together with all known peer IRKs, and letting the Controller perform hardware, fully automatic RPA generation and resolution. The Controller uses a Resolving List to store these entries. The size of the list is platform dependent and determined by gMaxResolvingListSize_c. For RPA resolution, the entries that do not fit in this list are processed by the Host to be resolved using the IRKs from Bonded Devices list.

Host Privacy can be enabled at any time. Controller Privacy should only be enabled while the device is not in the advertising, scanning or connection initiating state - otherwise the HCI LE Set Address Resolution Enable command will fail and privacy will not be considered as successfully enabled. Trying to enable one while the other is in progress generates a *gBleInvalidState_c* error. The same error is returned when trying to enable the same privacy type twice, or when trying to disable privacy when it is not enabled.

The recommended way of using Privacy is the Controller Privacy. However, enabling Controller Privacy requires at least a pair of local IRK and peer IRK, so this can only be enabled only after a pairing is performed with a peer and the IRKs are exchanged during the Key Distribution phase. When a device starts, if Privacy is required, the workflow is the following:

1. Enable Host Privacy using the local IRK.
2. Connect to a peer and perform pairing and bonding to exchange IRKs.
3. Disable Host Privacy.
4. Enable Controller Privacy using the local IRK and the peer IRK and peer identity address.

After enabling Host Privacy or Controller Privacy, the application must wait for the gHostPrivacyStateChanged_c or gControllerPrivacyStateChanged_c generic event and verify that privacy has been successfully enabled. Only then it is safe to proceed with setting advertising parameters (via the Gap_SetAdvertisingParameters or Gap_SetExtAdvertisingParameters APIs) or starting scanning (via the Gap_StartScanning API). Failure to do so could result in unwanted behavior, such as the device advertising or scanning with a public address.

Resolvable private addresses A Resolvable Private Address (RPA) is a random address generated using an Identity Resolving Key (IRK). This address appears completely random to an outside observer, so a device may periodically regenerate its RPA to maintain privacy, as there is no correlation between any two different RPAs generated using the same IRK.

On the other hand, an IRK can also be used to resolve an RPA, in other words, to check if this RPA has been generated with this IRK. This process is called “resolving the identity of a device”. Whoever has the IRK of a device can always try to resolve its identity against an RPA.

For example, assume device A frequently changes its RPA using IRKA. At some point, A bonds with B. A must give B a way to recognize it in a subsequent connection when it (A) has a different address. To achieve this purpose, A distributes the IRKA during the Key Distribution phase of the pairing process. B stores the IRKA it received from A.

Later, B connects to a device X that uses RPAX. This address appears completely random, but B can try to resolve RPAX using IRKA. If the resolving operation is successful, it means that IRKA was used to generate RPAX, and since IRKA belongs to device A, it means that X is A. So B was able to recognize the identity of device X, but nobody else can do that since they do not have IRKA.

Parent topic: Introduction

Non-resolvable private addresses A Non-Resolvable Private Address (NRPA) is a completely random address that has no generation pattern and therefore cannot be resolved by a peer.

A device that uses an NRPA that is changed frequently is impossible to track because each new address appears to belong to a new device.

Parent topic: Introduction

Multiple identity resolving keys If a device bonds with multiple peers, all of which are using RPAs, it needs to store the IRK of each in order to be able to recognize them later (see previous section).

This means that whenever the device connects to a peer that uses an unknown RPA, it needs to try and resolve the RPA with each of the stored IRKs. If the number of IRKs is large, then this introduces a lot of computation.

Performing all these resolving operations in the Host can be costly. It is much more efficient to take advantage of hardware acceleration and enable the Controller Privacy.

Parent topic: Introduction

Parent topic: Privacy feature

Host privacy To enable or disable Host Privacy, the following API may be used:

```
bleResult_t Gap_EnableHostPrivacy
(
    bool_t              enable,
    const uint8_t *     aIrk
);
```

When *enable* is set to TRUE, the *aIrk*parameter defines which type of Private Address to generate. If *aIrk* is NULL, then a new NRPA is generated periodically and written into the Controller. Otherwise, an IRK is copied internally from the *aIrk* address and it is used to periodically generate a new RPA.

The lifetime of the Private Address (NRPA or RPA) is a number of seconds contained by the *gGapHostPrivacyTimeout*external constant, which is defined in the *ble_config.c* source file. The default value for this is 900 (15 minutes).

When Host Privacy is enabled, the Host ignores the ownAddressType value for the advertising, scanning or connect parameters. It will always use the random address type in order to use the RPA configured in the Controller in the packets sent over the air.

As mentioned in the Introduction section, call this API for random address generation. For random address resolution there is no need to do so, it is performed by default against the bonded devices list.

Parent topic: Privacy feature

Controller privacy To enable or disable Controller Privacy, the following API may be used:

```
bleResult_t Gap_EnableControllerPrivacy
(
    bool_t                              enable,
    const uint8_t *                       aOwnIrk,
    uint8_t                             peerIdCount,
    const gapIdentityInformation_t*       aPeerIdentities
);
```

When *enable* is set to TRUE, *aOwnIrk*parameter shall not be NULL, *peerIdCount* shall not be zero or greater than *gMaxResolvingListSize_c,* and *aPeerIdentities* shall not be NULL.

The IRK defined by *aOwnIrk* is used by the Controller to periodically generate a new Resolvable Private Address (RPA). The lifetime of the RPA is a number of seconds contained by the *gGapControllerPrivacyTimeout*external constant, which is defined in the *ble_config.c* source file. The default value for this is 900 (15 minutes).

The *aPeerIdentities* is an array of identity information for each bonded device. The identity information contains the device's identity address (public or random static address) and the device's IRK. This array can be obtained from the Host with the *Gap_GetBondedDevicesIdentityInformation*API.

Enabling Controller Privacy involves a quick sequence of commands to the Controller. When the sequence is complete, the *gControllerPrivacyStateChanged_c* generic event is triggered.

Privacy mode In Bluetooth LE 5.0, the privacy mode has been introduced as an optional feature and is part of the GAP identity structure together with the address and address type. There are two modes: Network Privacy Mode (default) and Device Privacy Mode. These are valid only for Controller Privacy.

A device in network privacy mode only accepts packets from peers using private addresses.

A device in device privacy mode also accepts packets from peers using identity addresses, even if the peer had previously distributed the IRK. Private addresses are also accepted.

The privacy mode of a device is stored in NVM together with the IRK with a default value of Network. If the application wants to change this value it can extract the peer identities, modify the privacy mode from network to device and then enable Controller Privacy with the value.

To change the privacy mode of a device and make the change persistent, the user must call the following API:

```
bleResult_t Gap_SetPrivacyMode
(
    uint8_t nvmIndex,
    blePrivacyMode_t privacyMode
);
```

Parent topic: Controller privacy

Scanning and initiating When a Central device is scanning while Controller Privacy is enabled, the Controller actively tries to resolve any RPA contained in the Advertising Address field of advertising packets. If any match is found against the peer IRK list, then the *advertisingAddressResolved* parameter from the scanned device structure is set to TRUE.

In this case, the *addressType* and *aAddress* fields no longer contain the actual Advertising Address as seen over the air, but instead they contain the identity address of the device whose IRK was able to resolve the Advertising Address. In order to connect to this device, these fields shall be used to complete the *peerAddressType* and *peerAddress* fields of the connection request parameter structure, and the *usePeerIdentityAddress* field shall be set to TRUE.

If *advertisingAddressResolved* is equal to FALSE, then the advertiser is using a Public or Random Static Address, an NRPA, or a RPA that could not be resolved. Therefore, the connection to this device is initiated as if Controller Privacy was not enabled, by setting *usePeerIdentityAddress* to FALSE.

Parent topic: Controller privacy

Advertising When a Peripheral starts advertising while Controller Privacy is enabled, the *ownAddressType* field of the advertising parameter structure is unused. Instead, the Controller always generates an RPA and advertises with it as Advertising Address.

If directed advertising is used, the Host only allows advertising to a device in the resolving list in order to be able to generate RPAs.

Parent topic: Controller privacy

Connected When a device connects while Controller Privacy is enabled, the *gConnEvtConnected_c* connection event parameter structure contains more relevant fields than without Controller Privacy.

The *peerRpaResolved* field equals TRUE if the peer was using an RPA that was resolved using an IRK from the list. In that case, the *peerAddressType* and *peerAddress*fields contain the identity address of the resolved device, and the actual RPA used to create the connection (the RPA that a Central used when initiating the connection, or the RPA that the Peripheral advertised with) is contained by the *peerRpa* field.

The *localRpaUsed*field equals TRUE if the local Controller was automatically generating an RPA when the connection was created, and the actual RPA is contained by the *localRpa* field.

Parent topic: Controller privacy

Parent topic: Privacy feature

Parent topic:*Generic Access Profile (GAP) Layer*

Setting PHY mode in a connection In Bluetooth LE 5.0, the user is able to change the PHY mode in a connection through the Link Layer PHY Update Procedure and choose between default 1 Mbit/s, 2 Mbit/s high data rate or the coded S2 or S8 PHYs with 500 Kbps or 125 Kbps for longer range.

To set the PHY, the user can call:

```
bleResult_t Gap_LeSetPhy
(
    bool_t        defaultMode,
    deviceId_t    deviceId,
    uint8_t       allPhys,
    uint8_t       txPhys,
    uint8_t       rxPhys,
    uint16_t      phyOptions
);
```

There are two modes to use this API:

1. If defaultMode is set to TRUE, the user can call this function without being in a connection, i.e. provide a device ID. The PHY option is used by the Link Layer in the PHY response when a connection is created and the peer device initiates the PHY Update Procedure. The application should listen for gLePhyEvent_c with the gPhySetDefaultComplete_c sub event type for the confirmation of the operation.
2. If defaultMode is set to FALSE, the user must also provide a valid device ID. The Host asks the Link Layer to initiate the PHY Update Procedure with the peer device using the provided parameters.

The application should listen for gLePhyEvent_c with the gPhyUpdateComplete_c sub event type for the confirmation of the update procedure to have ended. The result of the operation populates in the txPhy and rxPhy of the event. The result is from the negotiation of the local parameters and the peer PHY preferences.

To read the current PHY on a connection, call the following API:

```
bleResult_t Gap_LeReadPhy
(
    deviceId_t deviceId
);
```

The application should listen for gLePhyEvent_c with the gPhyRead_c sub event type for the confirmation of the operation. The txPhy and rxPhy indicate the current modes used in the connection.

Parent topic:*Generic Access Profile (GAP) Layer*

Data management of bonded devices The Host handles the management of the bonding data without requiring application intervention. The application must provide the NVM write, read, and erase functions presented in Non-Volatile Memory (NVM) access. The Host creates bonds if bonding is required after the pairing.

The bonded data structure is presented below, together with the GAP APIs that access it, for most APIs require a connection to be established with the device in the bonded list, the others can be accessed any time using the NVM index.

1. Bond Header – identity address and address type that uniquely identify a device together with the IRK and privacy mode.
    - Gap_GetBondedDevicesIdentityInformation – for all bonds

2. Bond Data Dynamic - security counters for signed operations – managed by the stack
3. Bond Data Static – LTK, CSRK, Rand, EDIV, security information for read and write authorizations
    - Gap_SaveKeys– NVM index
    - Gap_LoadKeys– NVM index
    - Gap_LoadEncryptionInformation - deviceId
    - Gap_Authorize – deviceId - GATT Server only
4. Bond Data Legacy - Legacy pair information and CSRK
    - Gap_LoadEncryptionInformation - deviceId
5. Bond Data Device Info - custom peer information (service discovery data) and device name
    - Gap_SaveCustomPeerInformation - deviceId
    - Gap_LoadCustomPeerInformation - deviceId
    - Gap_SaveDeviceName - deviceId
    - Gap_GetBondedDeviceName – NVM index
6. Bond Data Descriptor List - configuration of indications and notifications for CCCD handles – GATT Server only
    - Gap_CheckNotificationStatus - deviceId
    - Gap_CheckIndicationStatus- deviceId

However, there may be some cases when an application wants to manage this data to read data from a bonded device created by the Host, create a bond obtained out-of-band or update an existing bond. For this use case, two GAP APIs and a GAP event have been added.

1. **Load the Keys of a bonded device**.

    The user can call the following function to read the keys exchanged during pairing and stored by the Bluetooth LE Host Stack in the bond area when the pairing is complete.

    The application is informed of the NVM index through the gBondCreatedEvent_c sent by the stack immediately after the bond creation. The application is responsible for passing the memory in the pOutKeys OUT parameter to fill in the keys, if any of the keys are set to NULL, the stack does not fill that information. The pOutKeyFlags OUT parameter indicates to the application which of the keys were stored by the stack as not all of them may have been distributed during pairing.

    The pOutLeSc indicates if Bluetooth LE 4.2 LE Secure Connections Pairing was used, while the pOutAuth indicates if the peer device is authenticated for MITM protection. All these OUT parameters are recommended to be retrieved from the bond and added if later passed as input parameters for the save keys API.

    This function executes synchronously.

```
bleResult_t Gap_LoadKeys
(
    uint8_t             nvmIndex,
    gapSmpKeys_t*       pOutKeys,
    gapSmpKeyFlags_t*   pOutKeyFlags,
    bool_t*             pOutLeSc,
    bool_t*             pOutAuth
);
```

    The gapSmpKeys_t is the structure used during the key distribution phase, as well as in the gConnEvtKeysReceived_c event and is as follows. The difference is that the Bluetooth LE device address cannot be set to NULL neither when loading a bond or when creating one

as it identifies the bonded device together with the NVM index.
Table 3. 'gapSmpKeys_t' structure

Event Data	Data type	Data Description
cLtk-Size	uint8_t	Encryption Key Size filled by the stack. If aLtk is NULL, this is ignored.In Advanced Secure Mode, this should be the size of the LTK encrypted blob of 40 bytes.
aLtk	uint8_t*	Long Term (Encryption) Key or LTK encrypted blob if Advanced Secure Mode is enabled. NULL if LTK isnot distributed, else size is given by cLtk-Size
aIrk	uint8_t*	Identity Resolving Key. NULL if aIrk is not distributed.
aCsrk	uint8_t*	Connection Signature Resolving Key. NULL if aCsrk is not distributed.
cRand-Size	uint8_t	Size of RAND filled by the stack; usually equal to gcSmpMaxRandSize_c. IfaLtk is NULL, this is ignored.
aRand	uint8_t*	RAND value used to identify the LTK. If aLtk is NULL, this is ignored.
ediv	uint16_t	EDIV value used to identify the LTK. If aLtk is NULL, this is ignored.
ad-dressTy	bleAd-dress Type_t	Public or Random address.
aAd-dress	uint8_t*	Device Address. It cannot be NULL.

The structure for the GAP SMP Key Flags is the following:

Table 4. GAP SMP Key Flags

Flag Type	Description
gNoKeys_c	No key is available.
gLtk_c	Long-Term Key is available.
gIrk_c	Identity Resolving Key is available.
gCsrk_c	Connection Signature Resolving Key is available.

2. Save the Keys to create a bond or update an existing bonded device.

   The user can call the following function to create a bond on a device based on information obtained Out of Band. For instance, one can use the output of Gap_LoadKeys from the previous section. This can be useful in transferring a bond created by the stack after a pairing procedure or if the application wants to manipulate bonding data. The behavior of the stack remains the same, if the bonding is required after a pairing, the stack stores the bonding information if possible. In this case, the NVM index is passed to the application through gBondCreatedEvent_c.

   This function executes asynchronously, as the stack can create a bond during the execution. The application should listen for the previous mentioned event gBondCreatedEvent_c. The result of the function call is passed synchronously. However, if an asynchronous error has occurred during the actual save, it is passed to the application through the gInternalError_c event with a gSaveKeys_c error source.

   The stack creates a bond if the NVM index is free or update the keys from an NVM index if it stores a valid entry.

   The address from the GAP SMP Keys structure must not be NULL. If other members of the structure are NULL, they are ignored.

   LE SC flag indicates if Bluetooth LE 4.2 Secure Connections was used during pairing and Auth specifies if the peer is authenticated for MITM protection.

```
bleResult_t Gap_SaveKeys
(
    uint8_t                 nvmIndex,
    const gapSmpKeys_t*     pKeys,
    bool_t                  leSc,
    bool_t                  auth
);
```

3. Bond created event.

   A GAP event is added to the Bluetooth LE Generic Callback to inform the application of the NVM index whenever the stack creates a bond or when a Gap_SaveKeys request succeeds. The event is also generated if the NVM index was a valid occupied entry and only some of the keys in the bonded information have been updated.

   The NVM index is then used in the GAP APIs to save or load information from the bond.

Event Data	Data type	Data Description
nvmIndex	uint8_t	NVM index for the new created bond
addressType	bleAddressType_t	Public or Random (static) address of the bond
address	bleDeviceAddress_t	Address of the bond

Application removal of bonded devices data The application can remove a bonded device from NVM. The bonded device cannot be deleted if it is in an active connection. The application can remove one or all bonds by calling the following synchronous GAP APIs:

- Gap_RemoveBond(uint8_t nvmIndex) – nvmIndex can be obtained via the Gap_CheckIfBonded API.
- Gap_RemoveAllBonds() - no connections should be active otherwise the call fails.

Removing a bonded device does not affect the controller address resolution state nor the contents of either the Controller Filter Accept List or the Controller Resolving List. If Controller Privacy is enabled, it remains so until it is disabled or the device is reset.

In a scenario where the user wants to remove a bonded device and all its effects on device behavior (Controller Filter Accept List, Controller Resolving List), the following operations should be executed:

- Gap_ClearFilterAcceptList or Gap_RemoveDeviceFromFilterAcceptList
  - Clear Controller Filter Accept List or clear a device from Filter Accept List.
- Gap_RemoveAllBonds or Gap_RemoveBond
  - All bonded devices are removed or one bonded device is removed from NVM.
- BleConnManager_DisablePrivacy
  - Controller Privacy is disabled, Controller Resolving List is cleared and address resolution is disabled. The device should not be advertising or scanning, otherwise this call fails.
- BleConnManager_EnablePrivacy
  - Called after the gControllerPrivacyStateChanged_c event is received, confirming Controller Privacy has been disabled. If not all bonds have been deleted, Controller Privacy is reenabled. In the absence of bonds, Host Privacy is enabled.

Parent topic: Data management of bonded devices

Parent topic:*Generic Access Profile (GAP) Layer*

Controller enhanced notifications This section describes how the application can configure and monitor the notifications generated by the Bluetooth Controller when advertising, scan, or connection events occur. This feature is proprietary to NXP that is available on selected Controllers.

The user can choose between two options:

1. Enable notifications from the GAP layer and monitor GAP events in the GAP Generic Callback. The controller issues HCI vendor-specific events processed by the Bluetooth LE Host and presented to the application in the GAP Generic Callback.
2. Enable notifications from the Controller interface and monitor controller events in a user-defined Application Callback.
3. Combination of the above two options: configure feature at GAP layer and install an Application Callback through the Controller interface. After setting the callback, the HCI vendor-specific events are not issued and implicitly the GAP events. Instead, the user receives the notifications in the installed callback until setting the callback to NULL again if it wants to revert to GAP events.

- **GAP configuration:**

  The user should call the following function to enable various events from the mask or use Event None to disable the feature. The Device ID is valid only for connection events.

```
bleResult_t Gap_ControllerEnhancedNotification
(    uint16_t eventType,
   deviceId_t deviceId
);
```

  The event type is a bitmask having the following options:
  Table 5. Event types and their description

Event Type	Event Description
gNotifEventNone_c	No enhanced notification event enabled
gNotifConnEventOver_c	Connection event over
gNotifConnRxPdu_c	Connection RX PDU
gNotifAdvEventOver_c	Advertising event over
gNotifAdvTx_c	Advertising ADV transmitted
gNotifAdvScanReqRx_c	Advertising SCAN REQ RX
gNotifAdvConnReqRx_c	Advertising CONN REQ RX
gNotifScanEventOver_c	Scanning event over
gNotifScanAdvPktRx_c	Scanning ADV PKT RX
gNotifScanRspRx_c	Scanning SCAN RSP RX
gNotifScanReqTx_c	Scanning SCAN REQ TX
gNotifConnCreated_c	Connection created
gNotifChannelMatrix_c	Enable channel status monitoring
gNotifPhyReq_c	Phy Req Pdu ack received
gNotifConnChannelMapUpdate_c	Channel map update
gNotifConnInd_c	Connect indication
gNotifPhyUpdateInd_c	Phy update indication

  After enabling events, the user should wait for a gControllerNotificationEvent_c GAP Generic Event in the GAP Generic Callback. The first event received should have the event type set to gNotifEventNone_c with a status of success confirming the selected event mask has been enabled. The same event types apply for both the GAP command and the GAP event. The structure for the Controller Notification event is the following:
  Table 6. Controller Notification Event structure

Event Data	Data type	Data Description
event-Type	bleNotification-Event_t	Enhanced notification event type
devi-ceId	deviceId_t	Device id of the peer, valid for connection events
rssi	int8_t	RSSI, valid for RX event types
channel	uint8_t	Channel, valid for connection event over or Rx/Tx events
ce_counte	uint16_t	Connection event counter, valid for connection events only
status	bleResult_t	Status of the request to select which events to be enabled/disabled
times-tamp	uint16_t	Timestamp in 625 µs slots, valid for Conn RX event and Conn Created event
adv_hand	uint8_t	Advertising Handle, valid for advertising events, if multiple ADV sets supported

- **Controller configuration:**

  The user should call the following function to enable various events from the mask or use Event None to disable the feature. The same event types apply as the GAP layer types. The connection handle is valid only for connection events.

```
bleResult_t Controller_ConfigureEnhancedNotification
(
    uint16_t eventType,
    uint16_t conn_handle
);
```

  The event monitoring is done in a user-installed callback by calling:

```
bleResult_t Controller_RegisterEnhancedEventCallback
(
    bleCtrlNotificationCallback_t notificationCallback
);
```

  Where the types are the following:

```
typedef struct bleCtrlNotificationEvent_tag
{
    uint16_t event_type; /*! bleNotificationEventType_t */
    uint16_t conn_handle;
    uint8_t  rssi;
    uint8_t  channel_index;
    uint16_t conn_ev_counter;
    uint16_t timestamp;
    uint8_t  adv_handle;
} bleCtrlNotificationEvent_t;
typedef void (*bleCtrlNotificationCallback_t)
(
    bleCtrlNotificationEvent_t    *pNotificationEvent
);
```

  The event structure is nearly identical as the GAP one, except there is no status as the function call executes synchronously.

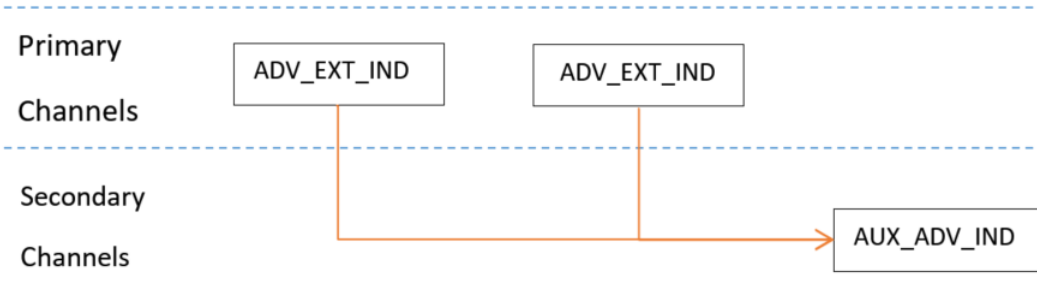
Table 7. 'bleCtrlNotificationEvent_tag' Event structure

Event Data	Data type	Data Description
event_type	bleNotifica-tionEvent_t	Enhanced notification event type
conn_handle	uint16_t	Connection handle of the peer, valid for connection events
rssi	int8_t	RSSI, valid for RX event types
chan-nel_index	uint8_t	Channel, valid for connection event over or Rx/Tx events
conn_ev_cou:	uint16_t	Connection event counter, valid for connection events only
timestamp	uint16_t	Timestamp in 625 µs slots, valid for Conn RX event and Conn Created event
adv_handle	uint8_t	Advertising Handle, valid for advertising events, if multiple ADV sets supported

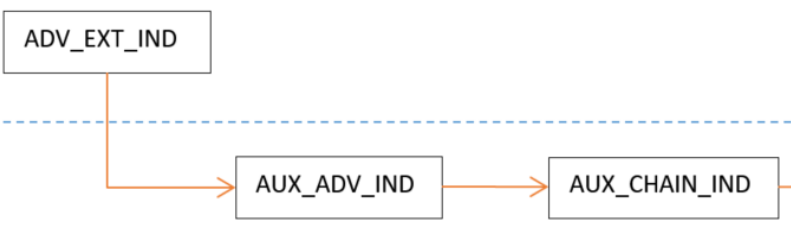
Parent topic:*Generic Access Profile (GAP) Layer*

Extended advertising Starting with Bluetooth 5, the advertising channels are separated in primary advertising channels and secondary advertising channels:

1. Primary advertising channels
   - Use 3 legacy advertising channels 37, 38, and 39.
   - Can use either legacy 1M PHY or new LE Coded PHY.
   - PHY payload can vary from 6 to 37 bytes.
   - Packets on these channels are part of the advertising events.
2. Secondary advertising channels
   - Use 37 channels, with the same channel index as the data channels.
   - Can use any LE PHY, but the same PHY during an Extended Advertising Event.
   - PHY payload can vary from 0 to 255 bytes.
   - Auxiliary packets on these channels are part of the Extended Advertising Event that begins at the same time with the advertising event on primary channel and ends with the last packet on the secondary channel.



Extended advertising



Extended advertising – Multiple chains

An advertising data set is represented by advertising PDUs belonging together in an advertising event. Each set has different advertising parameters: PDU type, advertising interval, and PHY mode. The advertising data sets are identified by the Advertising SID (Set ID) field from the ADI – Advertising Data Info. Advertising data or Scan response data can be changed for each adverting data set and the random value of DID (Data ID) field is updated to differentiate between them.

Refer to Figure 1 and Figure 2.

Peripheral setup This section describes the extended advertising GAP API. The application should not use both the extended and legacy API Refer to Scanning. If this requirement cannot be met, the application should at least wait for the generated events in the Advertising Callback prior to using the other API. That is, it is advisable to call legacy functions only after the event pertaining to an extended API is received, and vice versa. This GAP constraint can be considered an extension of the HCI constraint from the Bluetooth 5 specification: "A Host should not issue legacy commands to a Controller that supports the LE Feature (Extended Advertising)".

The application configures extended advertising by going through the following states:

1. Set the extended advertising parameters by calling:

```
bleResult_t Gap_SetExtAdvertisingParameters
(
gapExtAdvertisingParameters_t* pAdvertisingParameters
);
```

   It may use the default set of parameters *gGapDefaultExtAdvertisingParameters_d*. The application should wait for a *gExtAdvertisingParametersSetupComplete_c event* in the Generic Callback. Only one advertising set can be configured at a time. Comparing with the legacy *Gap_SetAdvertisingParameters* command, the new set of parameters is as follows.

Parameter	Description
SID	Value of the Advertising SID subfield in the ADI field of the PDU.
handle	Used to identify an advertising set. Possible values are 0x00 or 0x01 since the current implementation supports two advertising sets.
extAdvProperties	BIT0 - Connectable advertising
extAdvProperties	BIT1 - Scannable advertising
extAdvProperties	BIT2 - Directed advertising
extAdvProperties	BIT3 - High Duty Cycle Directed Connectable advertising (≤3.75 ms Advertising Interval)
extAdvProperties	BIT4 - Use legacy advertising PDUs
extAdvProperties	BIT5 - Omit advertiser's address from all PDUs ("anonymous advertising")
extAdvProperties	BIT6 - Include TxPower in the extended header of the advertising PDU. If legacy advertising PDU types are being used (BIT4 = 1), permitted properties values are presented in the next table. If the advertising set already contains data, the type shall be one that supports advertising data and the amount of data shall not exceed 31 octets. If extended advertising PDU types are being used (BIT4 =0), then the advertisement shall not be both connectable and scannable. While high duty cycle directed connectable advertising (≤ 3.75 ms advertising interval) shall not be used (BIT3 = 0).
txPowe	Maximum power level at which the advertising packets are to be transmitted, the Controller can choose any power level <= txPower. Value 127 to be used if Host has no preference.
primaryPHY	PHY for ADV_EXT_IND: LE 1 M or LE Coded
secondaPHY	PHY for AUX_ADV_IND and periodic advertising: LE 1 M, LE 2 M or LE Coded. Ignored for legacy advertising
secondaMaxS	Maximum advertising events that the Controller can skip before sending the AUX_ADV_IND packets on the secondary advertising channel. Higher values may result in lower power consumption. Ignored for legacy advertising
enableScanReqNotifi-	Whether to enable notifications when scanning PDUs (SCAN_REQ, AUX_SCAN_REQ) are received. If enabled, the application is notified upon scan requests by gExtScanNotification_c events in the Advertising Callback

When using LE Coded PHY for advertising, the default coding scheme chosen by link layer is S=8 (125 kb/s data rate). To change the default coding scheme, the user has two options:

- At compile time by defining *mLongRangeAdvCodingScheme_c*, or
- At run time by calling the API *Controller_ConfigureAdvCodingScheme()*. In both cases, the value of the define or the parameter of the API has to be an appropriate value for primary and secondary PHYs as defined by the enumeration *advCodingScheme_tag* found in *controller_interface.h*. **API Controller_ConfigureAdvCodingScheme()**

EventType	PDU Type	Advertising Event Properties
Connectable and scannable undirected	ADV_IND	00010011b
Connectable directed (low duty cycle)	ADV_DIRECT_IND	00010101b
Connectable directed (high duty cycle)	ADV_DIRECT_IND	00011101b
Scannable undirected	ADV_SCAN_IND	00010010b
Non-connectable and Nonscannable undirected	ADV_NONCONN_INI	00010000b

2. Set the advertising data and/or scan response data by calling:

```
bleResult_t Gap_SetExtAdvertisingData
(
uint8_t handle,
gapAdvertisingData_t* pAdvertisingData,
gapScanResponseData_t* pScanResponseData
);
```

Note: Either of the pAdvertisingData or pScanResponseData parameters can be NULL, but not both. For extended advertising (BIT4 = 0) only one must be different than NULL – the scannable advertising bit (BIT1) indicates whether pAdvertisingData (BIT1 = 0) orpScanResponseData (BIT1 = 1) is accepted. The total amount of Advertising Data shall not exceed 1650 bytes. Application should wait for a *gExtAdvertisingDataSetupComplete_c* event in the Generic Callback.

3. Enable extended advertising by calling:

```
bleResult_t Gap_StartExtAdvertising
(
gapAdvertisingCallback_t advertisingCallback,
gapConnectionCallback_t connectionCallback,
uint8_t handle,
uint16_t duration,
uint8_t maxExtAdvEvents
);
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartExtAdvertising
(
    appExtAdvertisingParams_t *pExtAdvParams,
    gapAdvertisingCallback_t  pfAdvertisingCallback,
    gapConnectionCallback_t   pfConnectionCallback
);
```

The API goes through the steps of setting the advertising data and parameters. Events from the Host task are treated in the *App_AdvertiserHandler()* function, implemented in *app_advertiser.c*. To set the extended advertising parameters and data *BluetoothLEHost_StartExtAdvertising* a parameter of the following type:

```
typedef struct appExtAdvertisingParams_tag
{
    gapExtAdvertisingParameters_t *pGapExtAdvParams;
    gapAdvertisingData_t *pGapAdvData;
    gapScanResponseData_t *pScanResponseData;
    uint8_t                handle;
    uint16_t               duration;
    uint8_t                maxExtAdvEvents;
} appExtAdvertisingParams_t;
```

Advertising may be enabled for each previously configured advertising set, identified by the handle parameter. If duration is set to 0, advertising continues until the Host disables it, otherwise advertising is only enabled for this period (multiple of 10 ms). *maxExtAdvEvents* represent the maximum number of extended advertising events the Controller shall attempt to send prior to terminating the extended advertising, ignored if set to 0. Application should wait for a *gExtAdvertisingStateChanged_c* or a *gAdvertisingCommandFailed_c* event in the Advertising Callback.

4. Disable advertising by calling:

```
bleResult_t Gap_StopExtAdvertising
(
uint8_t handle
);
```

Application should wait for a *gExtAdvertisingStateChanged_c* or a *gAdvertisingCommandFailed_c* event in the Advertising Callback.

5. Remove the advertising set by calling:

```
bleResult_t Gap_RemoveAdvSet
(
uint8_t handle
);
```

Application should wait for a *gExtAdvertisingSetRemoveComplete_c* event in the Generic Callback.

Parent topic: Extended advertising

Central setup The application configures the extended scanning by going through the following states:

1. Start scanning by calling:

```
bleResult_t Gap_StartScanning
(
const gapScanningParameters_t* pScanningParameters,
gapScanningCallback_t scanningCallback,
gapFilterDuplicates_t enableFilterDuplicates,
uint16_t duration,
uint16_t period
)
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartScanning
(
    appScanningParams_t   *pAppScanParams,
    gapScanningCallback_t pfCallback
);
```

The API starts scanning using the given parameters, which must have the following structure:

```
typedef struct appScanningParams_tag
{
    gapScanningParameters_t *pHostScanParams;        /*!< Pointer to host scan structure */
    gapFilterDuplicates_t enableDuplicateFiltering;  /*!< Duplicate filtering mode */
    uint16_t duration;                                    /*!< scan duration  */
    uint16_t period;                                      /*!< scan period  */
} appScanningParams_t;
```

Application may use the default set of parameters *gGapDefaultExtScanningParameters_d*. If the *pScanningParameters* pointer is NULL, the latest set of parameters are used. The *scanningPHYs* parameter indicates the PHYs on which the advertising packets should be received on the primary advertising channel. As a result, permitted values for the parameter are 0x01 (scan LE 1M), 0x04 (scan LE Coded) and 0x05 (scan both LE 1M and LE Coded). There are no strict timing rules for scanning, yet if both PHYs are enabled for scanning, the scan interval value must be large enough to accommodate two scan windows (interval >= 2 * window).

If the advertiser uses legacy advertising PDUs, the device may actively scan by sending a SCAN_REQ PDU to the advertiser on the LE 1M primary advertising channel (no secondary channel in legacy advertising). Respectively, if the advertiser uses extended advertising PDUs, the active scan operation takes place on the secondary advertising channel. After the device receives a scannable ADV_EXT_IND PDU on the primary advertising channel (PHY LE 1M or Coded), it starts listening for the AUX_ADV_IND PDU on the secondary advertising channel (PHY 1M, 2M or Coded). Once received, the device sends an AUX_SCAN_REQ to the advertiser. Next, an AUX_SCAN_RSP PDU should be received, containing the scan response data. Application should wait for a *gScanStateChanged_c* or a *gScanCommandFailed_c*in the Scanning Callback.

2. Collect information by waiting for *gDeviceScanned_c* (legacy advertising PDUs) or *gExtDeviceScanned_c* (extended advertising PDUs) event in the Scanning Callback. The *gExtDeviceScanned_c* event contains additional information pertaining to the extended received PDU, such as: primary PHY, secondary PHY, advertising SID, interval of the periodic advertising if enabled in the set.

   When using the common application structure, the application can use the following API defined in *app_conn.h*, to search the contents from *pData* in an advertising element:

   ```
   bool_t BluetoothLEHost_MatchDataInAdvElementList
   (
       gapAdStructure_t *pElement,
       void             *pData,
       uint8_t          iDataLen
   );
   ```

3. Stop scanning by calling the function below:

   ```
   bleResult_t Gap_StopScanning(void);
   ```

   Application should wait for a *gScanStateChanged_c* or a *gScanCommandFailed_c* in the Scanning Callback.

4. Connect to a device by calling the function below:

   ```
   bleResult_t Gap_Connect
   (
   const gapConnectionRequestParameters_t* pParameters,
   gapConnectionCallback_t connCallback
   );
   ```

   When using the common application structure, the following API can be used:

```
bleResult_t BluetoothLEHost_Connect
(
    gapConnectionRequestParameters_t*    pParameters,
    gapConnectionCallback_t              connCallback
);
```

The *initiatingPHYs* parameter indicates the PHYs on which the advertising packets should be received on the primary advertising channel and the PHYs for which connection parameters have been specified. The parameter is a bitmask of PHYs: BIT0 = LE 1M, BIT1 = LE 2M and BIT2 = LE Coded. The Host may enable one or more initiating PHYs, but it must at least set one bit for a PHY allowed for scanning on the primary advertising channel, i.e., BIT0 for LE 1M PHY or BIT2 for LE Coded PHY.

If the advertiser uses legacy advertising PDUs, the device may connect by sending a CONNECT_IND PDU to the advertiser on the LE 1M primary advertising channel (no secondary channel in legacy advertising). On the other hand, if the advertiser uses extended advertising PDUs, the extended connect operation takes place on the secondary advertising channel. After the device receives a connectable ADV_EXT_IND PDU on the primary advertising channel (PHY LE 1M or Coded), it starts listening for the connectable AUX_ADV_IND PDU on the secondary advertising channel (PHY 1M, 2M or Coded). Once received, the device sends an AUX_CONNECT_REQ to the advertiser. Next, if AUX_CONNECT_RSP PDU is received, the device enters the Connection State in the Central role on the secondary advertising channel PHY.

Application should wait for a *gConnEvtConnected_c* event in the Connection Callback. If the channel selection algorithm #2 is used for this connection, then a *gConnEvtChanSelectionAlgorithm2_c* event is also generated.

After the connection is successfully established, the application may choose to read the connection PHY by calling the *Gap_LeReadPhy* API. It may also opt to change the PHY of the connection by triggering a PHY Update Procedure using the *Gap_LeSetPhy API*. However, the Controller might not be able to perform the change if, in case the peer does not support the new requested PHY.

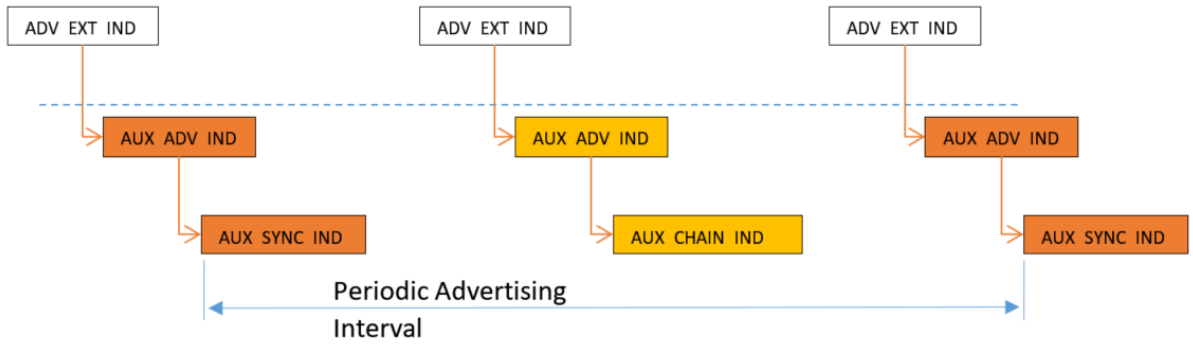
Parent topic: Extended advertising

Parent topic:*Generic Access Profile (GAP) Layer*

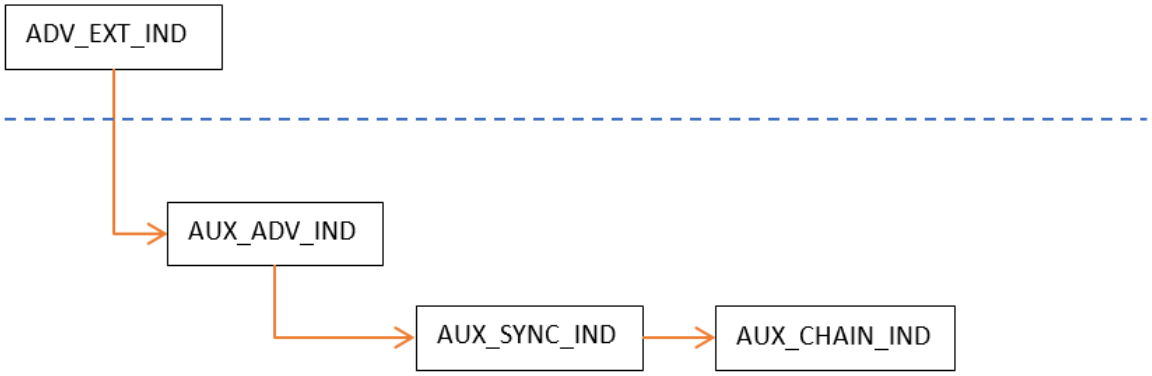
Periodic Advertising Periodic channels are used for periodic broadcast between unconnected devices. A periodic channel is represented by a channel map and a set of hopping and timing parameters.

The set of channels is represented by the 37 data channels. A packet sent by an advertiser can also have a payload of up to 255 bytes and it can be sent on any LE PHY. Figure 1 and Figure 2.

Extended Advertising and Periodic Advertising combined



Periodic Advertising – Multiple chains



Peripheral Setup

1. First set the extended advertising parameters using Gap_SetExtAdvertisingParameters. The extended advertising type must be set to non-connectable and non-scannable.
2. Set the periodic advertising parameters using the same handle as in the previous command.

```
bleResult_t Gap_SetPeriodicAdvParameters
(
    gapPeriodicAdvParameters_t*     pAdvertisingParameters
);
```

   Wait for a *gPeriodicAdvParamSetupComplete_c*event in the generic callback.

3. Next, set the periodic advertising data by calling:

```
bleResult_t Gap_SetPeriodicAdvertisingData
(
    uint8_t                     handle,
    gapAdvertisingData_t*   pAdvertisingData,
    bool_t bUpdateDID
);
```

   pAdvertisingData cannot be NULL. If periodic advertising data must be empty, set *cNumAdStructures* to 0. Wait for a *gPeriodicAdvDataSetupComplete_c*event in the generic callback.

4. Start extended advertising using Gap_StartExtAdvertising.
5. Last, enable Periodic Advertising. Periodic advertising starts only after extended advertising is started.

```
bleResult_t Gap_StartPeriodicAdvertising
(
    uint8_t handle,
    bool_t  bIncludeADI
);
```

   Wait for a *gPeriodicAdvertisingStateChanged_c*event in the advertising callback.

Parent topic: Periodic Advertising

Central Setup The application may decide to listen to periodic advertising by going through the following states:

1. [Optional] Add a known periodic advertiser to the periodic advertiser list held in the Controller by calling:

```
bleResult_t Gap_UpdatePeriodicAdvList
(
    gapPeriodicAdvListOperation_t operation,
    bleAddressType_t              addrType,
    uint8_t*                      pAddr,
    uint8_t                       SID
);
```

Wait for the *gPeriodicAdvListUpdateComplete_c* event in the Generic Callback.

2. Synchronize with a periodic advertiser by calling:

```
bleResult_t Gap_PeriodicAdvCreateSync
(
    gapPeriodicAdvSyncReq_t*    pReq,
);
```

pReq parameter *filterPolicy*can be set to *gUseCommandParameters_c* to synchronize with the given peer, or to *gUsePeriodicAdvList_c* to start synchronizing with all the devices in the previously populated periodic advertiser list.

Wait for the *gPeriodicAdvSyncEstablished_c* event and check the status. If scanning is not enabled at the time this command is sent, synchronization occurs after scanning is started. Synchronization remains pending until *gPeriodicAdvSyncEstablished_c*event is received. If synchronization was successful, the *syncHandle* is returned in this event.

3. Terminate the synchronization with the periodic advertiser by calling:

```
bleResult_t Gap_PeriodicAdvTerminateSync
(
    uint16_t syncHandle
);
```

To cancel a pending synchronization, the application should call *Gap_PeriodicAdvTerminateSync* with *syncHandle* set to the reserved value *gBlePeriodicAdvOngoingSyncCancelHandle* and wait for *gPeriodicAdvCreateSyncCancelled_c* event.

Otherwise, to terminate an already established sync with an advertiser, use the *syncHandle* value from the *gPeriodicAdvSyncEstablished_c* event and wait for a *gPeriodicAdvSyncTerminated_c* event.

Parent topic: Periodic Advertising

Parent topic:*Generic Access Profile (GAP) Layer*

Periodic Advertising with Responses (PAwR) This section describes the Central and Peripheral setup for Periodic Advertising with Responses (PAwR).

Central Setup

1. Start scanning using Gap_StartScanning. Wait for gPeriodicDeviceScannedV2_c events in the scanning callback.
2. Synchronize with a periodic advertiser by calling Gap_PeriodicAdvCreateSync. Wait for the gPeriodicAdvSyncEstablished_c event in the scanning callback. When PAwR is involved, this event includes additional information such as number of subevents, subevent interval, response slot delay and spacing,
3. Synchronize to a PAwR subevent by calling Gap_SetPeriodicSyncSubevent. This API instructs the Controller to sync with a subset of the subevents within a PAwR train identified by syncHandle (obtained after synchronizing with the PAwR train in the previous step).

```
bleResult_t Gap_SetPeriodicSyncSubevent ( uint16_t syncHandle, const␣
↪gapPeriodicSyncSubeventParameters_t* pParams );
```

Wait for the gPeriodicSyncSubeventComplete_c event.

4. Use Gap_SetPeriodicAdvResponseData to set data in the AD format which would be sent as a Periodic Advertising Response to the broadcaster.

```
bleResult_t Gap_SetPeriodicAdvResponseData ( uint16_t syncHandle, const␣
↪gapPeriodicAdvertisingResponseData_t* pData );
```

5. Optionally, the periodic advertiser may initiate a connection. If no connection callback was set on the scanner via APIs such as Gap_Connect or Gap_StartAdvertising/ Gap_StartExtAdvertising, one must be explicitly set. This is achieved by calling BluetoothLEHost_SetConnectionCallback (defined in app_conn.h), which in turn calls Gap_SetConnectionCallback.

```
void Gap_SetConnectionCallback ( gapConnectionCallback_t pfConnectionCallback );
```

Parent topic: Periodic Advertising with Responses (PAwR)

Peripheral Setup

1. First set the extended advertising parameters using Gap_SetExtAdvertisingParameters. The extended advertising type must be set to non-connectable and non-scannable.
2. Set the periodic advertising parameters with the same handle as in the previous command. Use the Gap_SetPeriodicAdvParametersV2 command. Compared to Gap_SetPeriodicAdvParameters, this command also configures parameters relevant to PAwR, such as the number of subevents and response slots as well as timing information.

```
bleResult_t Gap_SetPeriodicAdvParametersV2
  (gapPeriodicAdvParametersV2_t* pAdvertisingParameters);
```

Wait for a gPeriodicAdvParamSetupComplete_c event in the generic callback.

3. Start extended advertising using Gap_StartExtAdvertising.
4. Start periodic advertising using Gap_StartPeriodicAdvertising.
5. Wait for gPerAdvSubeventDataRequest_c events. These events are used by the Controller to indicate that it is ready to transmit one or more subevents and it is requesting the advertising data for these subevents. Upon receiving an event, use Gap_SetPeriodicAdvSubeventData to set the advertising data for specific subevents.

```
bleResult_t Gap_SetPeriodicAdvSubeventData
    (uint8_t advHandle, const gapPeriodicAdvertisingSubeventData_t* pData);
```

Wait for the gPeriodicAdvSetSubeventDataComplete_c event in the generic callback.

6. Wait for gPerAdvResponse_c events. These events contain responses sent by devices who are synchronized to the periodic advertising. They include data in the AD format.
7. Optionally, PAwR allows the advertising device to initiate a connection to one of the synchronized scanners. The connection can be initiated by calling Gap_ConnectFromPawr.

```
bleResult_t Gap_ConnectFromPawr
    (const gapConnectionFromPawrParameters_t* pParameters, gapConnectionCallback_t connCallback);
```

Parent topic: Periodic Advertising with Responses (PAwR)

Parent topic:*Generic Access Profile (GAP) Layer*

Encrypted Advertising Data This section describes the Central and Peripheral setup for encrypted advertising data.

Central Setup Use the Gap_DecryptAdvertisingData API to decrypt the contents of "Encrypted Advertising Data" (0x31) AD types included in scanned data.

```
bleResult_t Gap_DecryptAdvertisingData
   (uint8_t *pData, uint16_t dataLength, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)
```

Parent topic: Encrypted Advertising Data

Peripheral Setup Use the Gap_EncryptAdvertisingData API to obtain the encrypted advertising data, which can then be placed inside the "Encrypted Advertising Data" (0x31) AD type.

```
bleResult_t Gap_EncryptAdvertisingData
  ( const gapAdvertisingData_t *pAdvertisingData, const uint8_t *pKey, const uint8_t *pIV, uint8_t␣
↪*pOutput )
```

Parent topic: Encrypted Advertising Data

Parent topic:*Generic Access Profile (GAP) Layer*

L2CAP credit-based channels The L2CAP layer, which is responsible for protocol multiplexing, segmentation, and reassembly operations, allows devices to communicate via connection-oriented channels. These channels use credit-based flow control, in which a device grants each peer a number of credits which the peer can use to send packets. The number of credits is decremented with every sent packet. A device can grant more credits to its peers over the duration of the connection.

Unlike the fixed L2CAP CIDs used by protocols such as ATT and SMP, credit-based channels use dynamically allocated CIDS (in the 0x0040-0xFFFF range). The CIDs are automatically allocated by the Bluetooth LE Host Stack.

The Bluetooth LE Host Stack supports both the Credit-based Flow Control Mode and the Enhanced Credit-based Flow Control Mode. In the Enhanced Credit-based Flow Control Mode, devices can open up to five channels in a single connect request/response exchange. Additionally, these channels can be later reconfigured with new MTU and MPS values. In the Credit-based Flow Control Mode, reconfiguration is not possible.

The first thing an application must do is register the control and data callbacks:

```
bleResult_t   L2ca_RegisterLeCbCallbacks
(
   l2caLeCbDataCallback_t      pCallback,
   l2caLeCbControlCallback_t   pCtrlCallback
);
```

The control callback receives events related to channel management such as connection, disconnection, received credits, reconfiguration, and so on.

The data callback receives the data which is being exchanged on the channel.

To use L2CAP credit-based channels, the application must register a PSM. The PSM is analogous to a TCP/UDP port. It is an identifier used to determine the upper layer protocol which is making use of the L2CAP channel. The dynamic PSM range is 0x0080-0x00FF. The number of PSMs supported by an application can be configured at compile time via the gL2caMaxLePsmSupported_c define. The following API must be called in order to register a PSM:

```
bleResult_t  L2ca_RegisterLePsm
(
  uint16_t     lePsm,
  uint16_t     lePsmMtu
);
```

The MTU configured via this API is used by every channel opened under the PSM, if the Credit-based Flow Control Mode is used. The minimum MTU is 23 and the maximum MTU is 65535.

When the Enhanced Credit-based Flow Control Mode is used, the MTU is specified at each connection request. In this mode, the minimum MTU is 64 and the maximum MTU is 65535.

The local MPS is not configurable by the application. It is set automatically by the Host Stack based on Controller capabilities. Usually, it will be 247.

A previously registered PSM can be deregistered:

```
bleResult_t L2ca_DeregisterLePsm
(
 uint16_t     lePsm
);
```

The number of credit-based channels that can be opened is configurable by the application via the gL2caMaxLeCbChannels_c define. This is the total number for all peers. To open a channel, the following API must be called:

```
bleResult_t L2ca_ConnectLePsm
(
  uint16_t     lePsm,
  deviceId_t   deviceId,
  uint16_t     initialCredits
);
```

To open up to five channels using Enhanced Credit-based Flow Control Mode, use this API:

```
bleResult_t  L2ca_EnhancedConnectLePsm
(
 uint16_t     lePsm,
 deviceId_t   deviceId,
 uint16_t     mtu,
 uint16_t     initialCredits,
 uint8_t      noOfChannels,
 uint16_t     *aCids
);
```

The connect APIs must be called by both the initiator and the responder (upon receiving the gL2ca_LePsmConnectRequest_c or gL2ca_LePsmEnhancedConnectRequest_c events in the application).

If the responder does not wish to accept the connection request, it can use the following APIs:

```
bleResult_t  L2ca_CancelConnection
   (
     uint16_t      lePsm,
     deviceId_t    deviceId,
    l2caLeCbConnectionRequestResult_t refuseReason
   );
bleResult_t L2ca_EnhancedCancelConnection
   (
   uint16_t lePsm,
   deviceId_t deviceId,
   l2caLeCbConnectionRequestResult_t refuseReason,
```

(continues on next page)

(continued from previous page)

```
    uint8_t noOfChannels,
    uint16_t *aCids
    );
```

When a channel has been successfully established, the gL2ca_LePsmConnectionComplete_c or gL2ca_LePsmEnhancedConnectionComplete_c events are received in the application.

To send data on a channel:

```
bleResult_t L2ca_SendLeCbData
(
  deviceId_t      deviceId,
  uint16_t        channelId,
  const uint8_t*  pPacket,
  uint16_t        packetLength
);
```

The Host Stack keeps track of the credits granted to peers for each channel and decrements them accordingly. When a peer's credit count reaches zero, the application is notified through the gL2ca_NoPeerCredits_c event and it can decide to send more credits to the peer for that channel:

```
bleResult_t L2ca_SendLeCredit
(
 deviceId_t     deviceId,
 uint16_t       channelId,
 uint16_t       credits
);
```

The application can also choose to be notified when the number of credits allocated to a peer for a certain channel is nearing 0, by setting the gL2caLowPeerCreditsThreshold_c macro to a non-zero value. When this limit is reached, the gL2ca_LowPeerCredits_c event is received and the application can choose to send more credits.

Similarly, when a device receives credits from a peer, the application is notified through the gL2ca_LocalCreditsNotification_c event. When the local device has used its last credit, it receives the same gL2ca_LocalCreditsNotification_c event with the localCredits field set to 0. The packet that could not be sent due to exhausting the credits remains queued in the Host Stack and it is sent automatically when the local device receives credits from the peer.

To improve application flow control, two notification-type events are implemented by the Host Stack:

- gL2ca_ChannelStatusChannelBusy_c
- gL2ca_ChannelStatusChannelIdle_c

When the application sends a packet using L2ca_SendLeCbData, it receives a gL2ca_ChannelStatusChannelBusy_c event in the L2CAP control callback when the Host Stack begins sending the packet. When the Host Stack has sent the packet, a gL2ca_ChannelStatusChannelIdle_c event is received. The application can choose to use this event as a signal that it is safe to send the next packet.

To disconnect a channel:

```
bleResult_t L2ca_DisconnectLeCbChannel
(
   deviceId_t  deviceId,
   uint16_t    channelId
);
```

As mentioned previously, channels which use the Enhanced Credit-based Flow Control Mode can be reconfigured. This is achieved via the API:

```
bleResult_t L2ca_EnhancedChannelReconfigure
(
    deviceId_t      deviceId,
    uint16_t        newMtu,
    uint16_t        newMps,
    uint8_t         noOfChannels,
    uint16_t        *aCids
);
```

The reconfiguration request is automatically accepted by the Host Stack if parameters are valid (as per the Bluetooth Core Spec v5.3, MTU cannot be lowered and MPS cannot be lowered for more than one channel). On the responder, the gL2ca_EnhancedReconfigureRequest_c is received by the application in case of a successful reconfiguration, informing it of the new channel parameters. On the initiator, the gL2ca_EnhancedReconfigureResponse_c is received, informing the application about the received response or a timeout.

Parent topic:*Generic Access Profile (GAP) Layer*

Enhanced ATT The Enhanced ATT protocol allows concurrent transactions to be handled by the stack. The sequential transaction rule still exists when EATT is used, but its scope is now defined as being per instance of the Enhanced ATT Bearer. EATT transactions might execute in parallel if they are supported by distinct L2CAP channels, which use the Enhanced Credit Based Flow Control Mode (that is, distinct Enhanced ATT Bearers).

When using an Enhanced ATT Bearer, ATT MTU and L2CAP MTU are independently configurable and may be reconfigured during a connection. An increase to the MTU is allowed but reducing its size is not. Allowing MTU to be increased without needing to reestablish the connection has an advantage. It eliminates the risk of a second application using the stack, being unable to continue, due to the previously negotiated MTU being too small.

Enhanced ATT bearers are identified through Bearer Ids. Enhanced ATT Bearer Ids are assigned internally and have a valid range between 1 and 251. The Unenhanced ATT bearer is always available for a connected peer device and has the *BearerId 0*.

EATT Credits management Credits for the L2CAP channels used by Enhanced ATT bearers may be managed internally if the autoCreditsMgmt parameter is set to TRUE in the Gap_EattConnectionRequest or Gap_EattConnectionAccept function call. Otherwise, the application is responsible for credits management.

If the application chooses to manage the credits of the L2CAP channels used as Enhanced ATT bearers, it should use the following function to send credits for a specified bearer to a peer device:

```
bleResult_t Gap_EattSendCredits
(
    deviceId_t  deviceId,
    bearerId_t  bearerId,
    uint16_t    credits
);
```

If the local credits or peer credits of the L2CAP channel used by an Enhanced ATT bearer reaches 0, agConnEvtEattBearerStatusNotification_c connection event is updated with a status value ofgEnhancedBearerSuspendedNoLocalCredits_c, or gEnhancedBearerNoPeerCredits_c respectively.

Parent topic: Enhanced ATT

EATT Connection establishment In order to take advantage of the Enhanced ATT features, first a number of Enhanced Bearers should be opened for a connected peer device. For this, the

function below may be used to create up to five Enhanced ATT bearers at a time:

```
bleResult_t Gap_EattConnectionRequest
(
    deviceId_t  deviceId,
    uint16_t    mtu,
    uint8_t     cBearers,
    uint16_t    initialCredits,
    bool_t      autoCreditsMgmt
);
```

The mtu parameter specifies the MTU for all the bearers to be established.

The cBearers parameter is used to specify the number of Enhanced ATT bearers to be opened, and should have a value between 1 and 5. The initialCredits parameter specifies the initial number of credits of the L2CAP credit based channels used as Enhanced ATT bearers.

The autoCreditsMgmt parameter is used to tell the Bluetooth LE Host Stack if it should manage L2CAP channel credits automatically. If set to TRUE the Bluetooth LE Host Stack automatically sends credits to a peer device when exhausted in chunks of initialCredits.

For example, to establish two Enhanced ATT bearers with a peer device the application may call the Gap_EattConnectionRequest as shown below:

```
bleResult_t result = Gap_EattConnectionRequest(peerDeviceId,
                                               64U,
                                               2U,
                                               3U,
                                               TRUE);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

If an EATT Connection Request is received from a peer device it would be signaled through the gConnEvtEattConnectionRequest_c connection event of type gapEattConnectionRequest_t sent to the connection callback. The application should handle this event by calling Gap_EattConnectionAccept. The example below shows how an application may accept an incoming EATT Connection Request with the same MTU as requested by the peer device.

```
case gConnEvtEattConnectionRequest_c:
{
    gapEattConnectionRequest_t *pEattConnectionReq = &pConnectionEvent->eventData.
↪eattConnectionRequest;

    bleResult_t result = Gap_EattConnectionAccept(peerDeviceId,
                                                  TRUE,
                                                  pEattConnectionReq->mtu,
                                                  3U,
                                                  TRUE);

    if (gBleSuccess_c != result)
    {
        /* Treat error */
    }
}
break;
```

In case the localMtu specified when accepting a connection differs from the MTU requested by the peer device, the minimum of the two would become the MTU of the Enhanced Bearers.

After the Gap_EattConnectionRequest or Gap_EattConnectionAccept is called, for the result the application should wait for the gConnEvtEattConnectionComplete_c connection event of type gapEattConnectionComplete_t shown below:

```
typedef struct {
    l2caLeCbConnectionRequestResult_t           status;
    uint16_t                                    mtu;
    uint8_t                                     cBearers;
    bearerId_t                                  aBearerIds[gGapEattMaxBearers];
} gapEattConnectionComplete_t;
```

If successful, the aBearerIds array contains the bearer ids, for the Enhanced ATT bearers established. These ids may be used with the GATT Enhanced APIs in order to trigger GATT procedures over Enhanced ATT bearers.

Parent topic: Enhanced ATT

EATT Bearer reconfiguration One of the advantages of Enhanced ATT bearers over the Unenhanced ATT bearer is the ability to increase the MTU multiple times. To reconfigure the MTU and/or MPS of existing Enhanced ATT bearers, the Gap_EattReconfigureRequest should be used. If a mps value of 0 is given, the maximum available MPS value for that channel is used.

For example, in order to reconfigure the MTU of two bearers from 64 to 128 the application may call Gap_EattReconfigureRequest as shown below:

```
bleResult_t result = gBleSuccess_c;
bearerId_t aBearerIds[2] = {1U, 2U};
result = Gap_EattReconfigureRequest(peerDeviceId,
                                    128U,
                                    0U,
                                    2U,
                                    aBearerIds);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The application should monitor the gConnEvtEattChannelReconfigureResponse_c connection event of type gapEattReconfigureResponse_t for the result.

The procedure triggered by Gap_EattReconfigureRequest updates only the local MTU. The ATT_MTU for Enhanced ATT bearers is the minimum of the MTU values of the two devices.

Parent topic: Enhanced ATT

EATT Bearer disconnection Individual Enhanced ATT bearers can be disconnected by calling the Gap_EattDisconnect API as shown below:

```
bleResult_t result = gBleSuccess_c;
result = Gap_EattDisconnect(peerDeviceId, bearerId);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The application should look for a connection event of type gEnhancedBearerDisconnected_c in the connection callback.

Parent topic: Enhanced ATT

Parent topic:*Generic Access Profile (GAP) Layer*

Generic Attribute Profile (GATT) Layer The GATT layer contains the APIs for discovering services and characteristics and transferring data between devices and is built on top of the Attribute Protocol (ATT).

The Attribute Protocol (ATT) transfers data between Bluetooth Low Energy devices on a dedicated L2CAP channel (channel ID 0x04).

As soon as a connection is established between devices, the GATT APIs are readily available. No initialization is required because the L2CAP channel is automatically created.

To identify the GATT peer instance, the same *deviceId* value from the GAP layer (obtained in the *gConnEvtConnected_c*connection event) is used.

There are two GATT roles that define the two devices exchanging data over ATT:

- GATT Server – the device that contains a GATT Database, which is a collection of services and characteristics exposing meaningful data. Usually, the Server responds to *requests* and *commands* sent by the Client. However, it can be configured to send data on its own through *notifications*and *indications*.
- GATT Client – the "active" device that usually sends *requests* and *commands* to the Server to *discover* Services and Characteristics on the Server's Database and to exchange data.

There is no fixed rule deciding which device is the Client and which one is the Server. Any device may initiate a request at any moment. Therefore, it temporarily acts as a Client, at which the peer device may respond, provided it has the Server support and a GATT Database.

Often, a GAP Central acts as a GATT Client to discover Services and Characteristics and obtain data from the GAP Peripheral, which usually has a GATT database. Many standard Bluetooth Low Energy profiles assume that the Peripheral has a database and must act as a Server. However, this is by no means a general rule.

Client APIs A Client can configure the ATT MTU, discover Services and Characteristics, and initiate data exchanges.

All the functions have the same first parameter: a *deviceId* which identifies the connected device whose GATT Server is targeted in the GATT procedure. This is necessary because a Client may be connected to multiple Servers at the same time.

First, however, the application must install the necessary callbacks.

Installing client callbacks There are three callbacks that the Client application must install.

Client procedure callback All the procedures initiated by a Client are asynchronous. They rely on exchanging ATT packets over the air.

To be informed of the procedure completion, the application must install a callback with the following signature:

```
typedef void (* gattClientProcedureCallback_t )
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
);
```

For EATT, the following signature should be used:

```
typedef void (*gattClientEnhancedProcedureCallback_t)
(
  deviceId_t deviceId,
  bearerId_t bearerId,
  gattProcedureType_t procedureType,
  gattProcedureResult_t procedureResult,
  bleResult_t error
);
```

To install this callback, the following function must be called:

```
bleResult_t GattClient_RegisterProcedureCallback
(
   gattClientProcedureCallback_t callback
);
```

The EATT procedure callback should be installed using the following API:

```
bleResult_t GattClient_RegisterEnhancedProcedureCallback
(
 gattClientEnhancedProcedureCallback_t callback
);
```

The *procedureType* parameter can be used to identify the procedure that was started and has reached completion. Only one procedure would be active at a given moment. Trying to start another procedure while a procedure is already in progress returns the error *gGattAnotherProcedureInProgress_c*.

The *procedureResult* parameter indicates whether the procedure completes successfully or an error occurs. In the latter case, the *error* parameter contains the error code.

```
void gatt ClientProcedureCallback
(
   deviceId_t             deviceId,
   gattProcedureType_t    procedureType,
   gattProcedureResult_t  procedureResult,
   bleResult_t            error
)
{
   switch (procedureType)
   {
      /* ... */
   }
}
GattClient_RegisterProcedureCallback(gattClientProcedureCallback);
```

Parent topic: Installing client callbacks

Notification and indication callbacks When the Client receives a notification from the Server, it triggers a callback with the following prototype:

```
typedef void (* gattClientNotificationCallback_t )
(
   deviceId_t     deviceId,
   uint16_t       characteristicValueHandle,
   uint8_t *      aValue,
   uint16_t       valueLength
);
```

The *deviceId* identifies the Server connection (for multiple connections at the same time). The *characteristicValueHandle* is the attribute handle of the Characteristic Value declaration in the GATT Database. The Client must have discovered it previously to be able recognize it.

For EATT, the following signature should be used:

```
typedef void (*gattClientEnhancedNotificationCallback_t)
 ( deviceId_t deviceId,
   bearerId_t bearerId,
   uint16_t characteristicValueHandle,
   uint8_t* aValue,
   uint16_t valueLength );
```

The callback must be installed with:

```
bleResult_t GattClient_RegisterNotificationCallback
(
    gattClientNotificationCallback_t callback
);
```

Very similar definitions exist for indications.

The EATT notification callback should be installed using the following API:

```
bleResult_t GattClient_RegisterEnhancedNotificationCallback
(
 gattClientEnhancedNotificationCallback_t callback
)
```

When receiving a notification or indication, the Client uses the *characteristicValueHandle* to identify which Characteristic was notified. The Client must be aware of the possible Characteristic Value handles that can be notified/indicated at any time, because it has previously activated them by writing its CCCD (see Reading and writing characteristic descriptors).

When the Client receives a multiple value notification from the Server, it triggers a callback with the following prototype:

```
typedef void (*gattClientMultipleValueNotificationCallback_t)
(
    deviceId_t      deviceId,
    /*!< Device ID identifying the active connection. */
    uint8_t* aHandleLenValue,
    /*!< The array of handle, value length, value tuples. */
    uint32_t totalLength
    /*!< Value array size. */
);
```

The callback must be installed with:

```
bleResult_t GattClient_RegisterMultipleValueNotificationCallback
(
    gattClientMultipleValueNotificationCallback_t callback
);
```

When using EATT, the following callback prototype and registration APIs should be used:

```
typedef void (*gattClientEnhancedMultipleValueNotificationCallback_t)
( deviceId_t deviceId,
    /*!< Device ID identifying the active connection. */
  bearerId_t bearerId,
   /*!< Bearer ID identifing the Enhanced ATT bearer used. */
  uint8_t* aHandleLenValue,
   /*!< The array of handle, value length, value tuples. */
  uint32_t totalLength      /*!< Value array size. */
 );
```

Parent topic: Installing client callbacks

Parent topic: Client APIs

MTU exchange A radio packet sent over the Bluetooth Low Energy contains a maximum of 27 bytes of data for the L2CAP layer. The L2CAP header is 4 bytes long, including the Channel ID. Therefore, all layers above L2CAP, including ATT and GATT, can only send 23 bytes of data in a radio packet (as per *Bluetooth 4.1 Specification for Bluetooth Low Energy*). This specification also sets the default length of an ATT packet (also called *ATT_MTU*) to 23. The ATT packet length is set to this value to maintain a logical mapping between radio packets and ATT packets.

Note: This number is fixed and cannot be increased in Bluetooth Low Energy 4.1.

Therefore, any ATT request fits in a single radio packet. If the layer above ATT wishes to send more than 23 bytes of data, the data must be fragmented into smaller packets and multiple ATT requests issued.

Despite this setting, the ATT protocol allows devices to increase the ATT_MTU, only if both can support it. Increasing the ATT_MTU has only one effect: the application does not have to fragment long data. However, it can send more than 23 bytes in a single transaction. The fragmentation is moved on to the L2CAP layer. Over the air though, there would still be more than one radio packet sent.

If the GATT Client supports a larger than default MTU, it must start an MTU exchange as soon as it connects to any server. During the MTU exchange, both devices would send their maximum MTU to the other, and the minimum of the two is chosen as the new MTU.

Consider an example where the Client supports a maximum ATT_MTU of 250, and the server supports a maximum value of 120 for the same attribute. For this case, after MTU exchange, both devices must set the new ATT_MTU value equal to 120.

To initiate the MTU exchange, call the following function from *gatt_client_interface.h*:

```
bleResult_t result = GattClient_ExchangeMtu(deviceId, mtu);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

When having the role of a GATT Client, the value of the maximum supported *ATT_MTU* of the local device is given as a parameter to the GattClient_ExchangeMtu API. On the GATT Server side, the application configures this value via the gcGattServerMtu_c variable that exists in the file ble_globals.c. The minimum of these two values is chosen as the new MTU for the connection.

When the exchange is complete, the *gGattProcExchangeMtu_c* procedure type triggers the Client callback.

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcExchangeMtu_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* To obtain the new MTU */
                uint16_t newMtu;
                bleResult_t result = Gatt_GetMtu(deviceId, &newMtu);
                if (gBleSuccess_c == result)
                {
                    /* Use the value of the new MTU */
```

(continues on next page)

(continued from previous page)

```
                (void) newMtu;
            }
        }
        else
        {
            /* Handle error */
        }
        break;
    /* ... */
  }
}
```

Note: The Exchange MTU procedure is only available for the unenhanced/legacy bearer. For procedures sent on enhanced bearers, the upper layer must use provided **L2CAP APIs** to create dedicated L2CAP channels. Each channel has its own MTU value specified upon creation, which can also be reconfigured later.

Parent topic: Client APIs

Service and characteristic discovery There are multiple APIs that can be used for Discovery. The application may use any of them, according to its necessities.

All of the following APIs have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearerId*parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer Id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

Discover all primary services The following API can be used to discover all the Primary Services in a Server's database:

```
bleResult_t GattClient_DiscoverAllPrimaryServices
(
    deviceId_t          deviceId,
    gattService_t *     aOutPrimaryServices,
    uint8_t             maxServiceCount,
    uint8_t *           pOutDiscoveredCount
);
```

The *aOutPrimaryServices* parameter must point to an allocated array of services. The size of the array must be equal to the value of the *maxServiceCount* parameter, which is passed to make sure the GATT module does not attempt to write past the end of the array if more Services are discovered than expected.

The *pOutDiscoveredCount* parameter must point to a static variable because the GATT module uses it to write the number of Services discovered at the end of the procedure. This number is less than or equal to the *maxServiceCount*.

If there is equality, it is possible that the Server contains more than *maxServiceCount* Services, but they could not be discovered as a result of the array size limitation. It is the application developer's responsibility to allocate a large enough number according to the expected contents of the Server's database.

In the following example, the application expects to find no more than 10 Services on the Server.

```
#define mcMaxPrimaryServices_c 10
static gattService_t primaryServices[mcMaxPrimaryServices_c];
uint8_t mcPrimaryServices;
bleResult_t result = GattClient_DiscoverAllPrimaryServices
```

(continues on next page)

(continued from previous page)

```
(
    deviceId,
    primaryServices,
    mcMaxPrimaryServices_c,
    &mcPrimaryServices
);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The operation triggers the Client Procedure Callback when complete. The application may read the number of discovered services and each service's handle range and UUID.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverAllPrimaryServices_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read number of discovered services */
                PRINT( mcPrimaryServices );
                /* Read each service's handle range and UUID */
                for (int j = 0; j < mcPrimaryServices; j++)
                {
                    PRINT( primaryServices[j]. startHandle );
                    PRINT( primaryServices[j]. endHandle );
                    PRINT( primaryServices[j]. uuidType );
                    PRINT( primaryServices[j]. uuid );
                }
            }
            else
            {
                /* Handle error */
                PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover primary services by UUID To discover only Primary Services of a known type (Service UUID), the following API can be used:

```
bleResult_t GattClient_DiscoverPrimaryServicesByUuid
(
    deviceId_t          deviceId,
    bleUuidType_t       uuidType,
    const bleUuid_t *   pUuid,
    gattService_t *     aOutPrimaryServices,
```

(continues on next page)

(continued from previous page)

```
    uint8_t             maxServiceCount,
    uint8_t *           pOutDiscoveredCount
);
```

The procedure is very similar to the one described in Discover all primary services. The only difference is this time we are filtering the search according to a Service UUID described by two extra parameters: *pUuid* and *uuidType*.

This procedure is useful when the Client is only interested in a specific type of Services. Usually, it is performed on Servers that are known to contain a certain Service, which is specific to a certain profile. Therefore, most of the times the search is expected to find a single Service of the given type. As a result, only one structure is usually allocated.

For example, when two devices implement the Heart Rate (HR) Profile, an HR Collector connects to an HR Sensor and may only be interested in discovering the Heart Rate Service (HRS) to work with its Characteristics. The following code example shows how to achieve this. Standard values for Service and Characteristic UUIDs, as defined by the Bluetooth SIG, are located in the *ble_sig_defines.h* file.

```
static gattService_t heartRateService;
static uint8_t mcHrs;
bleResult_t result = GattClient_DiscoverPrimaryServicesByUuid
(
    deviceId,
    gBleUuidType16_c,             /* Service UUID type */
    gBleSig_HeartRateService_d,   /* Service UUID */
    &heartRateService,            /* Only one HRS is expected to be found */
    1,
    &mcHrs
   /* Will be equal to 1 at the end of the procedure
    if the HRS is found, 0 otherwise */

);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

In the Client Procedure Callback, the application should check if any Service with the given UUID was found and read its handle range (also perhaps proceed with Characteristic Discovery within that service range).

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverPrimaryServicesByUuid_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                if (1 == mcHrs)
                {
                    /* HRS found, read the handle range */
                    PRINT( heartRateService. startHandle );
                    PRINT( heartRateService. endHandle );
```

(continues on next page)

(continued from previous page)

```
            }
            else
            {
                /* HRS not found! */
            }
        }
        else
        {
            /* Handle error */
            PRINT( error );
        }
        break;
    /* ... */
  }
}
```

Parent topic: Service and characteristic discovery

Discover included services Discover all primary services shows how to discover Primary Services. However, a Server may also contain Secondary Services, which are not meant to be used standalone and are usually included in the Primary Services. The inclusion means that all the Secondary Service's Characteristics may be used by the profile that requires the Primary Service.

Therefore, after a Primary Service has been discovered, the following procedure may be used to discover services (usually Secondary Services) included in it:

```
bleResult_t GattClient_FindIncludedServices
(
  deviceId_t          deviceId,
  gattService_t *     pIoService,
  uint8_t             maxServiceCount
);
```

The service structure that *pIoService* points to must have the *aIncludedServices* field linked to an allocated array of services, of size *maxServiceCount*, chosen according to the expected number of included services to be found. This is the application's choice, usually following profile specifications.

Also, the service's range must be set (the *startHandle* and *endHandle* fields), which may have already been done by the previous Service Discovery procedure (as described in Discover all primary services and Discover primary services by UUID).

The number of discovered included services is written by the GATT module in the *cNumIncludedServices* field of the structure from *pIoService*. Obviously, a maximum of *maxServiceCount* included services is discovered.

The following example assumes the Heart Rate Service was discovered using the code provided in Discover primary services by UUID.

```
/* Finding services included in the Heart Rate Primary Service */
gattService_t * pPrimaryService = &heartRateService;
#define mxMaxIncludedServices_c 3
static gattService_t includedServices[mxMaxIncludedServices_c];
/* Linking the array */
pPrimaryService-> aIncludedServices = includedServices;
bleResult_t result = GattClient_FindIncludedServices
(
  deviceId,
  pPrimaryService,
  mxMaxIncludedServices_c
```

(continues on next page)

(continued from previous page)

```
);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

When the Client Procedure Callback is triggered, if any included services are found, the application can read their handle range and their UUIDs.

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcFindIncludedServices_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read included services data */
                PRINT( pPrimaryService-> cNumIncludedServices );
                for (int j = 0; j < pPrimaryService-> cNumIncludedServices ; j++)
                {
                    PRINT( pPrimaryService-> aIncludedServices [j]. startHandle );
                    PRINT( pPrimaryService-> aIncludedServices [j]. endHandle );
                    PRINT( pPrimaryService-> aIncludedServices [j]. uuidType );
                    PRINT( pPrimaryService-> aIncludedServices [j]. uuid );
                }
            }
            else
            {
                /* Handle error */
                PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover all characteristics of a service The main API for Characteristic Discovery has the following prototype:

```
bleResult_t GattClient_DiscoverAllCharacteristicsOfService
(
    deviceId_t          deviceId,
    gattService_t *     pIoService,
    uint8_t             maxCharacteristicCount
);
```

All required information is contained in the service structure pointed to by *pIoService*, most importantly being the service range (*startHandle* and *endHandle*) which is usually already filled out by a Service Discovery procedure. If not, they need to be written manually.

Also, the service structure's *aCharacteristics* field must be linked to an allocated characteristic array.

The following example discovers all Characteristics contained in the Heart Rate Service discovered in Section Discover primary services by UUID.

```
gattService_t* pService = &heartRateService
#define mcMaxCharacteristics_c 10
static gattCharacteristic_t hrsCharacteristics[mcMaxCharacteristics_c];
pService->aCharacteristics = hrsCharacteristics;
bleResult_t result = GattClient_DiscoverAllCharacteristicsOfService
(
    deviceId,
    pService,
    mcMaxCharacteristics_c
);
```

The Client Procedure Callback is triggered when the procedure completes.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
      /* ... */
      case gGattProcDiscoverAllCharacteristics_c:
          if (gGattProcSuccess_c == procedureResult)
          {
            /* Read number of discovered Characteristics */
              PRINT(pService-> cNumCharacteristics );
            /* Read discovered Characteristics data */
              for ( uint8_t j = 0; j < pService-> cNumCharacteristics ; j++)
                {
            /* Characteristic UUID is found inside the value field
               to avoid duplication */
                  PRINT(pService-> aCharacteristics [j]. value . uuidType );
                  PRINT(pService-> aCharacteristics [j]. value . uuid );
            /* Characteristic Properties indicating the supported operations:
                   * - Read
                   * - Write
                   * - Write Without Response
                   * - Notify
                   * - Indicate
            */
                  PRINT(pService-> aCharacteristics [j]. properties );
           /* Characteristic Value Handle is used to identify the
              Characteristic in future operations */
                  PRINT(pService-> aCharacteristics [j]. value . handle );
                }
            }
            else
            {
               /* Handle error */
               PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover characteristics by UUID This procedure is useful when the Client intends to discover a specific Characteristic in a specific Service. The API allows for multiple Characteristics of the same type to be discovered, but most often it is used when a single Characteristic of the given type is expected to be found.

Continuing the example from Discover primary services by UUID, assume the Client wants to discover the Heart Rate Control Point Characteristic inside the Heart Rate Service, as shown in the following code.

```
gattService_t * pService = &heartRateService;
static gattCharacteristic_t hrcpCharacteristic;
static uint8_t mcHrcpChar;
bleUuid_t hrcpUuid;
hrcpUuid.uuid16 = gBleSig_HrControlPoint_d;
bleResult_t result = GattClient_DiscoverCharacteristicOfServiceByUuid
(
    deviceId,
    gBleUuidType16_c,
    &hrcpUuid,
    pService,
    &hrcpCharacteristic,
    1,
    &mcHrcpChar
);
```

This API can be used as in the previous examples, following a Service Discovery procedure. However, the user may want to perform a Characteristic search with UUID over the entire database, skipping the Service Discovery entirely. To do so, a dummy service structure must be defined and its range must be set to maximum, as shown in the following example:

```
gatt Service_t dummyService;
dummyService. startHandle = 0x0001;
dummyService. endHandle = 0xFFFF;
static gattCharacteristic_t hrcpCharacteristic;
static uint8_t mcHrcpChar;
bleResult_t result = GattClient_DiscoverCharacteristicOfServiceByUuid
(
    deviceId,
    gBleUuidType16_c,
    gBleSig_HrControlPoint_d,
    &dummyService,
    &hrcpCharacteristic,
    1,
    &mcHrcpChar
);
```

.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverCharacteristicByUuid_c:
            if (gGattProcSuccess_c == procedureResult)
            {
```

(continues on next page)

(continued from previous page)

```
            if (1 == mcHrcpChar)
            {
                /* HRCP found, read discovered data */
                PRINT(hrcpCharacteristic. properties );
                PRINT(hrcpCharacteristic. value . handle );
            }
            else
            {
                /* HRCP not found! */
            }
        }
        else
        {
            /* Handle error */
            PRINT(error);
        }
        break;
    /* ... */
  }
}
```

Parent topic: Service and characteristic discovery

Discover characteristic descriptors To discover all descriptors of a Characteristic, the following API is provided:

```
bleResult_t GattClient_DiscoverAllCharacteristicDescriptors
(
  deviceId_t                deviceId,
  gattCharacteristic_t *    pIoCharacteristic,
  uint16_t                  endingHandle,
  uint8_t                   maxDescriptorCount
);
```

The *pIoCharacteristic* pointer must point to a Characteristic structure with the *value.handle* field set (either by a discovery operation or by the application) and the *aDescriptors* field pointed to an allocated array of Descriptor structures.

The *endingHandle* should be set to the handle of the next Characteristic or Service declaration in the database to indicate when the search for descriptors must stop. The GATT Client module uses ATT Find Information Requests to discover the descriptors, and it does so until it discovers a Characteristic or Service declaration or until *endingHandle* is reached. Thus, by providing a correct ending handle, the search for descriptors is optimized and the number of packets sent over the air is reduced.

If, however, the application does not know where the next declaration lies and cannot provide this optimization hint, the *endingHandle* should be set to *0xFFFF*.

Continuing the example from Discover characteristics by UUID, the following code assumes that the Heart Rate Control Point Characteristic has no more than 5 descriptors and performs Descriptor Discovery.

```
#define mcMaxDescriptors_c 5
static gattAttribute_t aDescriptors[mcMaxDescriptors_c];
hrcpCharacteristic. aDescriptors = aDescriptors;
bleResult_t result = GattClient_DiscoverAllCharacteristicDescriptors
(
  deviceId,
  &hrcpCharacteristic,
  0xFFFF, /* We do not know where the next Characterstic Service begins */
```

(continues on next page)

(continued from previous page)

```
    mcMaxDescriptors_c
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered at the end of the procedure.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverAllCharacteristicDescriptors_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read number of discovered descriptors */
                PRINT(hrcpCharacteristic. cNumDescriptors );
                /* Read descriptor data */
                for ( uint8_t j = 0; j < hrcpCharacteristic. cNumDescriptors ; j++)
                {
                    PRINT(hrcpCharacteristic. aDescriptors [j]. handle );
                    PRINT(hrcpCharacteristic. aDescriptors [j]. uuidType );
                    PRINT(hrcpCharacteristic. aDescriptors [j]. uuid );
                }
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Parent topic: Client APIs

Reading and writing characteristics All the APIs described in the following sections have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearer id* parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

Characteristic value read procedure The main API for reading a Characteristic Value is shown here:

```
bleResult_t GattClient_ReadCharacteristicValue
(
    deviceId_t                  deviceId,
```

(continues on next page)

(continued from previous page)

```
    gattCharacteristic_t *      pIoCharacteristic,
    uint16_t                    maxReadBytes
);
```

This procedure assumes that the application knows the Characteristic Value Handle, usually from a previous Characteristic Discovery procedure. Therefore, the *value.handle* field of the structure pointed by *pIoCharacteristic* must be completed.

Also, the application must allocate a large enough array of bytes where the received value (from the ATT packet exchange) is written. The *maxReadBytes* parameter is set to the size of this allocated array.

The GATT Client module takes care of long characteristics, whose values have a greater length than can fit in a single ATT packet, by issuing repeated ATT Read Blob Requests when needed.

The following examples assume that the application knows the Characteristic Value Handle and that the value length is variable, but limited to 50 bytes.

```
gattCharacteristic_t myCharacteristic;
myCharacteristic.value.handle = 0x10AB;
#define mcMaxValueLength_c 50
static uint8_t aValue[mcMaxValueLength_c];
myCharacteristic.value.paValue = aValue;
bleResult_t result = GattClient_ReadCharacteristicValue
(
    deviceId,
    &myCharacteristic,
    mcMaxValueLength_c
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

Regardless of the value length, the Client Procedure Callback is triggered when the reading is complete. The received value length is also filled in the *value* structure.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcReadCharacteristicValue_c:
            if (*gGattProcSuccess_c == procedureResult)
            {
                /* Read value length */
                PRINT(myCharacteristic.value.valueLength );
                /* Read data */
                for ( uint16_t j = 0; j < myCharacteristic.value.valueLength ; j++)
                {
                    PRINT(myCharacteristic.value.paValue [j]);
                }
            }
            else
            {
                /* Handle error */
```

(continues on next page)

(continued from previous page)

```
            PRINT(error);
        }
        break;
    /* ... */
  }
}
```

Parent topic: Reading and writing characteristics

Characteristic read by UUID procedure This API for this procedure is shown here:

```
bleResult_t GattClient_ReadUsingCharacteristicUuid
(
    deviceId_t              deviceId,
    bleUuidType_t           uuidType,
    const bleUuid_t*        pUuid,
    const gattHandleRange_t* pHandleRange,
    uint8_t*                aOutBuffer,
    uint16_t                maxReadBytes,
    uint16_t*               pOutActualReadBytes
);
```

This provides support for an important optimization, which involves reading a Characteristic Value without performing any Service or Characteristic Discovery.

For example, the following is the process to write an application that connects to any Server and wants to read the device name.

The device name is contained in the Device Name Characteristic from the GAP Service. Therefore, the necessary steps involve discovering all primary services, identifying the GAP Service by its UUID, discovering all Characteristics of the GAP Service and identifying the Device Name Characteristic (alternatively, discovering Characteristic by UUID inside GAP Service), and, finally, reading the device name by using the Characteristic Read Procedure.

Instead, the Characteristic Read by UUID Procedure allows reading a Characteristic with a specified UUID, assuming one exists on the Server, without knowing the Characteristic Value Handle.

The described example is implemented as follows:

```
#define mcMaxValueLength_c 10 /* Sample value length , adjust as needed */
/* First byte is for handle-value pair length. Next 2 bytes are the handle */
static uint8_t aValue[1 + 2 + mcMaxValueLength_c];
static uint16_t deviceNameLength;
bleUuid_t uuid = {
        .uuid16 = gBleSig_GapDeviceName_d
};
bleResult_t result = GattClient_ReadUsingCharacteristicUuid
(
    deviceId,
    gBleUuidType16_c,
    &uuid,
    &pHandleRange,
    aValue,
    1 + 2 + mcMaxValueLength_c,
    &deviceNameLength
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered when the reading is complete. Because only one air packet is exchanged during this procedure, it can only be used as a quick reading of Characteristic Values with length no greater than *ATT_MTU – 1*.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
        /* ... */
         case gGattProcReadUsingCharacteristicUuid_c:
              if (gGattProcSuccess_c == procedureResult)
            {
               /* Read handle-value pair length */
                PRINT(aValue[0]);
                deviceNameLength -= 1;
              /* Read characteristic value handle */
                PRINT(aValue[1] | (aValue[2] << 8));
                deviceNameLength -= 2;
                /* Read value length */
                PRINT(deviceNameLength);
                /* Read data */
                for ( uint8_t j = 0; j < deviceNameLength; j++)
                {
                    PRINT(aValue[3 + j]);
                }
            }
             else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Reading and writing characteristics

Characteristic read multiple procedure The API for this procedure is shown here:

```
bleResult_t GattClient_ReadMultipleCharacteristicValues
(
    deviceId_t                  deviceId,
    uint8_t                     cNumCharacteristics,
    gattCharacteristic_t *      aIoCharacteristics
);
```

This procedure also allows an optimization for a specific situation, which occurs when multiple Characteristics, whose values are of known, fixed-length, can be all read in one single ATT transaction (usually one single over-the-air packet).

The application must know the value handle and value length of each Characteristic. It must also write the *value.handle*and *value.maxValueLength* with the aforementioned values, respectively, and then link the *value.paValue* field with an allocated array of size *maxValueLength*.

The following example involves reading three characteristics in a single packet.

```
#define mcNumCharacteristics_c 3
#define mcChar1Length_c 4
#define mcChar2Length_c 5
#define mcChar3Length_c 6
static uint8_t aValue1[mcChar1Length_c];
static uint8_t aValue2[mcChar2Length_c];
static uint8_t aValue3[mcChar3Length_c];
static gattCharacteristic_t myChars[mcNumCharacteristics_c];
myChars[0]. value . handle = 0x0015;
myChars[1]. value . handle = 0x0025;
myChars[2]. value . handle = 0x0035;
myChars[0]. value . maxValueLength = mcChar1Length_c;
myChars[1]. value . maxValueLength = mcChar2Length_c;
myChars[2]. value . maxValueLength = mcChar3Length_c;
myChars[0]. value . paValue = aValue1;
myChars[1]. value . paValue = aValue2;
myChars[2]. value . paValue = aValue3;
bleResult_t result = GattClient_ReadMultipleCharacteristicValues
(
    deviceId,
    mcNumCharacteristics_c,
    myChars
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

When the Client Procedure Callback is triggered, if no error occurs, each Characteristic's value length should be equal to the requested lengths.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t p  rocedureType,
    gattProcedureResult_t  procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
      case gGattProcReadMultipleCharacteristicValues_c:
          if (gGattProcSuccess_c == procedureResult)
          {
              for ( uint8_t i = 0; i < mcNumCharacteristics_c; i++)
              {
                  /* Read value length */
                  PRINT(myChars[i]. value . valueLength );
                  /* Read data */
                  for ( uint8_t j = 0; j < myChars[i]. value . valueLength ; j++)
                  {
                      PRINT(myChars[i]. value . paValue [j]);
                  }
              }
          }
          else
          {
              /* Handle error */
              PRINT(error);
          }
          break;
```

(continues on next page)

(continued from previous page)

```
        /* ... */
    }
}
```

If the server does not know the length of the characteristic values, then the Read Multiple Variable Characteristic Values procedure can be used. This sub-procedure is used to read multiple characteristic values of variable length from a server when the client knows the characteristic value handles. The response returns the characteristic values and their corresponding lengths in the Length Value Tuple List parameter.

```
bleResult_t GattClient_ReadMultipleVariableCharacteristicValues
(
  deviceId_t             deviceId,
  uint8_t                cNumCharacteristics,
  gattCharacteristic_t*  pIoCharacteristics
);
```

The following example involves reading three characteristics of variable length in a single packet.

```
#define mcNumCharacteristics_c 3
#define mcCharLengthMax_c 10
static uint8_t aValue1[mcCharLengthMax _c];
static uint8_t aValue2[mcCharLengthMax _c];
static uint8_t aValue3[mcCharLengthMax _c];
static gattCharacteristic_t myChars[mcNumCharacteristics_c];
myChars[0].value .handle = 0x0015;
myChars[1].value .handle = 0x0025;
myChars[2].value .handle = 0x0035;
myChars[0].value .paValue = aValue1;
myChars[1].value .paValue = aValue2;
myChars[2].value .paValue = aValue3;
bleResult_t result = GattClient_ReadMultipleVariableCharacteristicValues
(
    deviceId,
    mcNumCharacteristics_c,
    pIoCharacteristics
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The result of this procedure is sent to the application via the GATT procedure callback. The response includes the characteristic value together with a handle, length pair corresponding to each characteristic.

```
static void BleApp_GattClientCallback
(
    deviceId_t              serverDeviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureResult)
    {
        /* ... */
        case gGattProcReadMultipleVarLengthCharValues_c:
            if (gGattProcSuccess_c == procedureResult)
            {
```

(continues on next page)

(continued from previous page)

```
            for (uint8_t i = 0; i < mcNumCharacteristics_c; i++)
            {
                /* Print characteristic handle and length */
                PRINT(myChars[i].value.handle);
                PRINT(myChars[i].value.valueLength);
                for (uint8_t j = 0; j < myChars[i].value.maxValueLength; j++)
                {
                    /* Print characteristic value */
                    PRINT(myChars[i].value.paValue[j]);
                }
            }
        }
        else
        {
            /* Handle error */
        }
        break;
}
```

Parent topic: Reading and writing characteristics

Characteristic write procedure There is a general API that may be used for writing Characteristic Values:

```
bleResult_t GattClient_WriteCharacteristicValue
(
    deviceId_t                      deviceId,
    const gattCharacteristic_t *    pCharacteristic,
    uint16_t                        valueLength,
    const uint8_t *                 aValue,
    bool_t                          withoutResponse,
    bool_t                          signedWrite,
    bool_t                          doReliableLongCharWrites,
    const uint8_t *                 aCsrk
);
```

It has many parameters to support different combinations of Characteristic Write Procedures.

The structure pointed to by the *pCharacteristic* is only used for the *value.handle*field which indicates the Characteristic Value Handle. The value to be written is contained in the *aValue* array of size *valueLength*.

The *withoutResponse* parameter can be set to *TRUE* if the application wishes to perform a Write Without Response Procedure, which translates into an ATT Write Command. If this value is selected, the *signedWrite* parameter indicates whether data should be signed (Signed Write Procedure over ATT Signed Write Command), in which case the *aCsrk* parameters must not be NULL and contains the CSRK to sign the data with. Otherwise, both *signedWrite* and *aCsrk* are ignored.

Finally, *doReliableLongCharWrites* should be sent to *TRUE* if the application is writing a long Characteristic Value (one that requires multiple air packets due to *ATT_MTU* limitations) and wants the Server to confirm each part of the attribute that is sent over the air.

To simplify the application code, the following macros are defined:

```
#define GattClient_SimpleCharacteristicWrite(deviceId, pChar, valueLength, aValue) \
    GattClient_WriteCharacteristicValue\
        (deviceId, pChar, valueLength, aValue, FALSE, FALSE, FALSE, NULL)
```

This is the simplest usage for writing a Characteristic. It sends an ATT Write Request if the value length does not exceed the maximum space for an over-the-air packet (*ATT_MTU – 3*). Otherwise,

it sends ATT Prepare Write Requests with parts of the attribute, without checking the ATT Prepare Write Response data for consistency, and in the end an ATT Execute Write Request.

```
#define GattClient_CharacteristicWriteWithoutResponse(deviceId, pChar, valueLength, aValue) \
    GattClient_WriteCharacteristicValue\
        (deviceId, pChar, valueLength, aValue, TRUE, FALSE, FALSE, NULL)
```

This usage sends an ATT Write Command. Long Characteristic values are not allowed here and trigger a *gBleInvalidParameter_c* error.

```
#define GattClient_CharacteristicSignedWrite(deviceId, pChar, valueLength, aValue, aCsrk) \
    GattClient_WriteCharacteristicValue\
        (deviceId, pChar, valueLength, aValue, TRUE, TRUE, FALSE, aCsrk)
```

This usage sends an ATT Signed Write Command. The CSRK used to sign data must be provided.

This is a short example to write a 3-byte long Characteristic Value.

```
gattCharacteristic_t myChar;
myChar. value . handle = 0x00A0; /* Or maybe it was previously discovered? */
#define mcValueLength_c 3
uint8_t aValue[mcValueLength_c] = { 0x01, 0x02, 0x03 };
bleResult_t result = GattClient_SimpleCharacteristicWrite
(
    deviceId,
    &myChar,
    mcValueLength_c,
    aValue
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered when writing is complete.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcWriteCharacteristicValue_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Continue */
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Reading and writing characteristics

Parent topic: Client APIs

Reading and writing characteristic descriptors Two APIs are provided for these procedures which are very similar to Characteristic Read and Write.

The only difference is that the handle of the attribute to be read/written is provided through a pointer to an *gattAttribute_t* structure (same type as the *gattCharacteristic_t.value* field).

All of the following APIs have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearerId*parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer Id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

```
bleResult_t GattClient_ReadCharacteristicDescriptor
(
    deviceId_t              deviceId,
    gattAttribute_t *       pIoDescriptor,
    uint16_t                maxReadBytes
);
```

The *pIoDescriptor->handle* is required (it may have been discovered previously by *GattClient_DiscoverAllCharacteristicDescriptors*). The GATT module fills the value that was read in the fields *pIoDescriptor->aValue* (must be linked to an allocated array) and *pIoDescriptor->valueLength* (size of the array).

Writing a descriptor is also performed similarly with this function:

```
bleResult_t GattClient_WriteCharacteristicDescriptor
(
    deviceId_t              deviceId,
    gattAttribute_t *       pDescriptor,
    uint16_t                valueLength,
    uint8_t *               aValue
);
```

Only the *pDescriptor->handle* must be filled before calling the function.

One of the most frequently written descriptors is the Client Characteristic Configuration Descriptor (CCCD). It has a well-defined UUID (*gBleSig_CCCD_d*) and a 2-byte long value that can be written to enable/disable notifications and/or indications.

In the following example, a Characteristic's descriptors are discovered and its CCCD written to activate notifications.

```
static gattCharacteristic_t myChar;
myChar.value.handle = 0x00A0; /* Or maybe it was previously discovered? */
#define mcMaxDescriptors_c 5
static gattAttribute_t aDescriptors[mcMaxDescriptors_c];
myChar.aDescriptors = aDescriptors;
/* ... */
{
    bleResult_t result = GattClient_DiscoverAllCharacteristicDescriptors
    (
        deviceId,
        &myChar,
        0xFFFF,
        mcMaxDescriptors_c
    );
    if (gBleSuccess_c != result)
    {
        /* Handle error */
```

(continues on next page)

(continued from previous page)

```
    }
}
/* ... */
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
      /* ... */
      case gGattProcDiscoverAllCharacteristicDescriptors_c:
         if (gGattProcSuccess_c == procedureResult)
          {
           /* Find CCCD */
            for ( uint8_t j = 0; j < myChar.cNumDescriptors; j++)
               {
                if ((myChar.aDescriptors[j].uuidType == gBleUuidType16_c) &&
                   (gBleSig_CCCD_d == myChar.aDescriptors[j].uuid.uuid16))
                 {
                   uint8_t cccdValue[2];
                   packTwoByteValue(gCccdNotification_c, cccdValue);
                   bleResult_t result = GattClient_WriteCharacteristicDescriptor
                    (
                      deviceId,
                      &myChar. aDescriptors [j],
                      2,
                      (uint8_t*)&cccdValue
                    );
                   if (gBleSuccess_c != result)
                      {
                       /* Handle error */
                      }
                      break;
                 }
               }
            }
            else
            {
               /* Handle error */
               PRINT(error);
            }
            break;
       case gGattProcWriteCharacteristicDescriptor_c:
            if (gGattProcSuccess_c == procedureResult)
            {
               /* Notification successfully activated */
            }
            else
            {
               /* Handle error */
               PRINT(error);
            }
       /* ... */
    }
}
```

Parent topic: Client APIs

Resetting procedures To cancel an ongoing Client Procedure, the following API can be called:

```
bleResult_t GattClient_ResetProcedure (void);
```

It resets the internal state of the GATT Client and new procedure may be started at any time.

Parent topic: Client APIs

Parent topic:*Generic Attribute Profile (GATT) Layer*

Server APIs Once the GATT Database has been created and the required security settings have been registered with *Gap_RegisterDeviceSecurityRequirements*, all ATT Requests and Commands and attribute access security checks are handled internally by the GATT Server module.

Besides this automatic functionality, the application may use GATT Server APIs to send Notifications and Indication and, optionally, to intercept Clients' attempts to write certain attributes.

Server callback The first GATT Server call is the installation of the Server Callback, which has the following prototype:

```
typedef void (* gattServerCallback_t )
(
    deviceId_t              deviceId,    /*!< Device ID identifying the active connection. */
    gattServerEvent_t *     pServerEvent /*!< Server event. */
);
```

For EATT, the following signature should be used:

```
typedef void (*gattServerEnhancedCallback_t) ( deviceId_t deviceId, bearerId_t bearerId, gattServerEvent_
↪t* pServerEvent );
```

The callback can be installed with:

```
bleResult_t GattServer_RegisterCallback
(
    gattServerCallback_t callback
);
```

The EATT server callback should be installed using the following API:

```
bleResult_t GattServer_RegisterEnhancedCallback
(
     gattServerEnhancedCallback_t callback
);
```

The first member of the *gattServerEvent_t* structure is the *eventType*, an enumeration type with the following possible values:

- *gEvtMtuChanged_c:* Signals that the Client-initiated MTU Exchange Procedure has completed successfully and the *ATT_MTU* has been increased. The event data contains the new value of the *ATT_MTU*. Is it possible that the application flow depends on the value of the *ATT_MTU*, for example, there may be specific optimizations for different *ATT_MTU* ranges. This event is not triggered if the *ATT_MTU* was not changed during the procedure.
- *gEvtHandleValueConfirmation_c:* A Confirmation was received from the Client after an Indication was sent by the Server.
- *gEvtAttributeWritten_c, gEvtAttributeWrittenWithoutResponse_c:* See Attribute write notifications.
- *gEvtCharacteristicCccdWritten_c:* The Client has written a CCCD. The application should save the CCCD value for bonded devices with *Gap_SaveCccd*.

- *gEvtError_c:* An error occurred during a Server-initiated procedure.
- *gEvtLongCharacteristicWritten_c*: A long characteristic was written.
- *gEvtInvalidPduReceived_c*: An invalid PDU was received from Client. Application decides if disconnection is required.
- *gEvtAttributeRead_c*: An attribute registered with GattServer_RegisterHandlesForReadNotifications is being read.

Parent topic: Server APIs

Sending notifications and indications The APIs provided for these Server-initiated operations are very similar.

All of the following APIs have an enhanced counterpart of the form *GattServer_Enhanced[procedure]*. A *bearerId* parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearerId identifies the Unenhanced ATT bearer. Values higher than*0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

```
bleResult_t GattServer_SendNotification
(
    deviceId_t      deviceId,
    uint16_t        handle
);
bleResult_t GattServer_SendIndication
(
    deviceId_t      deviceId,
    uint16_t        handle
);
```

Only the attribute handle needs to be provided to these functions. The attribute value is automatically retrieved from the GATT Database.

Note: It is the application developer's responsibility to check if the Client designated by the *deviceId* has previously activated Notifications/Indications by writing the corresponding CCCD value. To do that, the following GAP APIs should be used:

```
bleResult_t Gap_CheckNotificationStatus
(
    deviceId_t      deviceId,
    uint16_t        handle,
    bool_t *        pOutIsActive
);
bleResult_t Gap_CheckIndicationStatus
(
    deviceId_t      deviceId,
    uint16_t        handle,
    bool_t *        pOutIsActive
    );
```

Note: It is necessary to use these two functions with the *Gap_SaveCccd* only for bonded devices, because the data is saved in NVM and reloaded at reconnection. For devices that do not bond, the application may also use its own bookkeeping mechanism.

There is an important difference between sending **Notifications and Indications**:

- The latter can only be sent one at a time. In addition, the application must wait for the Client Confirmation (signaled by the *gEvtHandleValueConfirmation_c* Server event, or by a *gEvtError_c* event with *gGattClientConfirmationTimeout_c* error code) before sending a new Indication. Otherwise, a *gEvtError_c* event with *gGattIndicationAlreadyInProgress_c* error code is triggered.

- The Notifications can be sent consecutively.

Parent topic: Server APIs

Attribute write notifications When the GATT Client reads and writes values from/into the Server's GATT Database, it uses ATT Requests.

The GATT Server module implementation manages these requests and, according to the database security settings and the Client's security status (authenticated, authorized, and so on), automatically sends the ATT Responses without notifying the application.

There are however some situations where the application needs to be informed of ATT packet exchanges. For example, a lot of standard profiles define, for certain Services, some, so-called, Control-Point Characteristics. These are Characteristics whose values are only of immediate significance to the application. Writing these Characteristics usually triggers specific actions.

For example, consider a fictitious Smart Lamp. It has Bluetooth Low Energy connectivity in the Peripheral role and it contains a small GATT Database with a Lamp Service (among other Services). The Lamp Service contains two Characteristics: the Lamp State Characteristic (LSC) and the Lamp Action Characteristic (LAC).

LSC is a "normal" Characteristic with Read and Write properties. Its value is either 0, lamp off, or 1, lamp on). Writing the value sets the lamp in the desired state. Reading it provides its current state, which is only useful when passing the information remotely.

The LAC has only one property, which is Write Without Response. The user can use the Write Without Response procedure to write only the value 0x01 (all other values are invalid). Whenever the user writes 0x01 in LAC, the lamp switches its state.

The LAC is a good example of a Control-Point Characteristic for these reasons:

- Writing a certain value (in this case 0x01) triggers an action on the lamp.
- The value the user writes has immediate significance only ("0x01 switches the lamp") and is never used again in the future. For this reason, it does not need to be stored in the database.

Obviously, whenever a Control-Point Characteristic is written, the application must be notified to trigger some application-specific action.

The GATT Server allows the application to register a set of attribute handles as "write-notifiable", in other words, the application wants to receive an event each time any of these attributes is written by the peer Client.

All Control-Point Characteristics in the GATT Database must have their Value handle registered. In fact, the application may register any other handle for write notifications for its own purposes with the following API:

```
bleResult_t GattServer_RegisterHandlesForWriteNotifications
(
    uint8_t         handleCount,
    const uint16_t *    aAttributeHandles
);
```

The *handleCount* is the size of the *aAttributeHandles* array and it cannot exceed *gcGattMaxHandleCountForWriteNotifications_c*.

After an attribute handle has been registered with this function, whenever the Client attempts to write its value, the GATT Server Callback is triggered with one of the following event types:

- *gEvtAttributeWritten_c* is triggered when the attribute is written with a Write procedure (ATT Write Request). In this instance, the application has to decide whether the written value is valid and whether it must be written in the database, and, if so, the application must write the value with the *GattDb_WriteAttribute*, see *GATT database application interface*. At this point, the GATT Server module does not automatically send the ATT Write Response over the air. Instead, it waits for the application to call this function:

```
bleResult_t GattServer_SendAttributeWrittenStatus
(
    deviceId_t      deviceId,
    uint16_t        attributeHandle,
    uint8_t         status
);
```

This API also has an enhanced counterpart, which adds the *bearerId* parameter.

The value of the *status* parameter is interpreted as an ATT Error Code. It must be equal to the *gAttErrCodeNoError_c* (0x00) if the value is valid and it is successfully processed by the application. Otherwise, it must be equal to a profile-specific error code (in interval 0xE0-0xFF) or an application-specific error code (in interval 0x80-0x9F).

- *gEvtAttributeWrittenWithoutResponse_c* is triggered when the attribute is written with a Write Without Response procedure (ATT Write Command). Because this procedure expects no response, the application may process it and, if necessary, write it in the database. Regardless of whether the value is valid or not, no response is needed from the application.
- *gEvtLongCharacteristicWritten_c* is triggered when the Client has completed writing a Long Characteristic value; the event data includes the handle of the Characteristic Value attribute and a pointer to its value in the database.

Attributes can also be registered for read notifications using the followng API:

```
bleResult_t GattServer_RegisterHandlesForReadNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);
```

To unregister one or more handles from the list for either write or read, the following APIs can be used:

```
bleResult_t GattServer_UnregisterHandlesForWriteNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);

bleResult_t GattServer_UnregisterHandlesForReadNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);
```

Parent topic: Server APIs

Parent topic:*Generic Attribute Profile (GATT) Layer*

GATT database application interface For over-the-air packet exchanges between a Client and a Server, the GATT Server module automatically retrieves data from the GATT database and responds to all ATT Requests from the peer Client, provided it passes the security checks. This ensures that the Server application does not have to perform any kind of searches over the database.

However, the application must have access to the database to write meaningful data into its characteristics. For example, a temperature sensor must periodically write the temperature, which is measured by an external thermometer, into the Temperature Characteristic.

For these kinds of situations, a few APIs are provided in the *gatt_db_app_interface.h* file.

Note: All functions provided by this interface are executed synchronously. The result of the operation is saved in the return value and it generates no event.

Writing and reading attributes These are the two functions to perform basic attribute operations from the application:

```
bleResult_t GattDb_WriteAttribute
(
    uint16_t           handle,
    uint16_t           valueLength,
    const uint8_t *    aValue
);
```

The value length must be valid, as defined when the database is created. Otherwise, a *gGattInvalidValueLength_c* error is returned.

Also, if the database is created statically, as explained in *Creating GATT database*, the *handle* may be referenced through the enumeration member with a friendly name defined in the *gatt_db.h*.

```
bleResult_t GattDb_ReadAttribute
(
    uint16_t      handle,
    uint16_t      maxBytes,
    uint8_t *     aOutValue,
    uint16_t *    pOutValueLength
);
```

The *aOutValue* array must be allocated with the size equal to *maxBytes*.

Parent topic:*GATT database application interface*

Finding attribute handles Although the application should be fully aware of the contents of the GATT Database, in certain situations it might be useful to perform some dynamic searches of certain attribute handles.

To find the handle value for a Service for which only the UUID is know the following API can be used:

```
bleResult_t GattDb_FindServiceHandle
(
 uint16_t startHandle,
 bleUuidType_t serviceUuidType,
 const bleUuid_t* pServiceUuid,
 uint16_t* pOutServiceHandle
);
```

To find a specific Characteristic Value Handle in a Service whose declaration handle is known, the following API is provided:

```
bleResult_t GattDb_FindCharValueHandleInService
(
    uint16_t               serviceHandle,
    bleUuidType_t          characteristicUuidType,
    const bleUuid_t *      pCharacteristicUuid,
    uint16_t *             pOutCharValueHandle
);
```

If the return value is *gBleSuccess_c*, the handle is written at *pOutCharValueHandle*. If the *serviceHandle* is invalid or not a valid Service declaration, the *gBleGattDbInvalidHandle_c* is returned. Otherwise, the search is performed starting with the *serviceHandle+1*. If no Characteristic of the given UUID is found, the function returns the *gBleGattDbCharacteristicNotFound_c* value.

To find a Characteristic Descriptor of a given type in a Characteristic, when the Characteristic Value Handle is known, the following API is provided:

```
bleResult_t GattDb_FindDescriptorHandleForCharValueHandle
(
    uint16_t            charValueHandle,
    bleUuidType_t       descriptorUuidType,
    const bleUuid_t *   pDescriptorUuid,
    uint16_t *          pOutDescriptorHandle
);
```

Similarly, the function returns *gBleGattDbInvalidHandle_c* is the handle is invalid. Otherwise, it starts searching from the *charValueHandle+1*. Then, *gBleGattDbDescriptorNotFound_c* is returned if no Descriptor of the specified type is found. Otherwise, its attribute handle is written at the *pOutDescriptorHandle* and the function returns *gBleSuccess_c*.

One of the most commonly used Characteristic Descriptors is the Client Configuration Characteristic Descriptor (CCCD), which has the UUID equal to *gBleSig_CCCD_d*. For this specific type, a special API is used as a shortcut:

```
bleResult_t GattDb_FindCccdHandleForCharValueHandle
(
    uint16_t     charValueHandle,
    uint16_t *   pOutCccdHandle
);
```

Parent topic:*GATT database application interface*

Creating GATT database The GATT Database contains several *GATT Services* where each Service must contain at least one *GATT Characteristic*.

The Attribute Database contains a collection of *attributes*. Each attribute has four fields:

- The *attribute handle* – a 2-byte database index, which starts from 0x0001 and increases with each new attribute, not necessarily consecutive; maximum value is 0xFFFF.
- The *attribute type* or *UUID* – a 2-byte or 16-byte UUID.
- The *attribute permissions* – 1 byte containing access flags; this defines whether the attribute's value can be read or written and the security requirements for each operation type
- The *attribute value* – an array of maximum 512 bytes.

The ATT does not interpret the UUIDs and values contained in the database. It only deals with data transfer based on the attributes' handles.

The GATT gives meaning to the attributes based on their UUIDs and groups them into Characteristics and Services.

There are two possible ways of defining the GATT database:

- At compile-time (statically) or
- At runtime (dynamically)

Creating static GATT database To define a GATT Database at compile-time, several macros are provided by the GATT_DB API. These macros expand in many different ways at compilation, generating the corresponding *Attribute Database* on which the Attribute Protocol (ATT) may operate.

This is the default way of defining the database.

The GATT Database definition is written in two files that are required to be added to the application project together with all macro expansion files:

- ***gatt_db.h*** - contains the actual declaration of Services and Characteristics.
- ***gat_uuid128.h*** – contains the declaration of Custom UUIDs (16-byte wide); these UUIDs are given a user-friendly name that is used in *gatt_db.h* file instead of the entire 16-byte sequence.

Declaring custom 128-bit UUIDs All Custom 128-bit UUIDs are declared in the required file *gatt_uuid128.h*.

Each line in this file contains a single UUID declaration. The declaration uses the following macro:

- *UUID128 (name, byte1, byte2, ..., byte16)*

The*name* parameter is the user-friendly handle that references this UUID in the *gatt_db.h*file.

The 16 bytes are written in the *LSB-first* order each one using the 0xZZ format.

Note: On some occasions, it is desired to reuse an 128-bit UUID declared in *gatt_uuid128.h*. The 16 byte array is available through its friendly name and be accessed by including *gatt_db_handles.h* in the application. It is strongly advised to use it only in read-only operations. For example:

```
(gatt_uuid128.h)
UID128(uuid_service_wireless_uart, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E,␣
↪0xBA, 0x00, 0x01, 0xFF, 0x01)
(app.c)
#include "gatt_db_handles.h"
.....
/* Start Service Discovery*/
BleServDisc_FindService(peerDeviceId, gBleUuidType128_c, (bleUuid_t*) &uuid_service_wireless_uart);
```

Parent topic: Creating static GATT database

Declaring a service There are two types of Services:

- *Primary Services*
- *Secondary Services* - these are only to be included by other Primary or Secondary Services

The Service declaration attribute has one of these UUIDs, as defined by the Bluetooth SIG:

- 0x2800 a.k.a. *«Primary Service»* - for a Primary Service declaration
- 0x2801 a.k.a. *«Secondary Service»* - for a Secondary Service declaration

The Service declaration attribute permissions are read-only and no authentication required. The Service declaration attribute value contains the *Service UUID*. The *Service Range* starts from the Service declaration and ends at the next service declaration. All the Characteristics declared within the Service Range are considered to belong to that Service. For a more comprehensive list of SIG defied UUID values, check ble_sig_defines.h.

Service declaration macros The following macros are to be used for declaring a Service:

- *PRIMARY_SERVICE (name, uuid16)*
  - Most often used.
  - The *name* parameter is common to all macros; it is a universal, user-friendly identifier for the generated attribute.
  - The *uuid16* is a 2-byte SIG-defined UUID, written in 0xZZZZ format.

- *PRIMARY_SERVICE_UUID32 (name, uuid32)*
  - This macro is used for a 4-byte, SIG-defined UUID, written in 0xZZZZZZZZ format.
- *PRIMARY_SERVICE_UUID128 (name, uuid128)*
  - The *uuid128* is the friendly name given to the custom UUID in the *gatt_uuid128.h* file.
- *SECONDARY _SERVICE (name, uuid16)*
- *SECONDARY_SERVICE_UUID32 (name, uuid32)*
- *SECONDARY _SERVICE_UUID128 (name, uuid128)*
  - All three are similar to Primary Service declarations.

Parent topic: Declaring a service

Include declaration macros Secondary Services are meant to be included by other Services, usually by Primary Services. Primary Services may also be included by other Primary Services. The inclusion is done using the Include declaration macro:

- *INCLUDE (service_name)*
  - The *service_name* parameter is the friendly name used to declare the Secondary Service.
  - This macro is used only for Secondary Services with a SIG-defined, 2-byte, Service UUID.
- *INCLUDE_CUSTOM (service_name)*
  - This macro is used for Secondary Services that have either a 4-byte UUID or a 16-byte UUID.

The effect of the service inclusion is that the *including* Service is considered to contain all the Characteristics of the *included* Service.

Parent topic: Declaring a service

Parent topic: Creating static GATT database

Declaring a characteristic A Characteristic must only be declared inside a Service. It belongs to the most recently declared Service, so the GATT Database must always begin with a Service declaration.

The Characteristic declaration attribute has the following UUID, as defined by the Bluetooth SIG:

- 0x2803 a.k.a. *«Characteristic»*

The Characteristic declaration attribute value contains:

- the *Characteristic UUID*
- the *Characteristic Value* 's declaration handle
- the *Characteristic Properties* – Read, Write, Notify, and so on. (1 byte of flags)

The *Characteristic Range*starts from the Characteristic declaration and ends before a new Characteristic or a Service declaration.

After the Characteristic declaration these follow:

- A *Characteristic Value* declaration (mandatory; immediately after the Characteristic declaration).
- Zero or more *Characteristic Descriptor* declarations.

Characteristic declaration macros The following macros are used to declare Characteristics:

- *CHARACTERISTIC (name, uuid16, properties)*
- *CHARACTERISTIC_UUID32 (name, uuid32, properties)*
- *CHARACTERISTIC _UUID128 (name, uuid128, properties)*

See Service declaration for *uuidXXX* parameter explanation.

The *properties* parameter is a bit mask. The flags are defined in the *gattCharacteristicPropertiesBitFields_t*.

Parent topic: Declaring a characteristic

Declaring characteristic values The Characteristic Value declaration immediately follows the Characteristic declaration and uses one of the following macros:

- *VALUE (name, uuid16, permissions, valueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID32 (name, uuid32, permissions, valueLength, valueByte1, valueByte2, ...)*
- *VALUE _UUID128(name, uuid128, permissions, valueLength, valueByte1, valueByte2, ...)*
  - See Declaring a service for description of the *uuidXXX* parameter.
  - The *permissions* parameter is a bit mask, whose flags are defined in *gattAttributePermissionsBitFields_t* .
  - The *valueLength* is the number of bytes to be allocated for the Characteristic Value. After this parameter, exactly [*valueLength*] bytes follow in 0xZZ format, representing the initial value of this Characteristic.

These macros are used to declare Characteristic Values of *fixed lengths*.

Some Characteristics have *variable length values*. For those, the following macros are used:

- *VALUE_VARLEN (name, uuid16, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID32_VARLEN (name, uuid32, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID128_VARLEN (name, uuid128, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
  - The number of bytes allocated for this Characteristic Value is *maximumValueLength*.
  - The number of *valueByteXXX* parameters shall be equal to *initialValueLength*.

Obviously, *initialValueLength* is, at most, equal to *maximumValueLength*.

Parent topic: Declaring a characteristic

Declaring characteristic descriptors Characteristic's Descriptors are declared after the Characteristic Value declaration and before the next Characteristic declaration.

The macros used to declare Characteristic Descriptors are very similar to those used to declare fixed-length Characteristic Values:

- *DESCRIPTOR (name, uuid16, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*
- *DESCRIPTOR_UUID32 (name, uuid32, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*
- *DESCRIPTOR_UUID128(name, uuid128, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*

A special Characteristic Descriptor that is used very often is the *Client Characteristic Configuration Descriptor (CCCD)*. This is the descriptor where clients write some of the bits to activate Server notifications and/or indications. It has a reserved, 2-byte, SIG-defined UUID (0x2902), and its attribute value consists of only 1 byte (out of which 2 bits are used for configuration, the other 6 are reserved).

Because the CCCD appears very often in Characteristic definitions for standard Bluetooth Low Energy profiles, a special macro is used for CCCD declaration:

- *CCCD (name)*

This simple macro is basically equivalent to the following Descriptor declaration:

```
* DESCRIPTOR \(name, *
   *0x2902, *
   (gGattAttPermAccessReadable_c
   | gGattAttPermAccessWritable_c),
   2, 0x00, 0x00)
```

Parent topic: Declaring a characteristic

Parent topic: Creating static GATT database

Static GATT database definition examples The GAP Service must be present on any GATT Database. It has the Service UUID equal to 0x1800, *«GAP Service»*, and it contains three read-only Characteristics, no authentication required: *Device Name, Appearance,*and *Peripheral Preferred Connection Parameters*. These also have well defined UUIDs in the SIG documents.

Most of the demos also include the optional GATT Security Levels characteristic, which defines the highest security requirements of the GATT server when operating in a LE connection.

The definition for this Service is shown here:

```
PRIMARY_SERVICE(service_gap, 0x1800)
   CHARACTERISTIC(char_device_name, 0x2A00, (gGattCharPropRead_c) )
       VALUE(value_device_name, 0x2A00, (gGattAttPermAccessReadable_c), 6, “Sensor”)
   CHARACTERISTIC(char_appearance, 0x2A01, (gGattCharPropRead_c) )
      VALUE(value_appearance, 0x2A01, (gGattAttPermAccessReadable_c), 2,
       0xC2, 0x03)
   CHARACTERISTIC(char_ppcp, 0x2A04, (gGattCharPropRead_c) )
       VALUE(value_ppcp, 0x2A04, (gGattAttPermAccessReadable_c), 8, 0x0A,
       0x00, 0x10, 0x00, 0x64, 0x00, 0xE2, 0x04)
   CHARACTERISTIC(char_security_levels, gBleSig_GattSecurityLevels_d,
      (gGattCharPropRead_c) )
       VALUE(value_security_levels, gBleSig_GattSecurityLevels_d,
       (gPermissionFlagReadable_c), 2, 0x01, 0x01)
```

Another often encountered Service is the Scan Parameters Service:

```
PRIMARY_SERVICE(service_scan_parameters, 0x1813)
     CHARACTERISTIC(char_scan_interval_window, 0x2A4F,          (gGattCharPropWriteWithoutRsp_
↪c) )
       VALUE(value_scan_interval_window, 0x2A4 (gGattAttPermAccessWritable), 4, 0x00, 0x00, 0x00,␣
↪0x00)
      CHARACTERISTIC(char_scan_refresh, 0x2A31, (gGattCharPropRead_c gGattCharPropNotify_c) )
       VALUE(value_scan_refresh, 0x2A31, (gGattAttPermAccessReadable_c), 1, 0x00) CCCD(cccd_scan_
↪refresh)
```

Note: All “user-friendly” names given in declarations are statically defined as *enum* members, numerically equal to the *attribute handle* of the declaration. This means that one of those names can be used in code wherever an attribute handle is required as a parameter of a function if *gatt_db_handles.h* is included in the application source file. For example, to write the value of the Scan Refresh Characteristic from the application-level code, use these instructions:

```
#include "gatt_db_handles.h"
...
uint8_t scan_refresh_value = 0x12;
GattDb_WriteAttribute(char_scan_refresh, 1, &scan_refresh_value);
```

For static database declarations, the 'attribute handle' is equal to the line number in the gatt_fb.h file, where the attribute is defined.

Parent topic: Creating static GATT database

Parent topic:*Creating GATT database*

Creating a GATT database dynamically To define a GATT Database at runtime, the *gGattDbDynamic_d* macro must be defined in *app_preinclude.h* with the value equal to 1.

Then, the application must use the APIs provided by the *gatt_db_dynamic.h* interface to add and remove Services and Characteristics as needed.

See Creating static GATT database for a detailed description of Service and Characteristic parameters.

Memory considerations The GATT Dynamic database module internally manages the memory allocation for the database. There are two ways to specify the memory configuration. The selection is done by setting gGattDynamicAttrSize_c and/or gGattDynamicValSize_c.

- *Statically pre-allocated memory area*
  - gGattDynamicAttrSize_c - the total size of the table of characteristics
  - gGattDynamicValSize_c - the total size of the values cumulated from all characteristics. If any of these macros is greater than 1 the application is responsible to decide upon the best value. In case the selected value is not big enough gBleOutOfMemory_c is returned by the GATT API calls.
- *Dynamic allocated memory area* If the gMemManagerLightExtendHeapAreaUsage define is set to 1 in the desired application, the whole available heap is used. In such as case, the user does not have to allocate space for the dynamic database. If this is not done, the user only needs to make sure that the MinimalHeapSize_c define is set to a high enough value considering all attributes and attribute values they want to add to the database, as well as other memory requirements the application might have. This method is used in case only several attributes are added dynamically. If multiple attributes are needed the method from *Statically pre-allocated memory area* above is recommended. This is because the heap gets fragmented with each adding of a new attribute.

Internally, two buffers are used by the dynamic database module: an attribute buffer and a value buffer. The attribute buffer size increases with the addition of each attribute to the database. The value buffer size increases depending on the UUID type and value lengths required by the application. The two buffers start with a minimum size and are reallocated whenever new requests to add entries are received and there is not enough available memory left. If the user removes these entries from the database, the memory reserved for those entries is not freed, but shifted, leaving room for new entries. Thus, an add operation after a remove operation might not necessarily reallocate the buffer if the new entries fit. The two buffers used by the Dynamic database module will not be available to the application until the user releases the database.

Parent topic: Creating a GATT database dynamically

Initialization and release Before anything can be added to the database, it must be initialized with an empty collection of attributes.

The *GattDbDynamic_Init()* API is automatically called by the *GattDb_Init()* implementation provided in the *gatt_database.c* source file. Application-specific code does not need to call this API again, unless at some point it destroys the database with *GattDb_ReleaseDatabase()*.

Parent topic: Creating a GATT database dynamically

Adding services The APIs that can be used to add Services are self-explanatory:

- *GattDbDynamic_AddPrimaryServiceDeclaration*
  - The Service UUID is specified as parameter.

    Memory requirements: one entry in the attribute buffer and UUID size in value buffer.
- *GattDbDynamic_AddSecondaryServiceDeclaration*
  - The Service UUID is specified as parameter.

    Memory requirements: one entry in the attribute buffer and UUID size in value buffer.
- *GattDbDynamic_AddIncludeDeclaration*
  - The Service UUID and handle range are specified as parameters.

    Memory requirements: one entry in the attribute buffer and 6 bytes in value buffer.

The functions have an optional out parameter *pOutHandle*. If its value is not NULL, the execution writes a 16-bit value in the pointed location representing the attribute handle of the added declaration. The application can use this handle as parameter in few *GattDbApp* APIs or in the Service removal functions.

At least one Service must be added before any Characteristic.

Parent topic: Creating a GATT database dynamically

Adding characteristics and descriptors The APIs for adding Characteristics and Descriptors are enumerated below:

- *GattDbDynamic_AddCharacteristicDeclarationAndValue*
  - The Characteristic UUID, properties, access permissions, and initial value are specified as parameters.
- *GattDbDynamic_AddCharacteristicDeclarationWithUniqueValue*
  - Multiple calls to this API allocate a unique 512-byte value buffer as an optimization for application that deal with large value buffers that do not always need to be stored separately.
- *GattDbDynamic_AddCharacteristicDescriptor*
  - The Descriptor UUID, access permissions and initial value are specified as parameters.
- *GattDbDynamic_AddCccd*
  - Shortcut for a CCCD.

Characteristics and descriptors are automatically added at the end of the database. Thus, a service declaration should be followed by all desired characteristic and descriptor definitions before adding a new service to the database.

Parent topic: Creating a GATT database dynamically

Removing services and characteristics To remove a Service or a Characteristic, the following APIs may be used, both of which only require the declaration handle as parameter:

- *GattDbDynamic_RemoveService*
- *GattDbDynamic_RemoveCharacteristic*

Parent topic: Creating a GATT database dynamically

Parent topic:*Creating GATT database*

Gatt caching

Service change feature The **service changed** feature applies to GATT servers and supports the service changed characteristic, dynamic databases, and handle value indications. The GATT clients that require to be notified for structural modifications on the database should write the CCCD of the Service Changed Characteristic on the server. The value of the Service Changed characteristic is represented by 2 handle values for the handle range affected by the modifications.

The changes that trigger a server database modification are represented by the following API calls:

- *GattDbDynamic_AddPrimaryServiceDeclaration*
- *GattDbDynamic_AddSecondaryServiceDeclaration*
- *GattDbDynamic_AddIncludeDeclaration*
- *GattDbDynamic_AddCharacteristicDeclarationAndValue*
- *GattDbDynamic_AddCharDescriptor*
- *GattDbDynamic_AddCccd*
- *GattDbDynamic_RemoveService*
- *GattDbDynamic_RemoveCharacteristic*

Those GATT server API calls update two internal handles to memorize the minimum and maximum range affected by the change.

After the GATT server database update is done, the application must call the GattDbDynamic_EndDatabaseUpdate() API. After this, a Service Changed indication is internally sent to each connected peer that has enabled these indications. The indication contains the handle range affected by the change.

. For bonded devices with whom the server is not currently in an active connection, the changes are buffered on the server and the peers are notified upon reconnection.

Parent topic: Gatt caching

Robust caching Robust caching is a feature where the server sends an error response to the client if the server does not consider the client to be aware of a database structural change. A server supporting robust caching provides the Client Supported Features, Database Hash, and Service Changed characteristics. To indicate support for robust caching, clients should write the robust caching bit (bit 0) of the Client Supported Features characteristic on the server. An example of the GAP service definition for a server with robust caching support is the following:

- *PRIMARY_SERVICE(service_gatt, gBleSig_GenericAttributeProfile_d)*
- *CHARACTERISTIC(char_service_changed, gBleSig_GattServiceChanged_d, (gGattCharPropIndicate_c))*
- *VALUE(value_service_changed, gBleSig_GattServiceChanged_d, (gPermissionNone_c), 4, 0x01, 0x00, 0xFF, 0xFF)*

- *CCCD(cccd_service_changed)*
- *CHARACTERISTIC(char_client_supported_features, gBleSig_GattClientSupportedFeatures_d, (gGattCharPropRead_c | gGattCharPropWrite_c))*
- *VALUE(value_client_supported_features, gBleSig_GattClientSupportedFeatures_d, (gPermissionFlagReadable_c | gPermissionFlagWritable_c), 8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)*
- *CHARACTERISTIC(char_database_hash, gBleSig_GattDatabaseHash_d, (gGattCharPropRead_c))*
- *VALUE(value_database_hash, gBleSig_GattDatabaseHash_d, (gPermissionFlagReadable_c), 16, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)*

The Client Supported Features characteristic should be declared as an array. The array size is equal to the maximum number of connections, which can be active at one time, so that each possible peer can have its own value. The Server Supported Features characteristic value is automatically written by the Host to indicate robust caching support when appropriate. The Database Hash characteristic value is a 128-bit unsigned integer number where the computed hash value is written.

In order to enable the Robust Caching feature, the *gGattCaching_d* define should be enabled in the file app_preinclude.h. In order to enable automatic Host support for Robust Caching, the *gGattAutomaticRobustCachingSupport_d*define should also be enabled. This includes writing the Client Supported Features characteristic, writing the CCCD of the Service Changed characteristic, and reading the Database Hash after reconnecting with a previously bonded peer to check for new changes. If this define is not enabled, then it is up to the application to read and write all necessary characteristic for Robust Caching support.

The client state is kept on the server using the following enum:

```
typedef enum
{
gGattClientChangeUnaware_c = 0x00U, /*!< Gatt client state */
gGattClientStateChangePending_c = 0x01U, /*!< Gatt client state */
gGattClientChangeAware_c = 0x02U, /*!< Gatt client state */
} gattCachingClientState_c;
```

The initial state of a client without a trusted relationship is change-aware. The state of a client with a trusted relationship remains unchanged from the previous connection. However, in cases where the database has been updated since the last connection, the initial state is change-unaware. When a database update occurs, all connected clients become change unaware.

If a change-unaware client sends an ATT command, the server ignores it. For ATT requests received from a change-unaware client, the server sends an error response with the error code set to *gAttErrCodeDatabaseOutOfSync_c*. The server should also not send indications and notifications to change unaware clients, except for the Service Changed indication. The state of a client is verified by the GATT server before executing each command, request or sending any notifications or indications.

The following PDU types are an exception to this rule and do not generate an gAttErrCodeDatabaseOutOfSync_c error code:

- ATT_FIND_INFORMATION_REQ
- ATT_FIND_BY_TYPE_VALUE_REQ
- ATT_READ_BY_GROUP_TYPE_REQ
- ATT_EXECUTE_WRTIE

For a change unaware client to become change aware again, one of the following must happen:

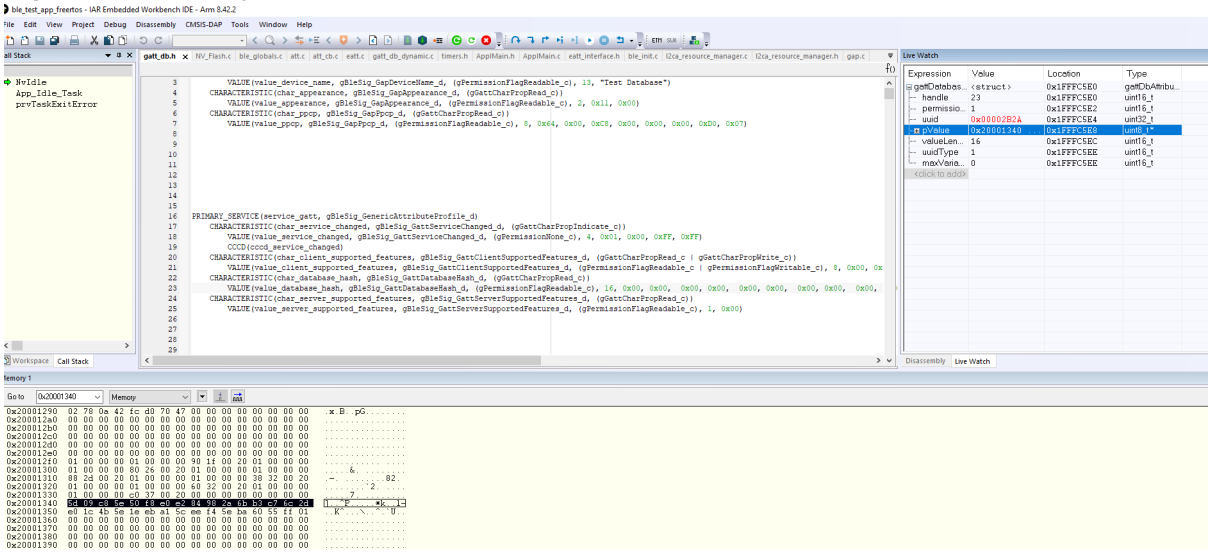
- The client receives and confirms a Service Changed Indication.

- The server, upon receiving a request from a change unaware client, sends the client a response with the error code set to Database Out Of Sync and then the server receives another ATT request from the client.
- The change unaware client reads the Database Hash characteristic and then the server receives another ATT request from the client.

The function *GattDb_ComputeDatabaseHash()* is used by the server to compute the hash value and save its value in the database. The computation is done when a read request for the database hash characteristic is first received from a peer GATT client for dynamic databases.

For static databases, hash computation is disabled by default. If you have a static database and want to compute the database hash, then declare the following define to TRUE in app_preinclude.h: *gGattDbComputeHash_d*. By doing this, the hash value is computed during the host initialization. The value is written directly to the database as characteristic and it can be viewed in the memory, as see in the image below. Since static databases do not change in structure over time, this value remains constant, so it can be saved separately and written manually to memory if needed. See Figure 10 below.

Figure 10. Memory view of the Database Hash characteristic



On the client side, the *GattClient_GetDatabaseHash()* function is used to read the hash value from a peer GATT server. If the *gGattAutomaticRobustCachingSupport_c* define is enabled, then the following steps are executed automatically:

- Writing the Client Supported Features characteristic value to indicate robust caching support – set BIT0 to 1.
- Write the CCCD of the Service Changed characteristic.
- Read the initial value of the Database Hash characteristic and store it locally.

Otherwise, just the read request for the Database Hash characteristic is sent to the peer.

The following arrays and variables are used for the implementations of the robust caching and service changed features (declared in ble_globals.c):

```
/* Service changed indication buffer */
   gattHandleRange_t gServiceChangedIndicationStorage[gMaxBondedDevices_c];

/* client saved values for service changed characteristic and CCCD handles */
   uint16_t mActiveServiceChangedCharHandle[gAppMaxConnections_c] = {gGattDbInvalidHandle_d};
   uint16_t mServiceChangedCharHandle[gMaxBondedDevices_c] = {gGattDbInvalidHandle_d};
   uint16_t mActiveServiceChangedCCCDHandle[gAppMaxConnections_c] = {gGattDbInvalidHandle_d};

/* server values for its own service changed characteristic and CCCD handles */
```

(continues on next page)

(continued from previous page)

```
    uint16_t mServerServiceChangedCharHandle;
    uint16_t mServerServiceChangedCCCDHandle;

/* client state information for bonded and active clients */
    gattCachingClientState_c gGattClientState[gMaxBondedDevices_c] = {gGattClientChangeAware_c};
    gattCachingClientState_c gGattActiveClientState[gAppMaxConnections_c] = {gGattClientChangeAware_
↪c};

/* Database hash values - the client needs a hash value for each possible peer */
    uint8_t mGattActiveServerDatabaseHash[gGattDatabaseHashSize_c * gAppMaxConnections_c] = {0};
    uint8_t mGattServerDatabaseHash[gGattDatabaseHashSize_c * gMaxBondedDevices_c] = {0};

/* client supported features handles for active gatt servers */
    uint16_t gGattActiveClientSupportedFeaturesHandles[gAppMaxConnections_c] =
↪{gGattDbInvalidHandle_d};

/* client supported features information for bonded gatt clients */
    uint8_t gGattClientSupportedFeatures[gMaxBondedDevices_c] = {0U};

/* index of the database hash characteristic in the database */
    uint32_t mServerDatabaseHashIndex = gGattDbInvalidHandleIndex_d;

/* index of the client supported features characteristic in the database */
    uint32_t mServerClientSupportedFeatureIndex = gGattDbInvalidHandleIndex_d;
```

It is up to the application to save a local copy of the information from the server's database and to initiate service discovery only on the first connection or when it is informed of a change by the peer using Service Changed and Robust Caching.

If the *gGattAutomaticRobustCachingSupport_c* define is not set, it is up to the application to check the Server Supported Features characteristic value on the peer, to write the Client Supported Features characteristic value and to write the Service Changed CCCD.

Two new GATT procedures are introduced as part of the Robust Caching feature. Both should be treated according to the application needs in the GATT procedure callback of the application.

- *gGattProcSignalServiceDiscoveryComplete_c –*informs the application that the service discovery procedure has finished after reading the Database Hash value using the read using characteristic UUID procedure. The application procedure callback should call *BleServDisc_Finished()*on this event when robust caching is supported.
- *gGattProcUpdateDatabaseCopy_c –*informs the application that its database copy is no longer up to date and service discovery should be reperformed. Used when the client received an error response with the *gAttErrCodeDatabaseOutOfSync_c* opcode, when the local database hash value is found to be out of sync with the one on the server, or when a service changed indication is received from the server.

If service discovery is performed using our ble_service_discovery module, then the application should wait for the *gDiscoveryFinished_c* event before initiating its own GATT procedures. The application should also make sure to not initiate a second GATT procedure which requires a response from the peer before receiving a response to the first request it made.

Parent topic: Gatt caching

Parent topic:*Creating GATT database*

Creating a Custom Profile This chapter describes how the user can create customizable functionality over the Bluetooth Low Energy Host Stack by defining profiles and services. The Temperature Profile, used by the Temperature Sensor and Collector applications, is used as a reference to explain the steps of building custom functionality.

Defining custom UUIDs The first step when defining a new service included in a profile is to define the custom 128-bit UUID for the service and the included characteristics. These values are defined in *gatt_uuid128.h*, which is located in the application folder. For example, the Temperature Profile uses the following UUID for the service:

```
/* Temperature */
UUID128(uuid_service_temperature, 0xfb ,0x34 ,0x9b ,0x5f ,0x80 ,0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x02,␣
↪0x00, 0xfe, 0x00, 0x00)
```

The definition of the services and characteristics are made in *gattdb.h*, as explained in *Creating GATT database*. For more details on how to structure the database, see *Application Structure*.

Parent topic:*Creating a Custom Profile*

Creating service functionality All defined services in the SDK have a common template which helps the application to act accordingly.

The service locally stores the device identification for the connected client. This value is changed on subscription and non-subscription events.

```
/*! Temperature Service - Subscribed Client*/
static deviceId_t mTms_SubscribedClientId;
```

The service is initialized and changed by the application through a service configuration structure. It usually contains the service handle, initialization values for the service (for example, the initial temperature for the Temperature Service) and in some cases user-specific structures that can store saved measurements (for example, the Blood Pressure Service). Below is an example for the custom Temperature Service:

```
/*! Temperature Service - Configuration */
typedef struct tmsConfig_tag
{
    uint16_t serviceHandle ;
    int16_t initialTemperature ;
} tmsConfig_t ;
```

The initialization of the service is made by calling the start procedure. The function requires as input a pointer to the service configuration structure. This function is usually called when the application is initialized. It resets the static device identification for the subscribed client and initializes both dynamic and static characteristic values. An example for the Temperature Service (TMS) is shown below:

```
bleResult_t Tms_Start ( tmsConfig_t *pServiceConfig)
{
    mTms_SubscribedClientId = gInvalidDeviceId_c;
    return Tms_RecordTemperatureMeasurement (pServiceConfig-> serviceHandle ,
                                             pServiceConfig-> initialTemperature );
}
```

The service subscription is triggered when a device connects to the server. It requires the peer device identification as an input parameter to update the local variable. On disconnect, the unsubscribe function is called to reset the device identification. For the Temperature Service:

```
bleResult_t Tms_Subscribe ( deviceId_t deviceId)
{
    mTms_SubscribedClientId = deviceId;
    return gBleSuccess_c;
}
bleResult_t Tms_Unsubscribe (void)
{
    mTms_SubscribedClientId = gInvalidDeviceId_c;
```

(continues on next page)

(continued from previous page)

```
    return gBleSuccess_c;
}
```

Depending on the complexity of the service, the API implements additional functions. For the Temperature Service, there is only a temperature characteristic that is notifiable by the server. The API implements the record measurement function which saves the new measured value in the GATT database and send the notification to the client device if possible. The function needs the service handle and the new temperature value as input parameters:

```
bleResult_t Tms_RecordTemperatureMeasurement ( uint16_t serviceHandle, int16_t temperature)
{
    uint16_t handle;
    bleResult_t result;
    bleUuid_t uuid = Uuid16(gBleSig_Temperature_d);
    /* Get handle of Temperature characteristic */
    result = GattDb_FindCharValueHandleInService(serviceHandle,
        gBleUuidType16_c, &uuid, &handle);
    if (result != gBleSuccess_c)
        return result;
    /* Update characteristic value */
    result = GattDb_WriteAttribute(handle, sizeof( uint16_t ), ( uint8_t *)&temperature);
    if (result != gBleSuccess_c)
        return result;
    Hts_SendTemperatureMeasurementNotification(handle);
    return gBleSuccess_c;
}
```

To accommodate some use cases where the service is reset, the stop function is called. The reset also implies a service unsubscribe. Below is an example for the Temperature Service:

```
bleResult_t Tms_Stop ( tmsConfig_t *pServiceConfig)
{
    return Tms_Unsubscribe();
}
```

Parent topic:*Creating a Custom Profile*

GATT client interactions The client side of the service, which includes the service discovery, notification configuration, attribute reads and others are left to be handled by the application. The application calls the GATT client APIs and reacts accordingly. The only exception for this rule is that the service interface declares the client configuration structure. This structure usually contains the service handle and the handles of all the characteristic values and descriptors discovered. Additionally it can contain values that the client can use to interact with the server. For the Temperature Service client, the structure is as follows:

```
/*! Temperature Client - Configuration */
typedef struct tmcConfig_tag
{
    uint16_t                hService;
    uint16_t                hTemperature ;
    uint16_t                hTempCccd ;
    uint16_t                hTempDesc ;
    gattDbCharPresFormat_t   tempFormat ;
} tmcConfig_t ;
```

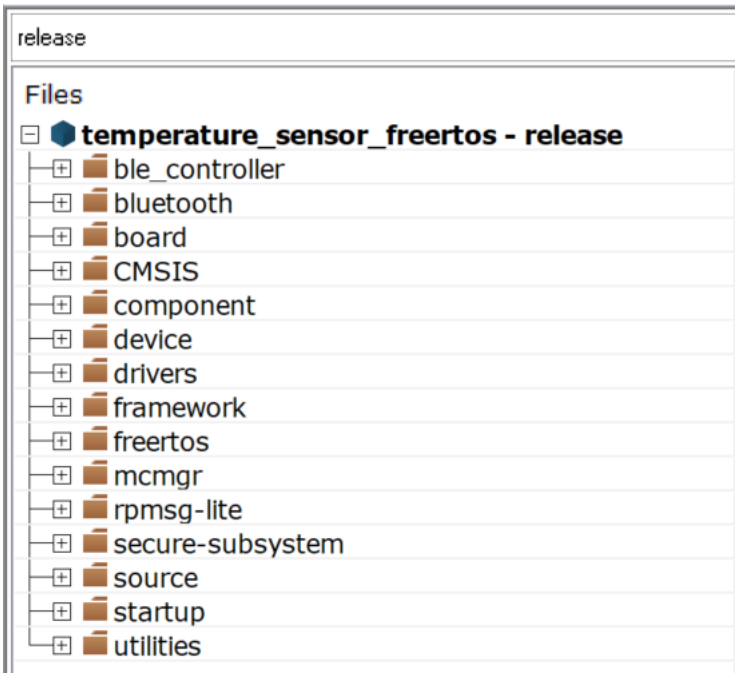
Parent topic:*Creating a Custom Profile*

Application Structure This chapter describes the organization of the Bluetooth Low Energy demo applications that can be found in the SDK. By familiarizing with the application structure,

the user is able to quickly adapt its design to an existing demo or create a new application.

The Temperature Sensor application is used as a reference to showcase the architecture.

Folder structure The Figure shows the application folder structure.

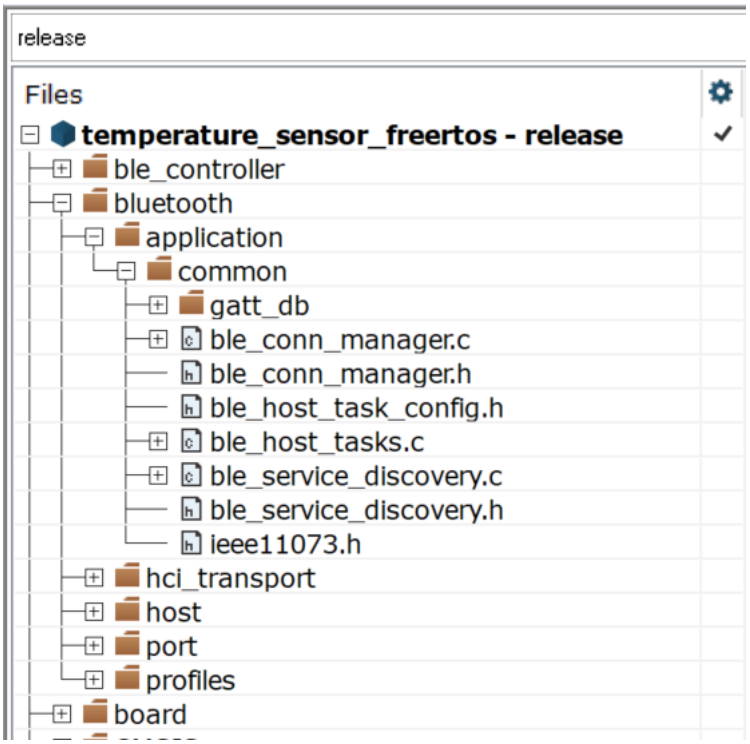


Application Folder structure in workspace

The *app* folder follows a specific structure which is recommended for any application developed using the Bluetooth Low Energy Host Stack:

- The *common* group contains the application framework shared by all profiles and demo applications:
  - Bluetooth Low Energy Connection Manager
  - Bluetooth Service Discovery Manager
  - Bluetooth Low Energy Stack and Task Initialization and Configuration
  - GATT Database

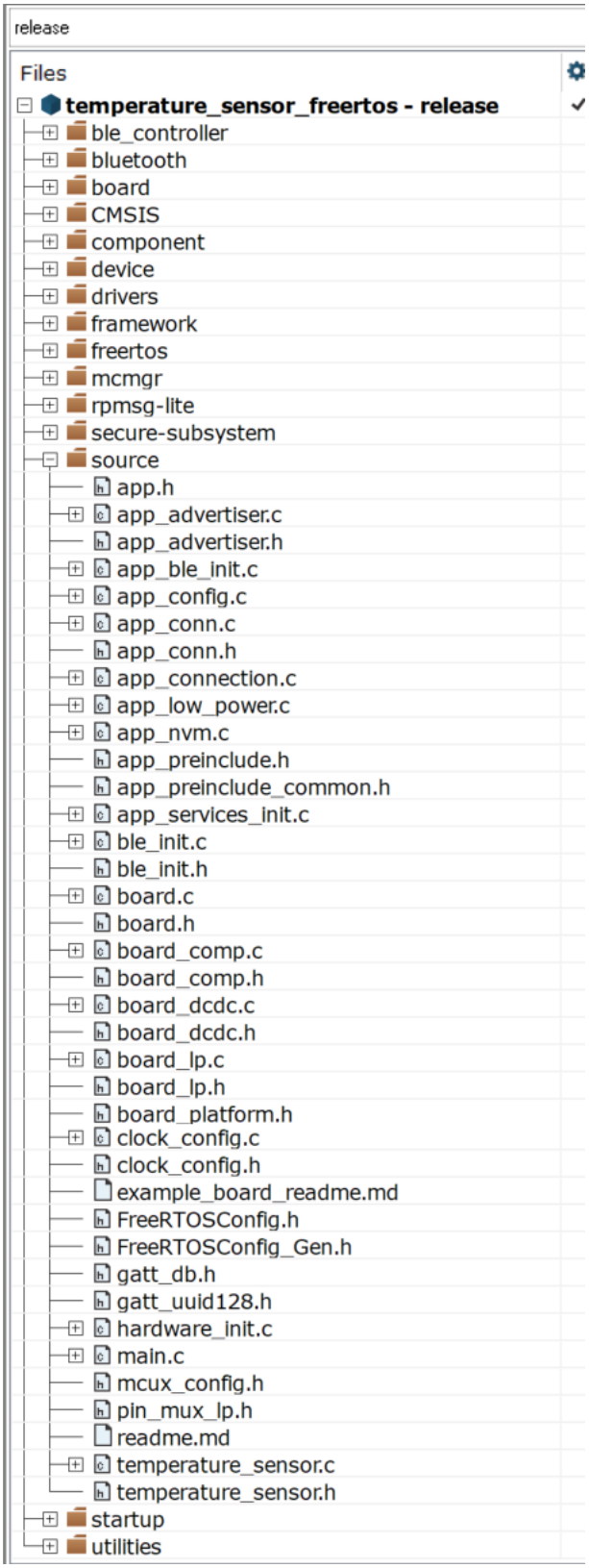
Common Group



- The *source* group contains code specific to the Temperature Sensor application.

  In the following examples, *app.c* is used as a placeholder for the main application source file. In the case of Temperature Sensor, it is *temperature_sensor.c*.

  The source group also contains files which aid with the handling of Host events and framework related functionality. These are: *app_conn.c*/*app_conn.h*, *app_advertiser.c*, *app_connection.c*, *app_scanner.c*, *app_nvm.c*, and *app_low_power.c*. These files do not allow the application to implement its own state machines, unless application-specific functionality is required. For example, the application is restricted from setting advertising parameters and starting advertising, unless certain application-specific functionalities, such as UI related updates are required.



Source Group

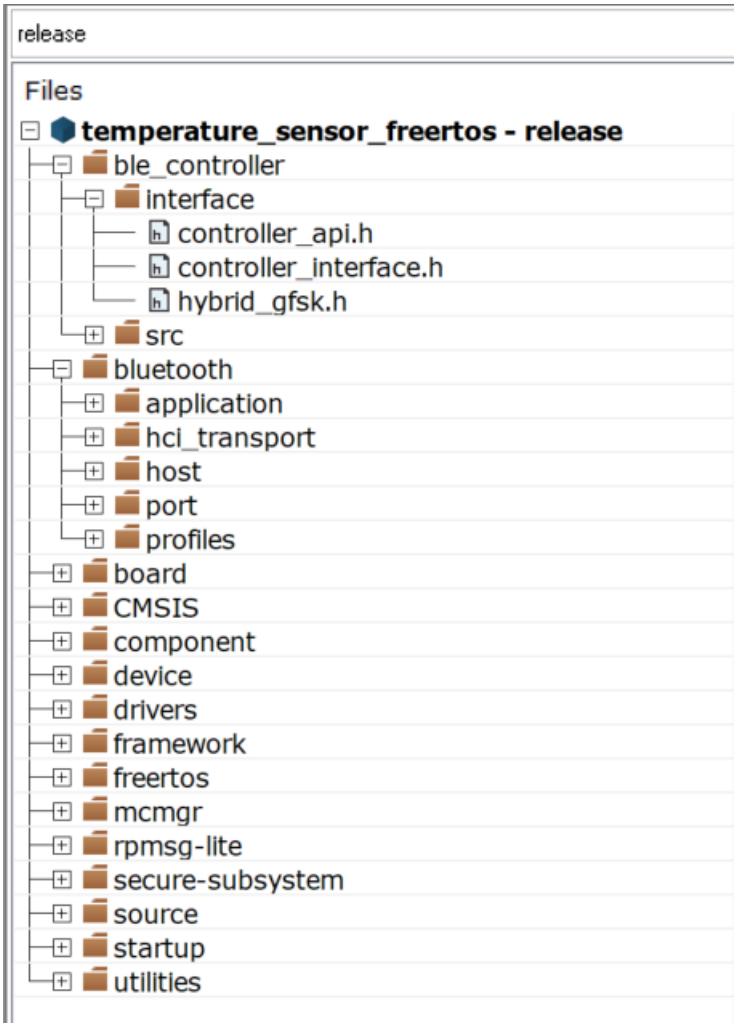
The *bluetooth* folder/group contains:

- The *host/interface*, and *host/config*. These are public interfaces and configuration files for the Host. The functionality is included in the library located in the *host/lib* subfolder. The folder is not shown in the IAR project structure, but added into the toolchain linker settings under the library category.
- *bluetooth/profiles* contains profile-specific code; it is used by each demo application of stan-

dard profiles.

The *ble_controller* folder/group contains:

- The *controller/interface*. These are public interfaces and configuration files for the Controller.



Bluetooth and Controller Groups

The framework and component folders/groups contain framework components used by the demo applications. For additional information, see the *Connectivity Framework Reference Manual*.

The *freertos* folder contains sources for the supported operating system.

Parent topic:*Application Structure*

Application main framework The Application Main module contains common code used by all the applications, such as:

- The Main Task
- Messaging framework between the Bluetooth LE Host Stack Task and the Application Task

Start task The Start Task (*start_task*) is the first task created by the operating system and is also the one that initializes the rest of the system. It initializes framework components (Memory Manager, Timers Manager etc.) and it calls *BluetoothLEHost_AppInit* from *app.c*, which is used to initialize the Bluetooth LE Host Stack as well as peripheral drivers specific to the implemented application.

The function calls *BluetoothLEHost_HandleMessages*, which represents the Application Task and is used to process events and messages from the Host Stack.

The stack size and priority of the main task are defined in *fsl_os_abstraction_config.h*:

```
#ifndef gMainThreadStackSize_c
#define gMainThreadStackSize_c 1024
#endif
#ifndef gMainThreadPriority_c
#define gMainThreadPriority_c 7
#endif
```

Parent topic: Application main framework

Application messaging The module contains a wrapper that is used to create messages for events generated by the Bluetooth LE Host Stack in the Host Task context. The wrapper also sends them to be processed by the application in the context of the Application Task.

For example, connection events generated by the Host are received by *App_ConnectionCallback*. The function creates a message, places it in the Host to Application queue and signals the Application with *gAppEvtMsgFromHostStack_c*. The Application Task de-queues the message and calls *App_HandleHostMessageInput*, which calls the corresponding callback implemented the application-specific code (*app.c*), in this example: *BleApp_ConnectionCallback*.

It is strongly recommended that the application developer uses the *app.c* module to add custom code on this type of callbacks.

Parent topic: Application main framework

Parent topic:*Application Structure*

Bluetooth LE Connection Manager The connection manager is a helper module that contains common application configurations and interactions with the Bluetooth LE Host Stack. It implements the following events and methods:

- Host Stack GAP Generic Event
- Host Stack Connection Event on both GAP Peripheral and GAP Central configuration
- Host Stack configuration for GAP Peripheral or GAP Central

GAP generic event The GAP Generic Event is triggered by the Bluetooth LE Host Stack and sent to the application via the generic callback. Before any application-specific interactions, the Connection Manager callback is called to handle common application events, such as device address storage.

```
void BleApp_GenericCallback ( gapGenericEvent_t * pGenericEvent)
{
  /* Call Bluetooth Low Energy Conn Manager */
  BleConnManager_GenericEvent(pGenericEvent);
  switch (pGenericEvent-> eventType )
  {
    ...
  }
}
```

In the **BleConnManager_GenericEvent** function, local keys are generated.

- The local LTK, IRK, and CSRK as well as the EDIV and RAND are obtained hashing over the board's UID and stored in RAM as plain-text every time the **gInitializationComplete_c** event is received.

- In **Advanced Secure** mode, local IRK and CSRK are generated using the EdgeLock Secure Enclave and stored into a dedicated NVM data set as ELKE blobs (40 bytes blob encrypted using unique die key) on the first gInitializationComplete_c event received.

The NBU Decryption key for IRK is generated and distributed to the NBU over the private key bus. The EIRK blob (16 bytes blob which can be decrypted only by NBU hardware using NBU Decryption key for IRK) is generated from the IRK ELKE blob and stored in the RAM to be used for controller privacy on every gInitializationComplete_c event received. For the host privacy, the ELKE IRK blob is used instead. For details, refer to the section “Advanced security capabilities”.

Parent topic: Bluetooth LE Connection Manager

GAP configuration The GAP Central or Peripheral Configuration is used to create common configurations (such as setting the public address, registering the security requirements, adding the addresses of bonded devices in the Controller Filter Accept List), which can be customized by the application afterwards. It is called inside the *BluetoothLEHost_Initialized callback* function, before any application-specific configuration, as shown in the example code below.

```
static void BluetoothLEHost_Initialized()
{
    /* Set common GAP configuration */
    BleConnManager_GapCommonConfig();
    ...
}
```

Parent topic: Bluetooth LE Connection Manager

GAP connection event The GAP Connection Event is triggered by the Host Stack and sent to the application via the connection callback. Before any application-specific interactions, the Connection Manager callback is called to handle common application events, such as device connect, disconnect or pairing-related requests. It is called inside the registered connection such as shown below:

```
static voidBle App_ConnectionCallback ( deviceId_t peerDeviceId, gapConnectionEvent_t *␣
↪pConnectionEvent)
{
    /* Connection Manager to handle Host Stack interactions */
    BleConnManager_GapPeripheralEvent(peerDeviceId, pConnectionEvent);
    switch (pConnectionEvent-> eventType )
    {
        ...
    }
}
```

It is strongly recommended that the application developer uses the app.c module to add custom code.

Parent topic: Bluetooth LE Connection Manager

Privacy To enable or disable Privacy, the following APIs may be used:

```
bleResult_t
 BleConnManager_EnablePrivacy(void);
```

```
bleResult_t
 BleConnManager_DisablePrivacy(void);
```

The function BleConnManager_EnablePrivacy calls *BleConnManager_ManagePrivacyInternal* after checking if the privacy is enabled.

```
static bleResult_t
BleConnManager_ManagePrivacyInternal
(bool_t bCheckNewBond);
```

If the privacy feature is supported (*gAppUsePrivacy_d = 1*), the Connection Manager activates Controller Privacy or Host Privacy depending on the board capabilities.

The *bCheckNewBond* is a boolean that tells the Manager whether it should check or not if a bond between the devices already exists.

In order to update the identity information after a bond is added or removed privacy should be disabled and enabled. For pairing with bonding this is done automatically in *ble_conn_manager*. In case the application adds or removes a bond through the GAP API, it should also disable and enable privacy.

Parent topic: Bluetooth LE Connection Manager

Parent topic:*Application Structure*

GATT database The *gatt_db* contains a set of header files grouped in the *macros* subfolder. These macros are used for static code generation for the GATT Database by expanding the contents of the *gatt_db.h* file in different ways. *Creating GATT database* explains how to write the *gatt_db.h* file using user-friendly macros that define the GATT database.

At application compile time, the *gatt_database.c* file is populated with enumerations, structures, and initialization code used to allocate and properly populate the GATT database. In this way, the *gattDatabase*array and the *gGattDbAttributeCount_c* variable (see GATT database) are created and properly initialized.

Note: Do not modify any of the files contained in the *gatt_db* folder and its subfolder.

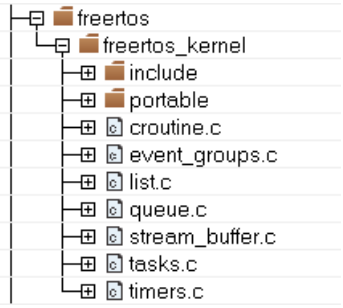
To complete the GATT database initialization, this demo application includes the required `gatt_db.h` and `gatt_uuid128.h` files in its specific application folder, along with other profile-specific configuration and code files.

Parent topic:*Application Structure*

RTOS specifics

Operating system selection The SDK offers different projects for each supported operating system (FreeRTOS OS) and for BareMetal configuration. To switch between systems, the user needs to switch the workspace.

The RTOS source code is found in the SDK package and is linked in the workspace in the *freertos* virtual folder, as shown in Figure:



Location of FreeRTOS source code in workspace

Parent topic:RTOS specifics

Bluetooth LE Host task configuration Application developers are provided with two files for RTOS task initialization:

- *ble_host_task_config.h*, and *ble_host_tasks.c* for the Host.

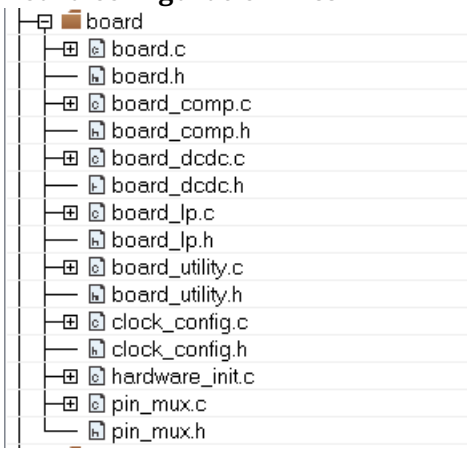
Reusing these files is recommended because they perform all the necessary RTOS-related work. The application developer must only modify the macros from **_config.h* files whenever tasks need a bigger stack size or different priority settings. The new values should be overridden in the *app_preinclude.h* file.

Parent topic:RTOS specifics

Parent topic:*Application Structure*

Board configuration The configuration files for the supported boards can be found in the *board* folder, as shown in Figure. The files contain clock and pin configurations that are used by the drivers. The user can customize the board files by modifying the configuration of the pins and clock source according to his design.

Board configuration files



Parent topic:*Application Structure*

Bluetooth Low Energy initialization The *ble_init.h* and *ble_init.c* files contain the declaration and the implementation of the following function:

```
bleResult_t Ble_Initialize
(
    gapGenericCallback_t gapGenericCallback
)
{
#if defined(gUseHciTransportDownward_d) && gUseHciTransportDownward_d
    /* HCI Transport Init */
    if (gHciSuccess_c != Hcit_Init(Ble_HciRecvFromIsr))
    {
        return gHciTransportError_c;
    }

    /*
     * Set BD Address in Controller. Must be done after HCI init
     * and before Host init.
     */
      Ble_SetBDAddr();

    /* Check for available memory storage */
    if (!Ble_CheckMemoryStorage())
    {
```

(continues on next page)

(continued from previous page)

```
        return gBleOutOfMemory_c;
    }
    /* Bluetooth Low Energy Host Tasks Init */
    if (KOSA_StatusSuccess != Ble_HostTaskInit())
    {
        return gBleOsError_c;
    }
    /* Bluetooth Low Energy Host Stack Init */
    return Ble_HostInitialize(gapGenericCallback, Hcit_SendPacket);
#elif defined(gUseHciTransportUpward_d) && gUseHciTransportUpward_d
#else /* gUseHciTransportUpward_d */
#endif /* gUseHciTransportUpward_d */
}
```

Note: This function should be used by your application because it correctly performs all the necessary Bluetooth Low Energy initialization.

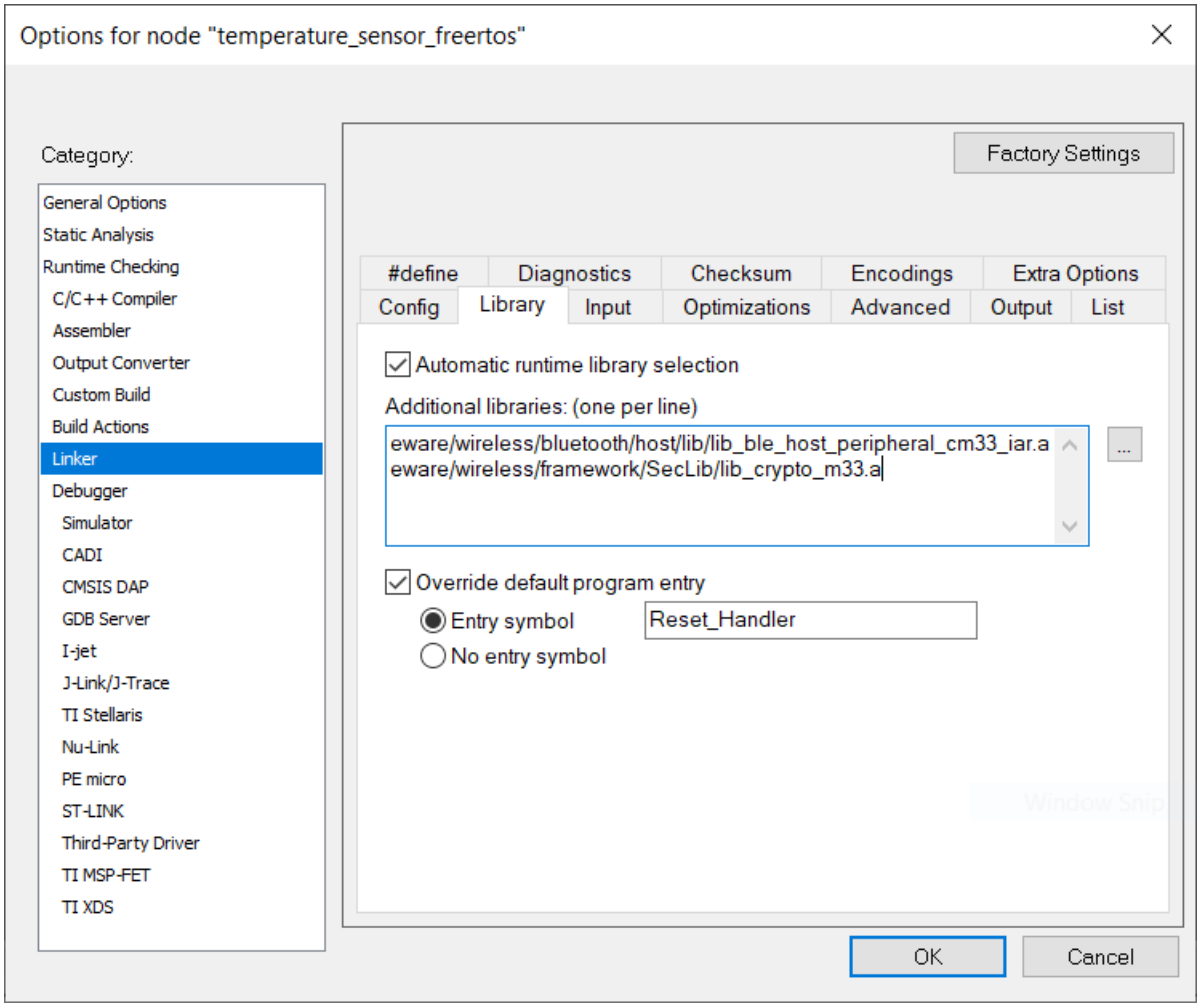
Step-by-step analysis is provided below:

1. First, the HCI interface is initialized by calling *Hcit_Init*. This initializes communication between the Host and the Controller.
2. After setting the BD address into the Controller (Ble_SetBDAddr) and performing memory validation checks (*Ble_CheckMemoryStorage*), the Host task is initialized by calling *Ble_HostTaskInit*.
3. Finally the *Ble_HostInitialize* function initializes the Host with the transport packet transmit function used as the *hciHostToControllerInterface_t parameter*.

Parent topic:*Application Structure*

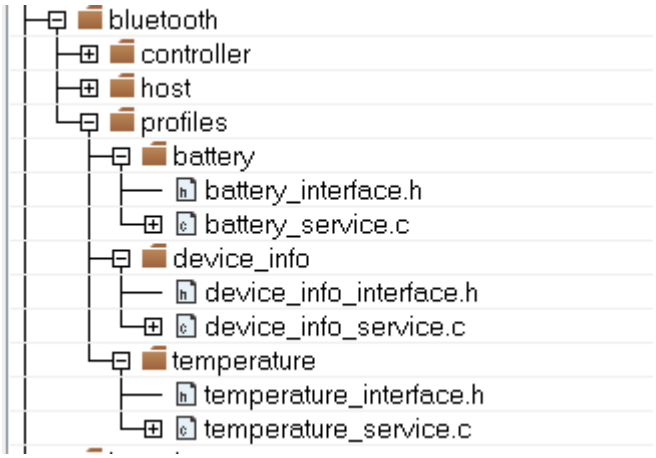
Bluetooth Low Energy Host Stack configuration The Bluetooth LE Host Stack libraries are found in the middleware/wireless/bluetooth/host/lib folder. The user should add the best matching library for its use case to the linker options of its project.

For example, the temperature sensor uses the Peripheral Host Stack library, as shown in Figure below:



Parent topic:*Application Structure*

Profile configuration The implemented profiles and services are located in *middleware/wireless/bluetooth/profiles* folder. The application links every service source file and interface it needs to implement the profile. For example, for the Temperature Sensor the tree looks as shown Figure:



The Temperature Profile implements the custom Temperature service, the Battery, and Device Information services.

Application code The application folder contains the following modules:

- *app.c* and *app.h*. This module stores the application-specific functionality (APIs for specific triggers, handling of peripherals, callbacks from the stack, handling of low power, and so on).

Before initializing the Bluetooth LE Host Stack, the start task calls *BluetoothLEHost_AppInit*. This function initializes application specific functionality before initializing the Bluetooth LE Host Stack by calling *BluetoothLEHost_Init*.

After the stack is initialized, the *BluetoothLEHost_Initialized* callback is called. The function contains configurations made to the Bluetooth LE Host Stack after the initialization. This includes registering callbacks, setting security for services, starting services, allocating timers, adding devices to the Filter Accept List, and so on. For example, the Temperature Sensor configures the following:

```
static void BluetoothLEHost_Initialized(void)
{
  /* Common GAP configuration */
    BleConnManager_GapCommonConfig();

  /* Register for callbacks*/
    (void)App_RegisterGattServerCallback(BleApp_GattServerCallback);

    mAdvState.advOn = FALSE;

  /* Start services */
    SENSORS_TriggerTemperatureMeasurement();
    (void)SENSORS_RefreshTemperatureValue();
    /* Multiply temperature value by 10. SENSORS_GetTemperature() reports temperature
    value in tenths of degrees Celsius. Temperature characteristic value is degrees
    Celsius with a resolution of 0.01 degrees Celsius (GATT Specification
    Supplement v6). */
    tmsServiceConfig.initialTemperature = (int16_t)(10 * SENSORS_GetTemperature());
    (void)Tms_Start(&tmsServiceConfig);

    basServiceConfig.batteryLevel = SENSORS_GetBatteryLevel();
    (void)Bas_Start(&basServiceConfig);
    (void)Dis_Start(&disServiceConfig);

  /* Allocate application timer */
    (void)TM_Open(appTimerId);

    AppPrintString("\r\nTemperature sensor -> Press switch to start advertising.\r\n");
}
```

To start the application functionality, BleApp_Start() function is called. This function usually contains code to start advertising for sensor nodes or scanning for central devices. In the example of the Temperature Sensor, the function is the following:

```
static void BleApp_Start(void)
{
    Led1On();

    if (mPeerDeviceId == gInvalidDeviceId_c)
    {
      /* Device is not connected and not advertising */
        if (!mAdvState.advOn)
        {
          /* Set advertising parameters, advertising to start on gAdvertisingParametersSetupComplete_c */
          BleApp_Advertise();
        }
    }
```

(continues on next page)

(continued from previous page)

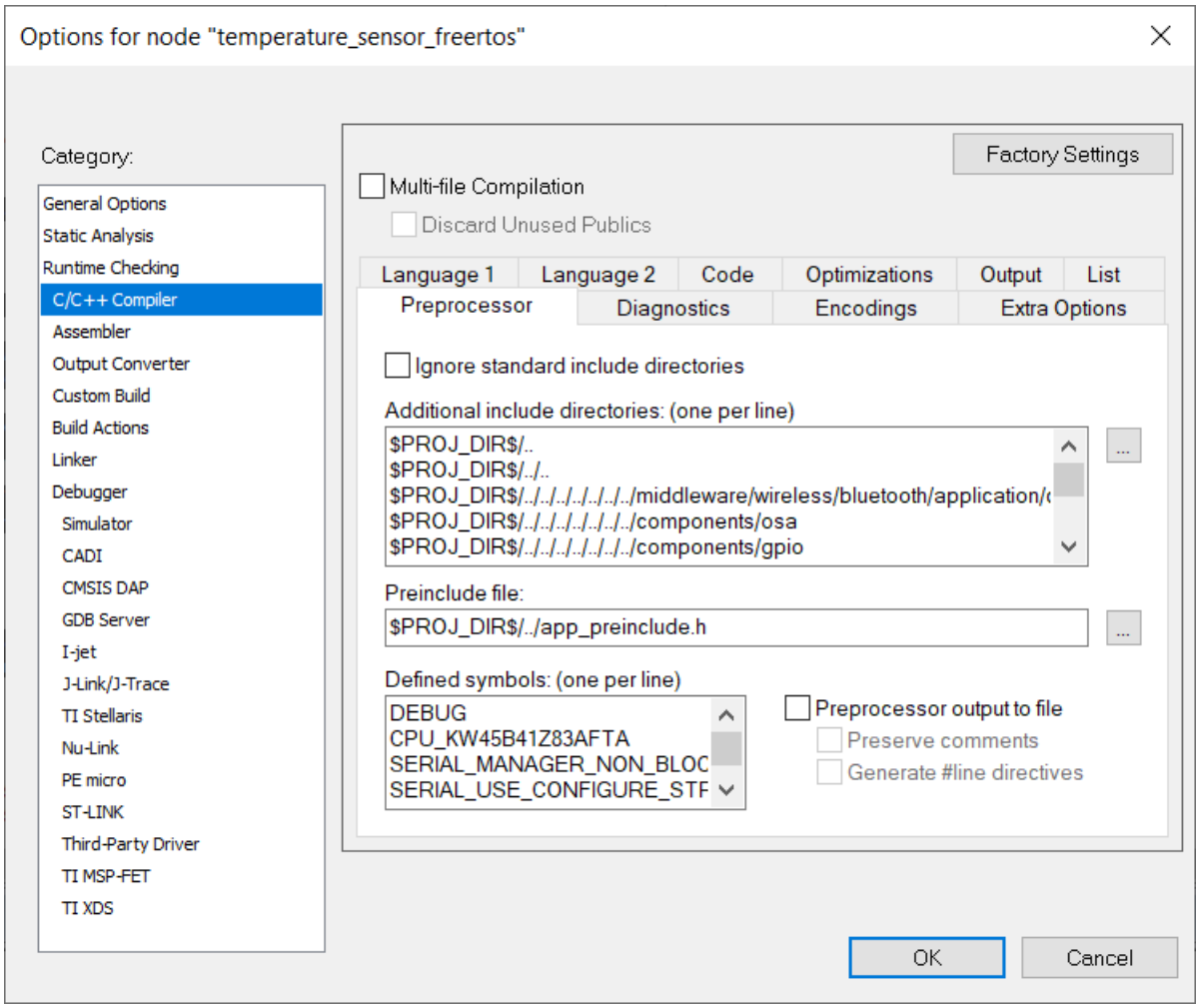
```
    else
    {
        /* Device is connected, send temperature value */
        BleApp_SendTemperature();
    }
}
```

- *app_config.c.* This file contains data structures that are used to configure the stack.

This includes advertising data, scanning data, connection parameters, advertising parameters, SMP keys, security requirements, and so on.

- *app_preinclude.h.*

This header file contains macros to override the default configuration of any module in the application. It is added as a preinclude file in the preprocessor command line in IAR, as shown in Figure:



- *gatt_db.h* and *gatt_uuid128.h.* The two header files contain the definition of the GATT database and the custom UUIDs used by the application. See *Creating GATT database* for more information.

Parent topic: Profile configuration

Parent topic:*Application Structure*

Multiple connections Applications can be configured to support multiple connections. To allow multiple connections, the *gAppMaxConnections_c* must be set to a value up to the maximum number of connections (*this value is chip-specific*). Refer to the chip documentation for the supported number of connections.

The application can save information about the peer devices it connects to according to the value of *gAppMaxConnections_c*. The Bluetooth Low Energy profile associated with the application use case must be instantiated to support the use of its functionality for each peer device. When handling multiple connections, the applications can behave as either the GAP central, GAP peripheral, or both at the same time. It is up to the application code to decide whether to start the advertising or scanning before creating the next connection. The supported combinations enable a device to connect as a peripheral to multiple centrals, as a central to multiple peripherals, or for it to be a central for some peers and a peripheral to others. The demo applications provide this functionality as an example of exercising multiple connection support. In such applications, the GAP role can be changed from central to peripheral and the information is saved for each peer device.

Parent topic:*Application Structure*

Bluetooth address generation BD_ADDR is represented by 48 bits that uniquely identify a device and consist of a 24-bit OUI (Organizationally Unique Identifier) and a 24-bit random part that varies between devices. There are multiple options of storing and using the BD_ADDR. Depending on the chip, it may be read from a device specific register (if supported), from the global hardware parameters stored in the flash, or generated randomly based on the processor-unique identifier. The demo applications provide a combination of the last two options. The *Ble_SetBDAddr* function is called during the initialization process, after initializing HCI and before initializing the Bluetooth LE Host Stack. The global hardware parameters are read from the flash. If a useful value is found, it is used as the BD address. If the found value is all 0xFFs, an address is generated by concatenating the OUI configured at compile time with three randomly generated bytes. The result is stored in the hardware parameters for future use and then set into the Controller. The Bluetooth LE Host Stack uses little-endian format to represent all addresses, in compliance with the Bluetooth Core Specification.

Parent topic:*Application Structure*

Repeated attempts Applications can be configured to enable protection against repeated Pairing Requests/Peripheral Security Requests coming from the same device. This is to prevent an intruder from repeating the pairing process with a large number of different keys in order to extract information about the local device's private key. If this feature is enabled, after a pairing procedure fails, another attempt to pair coming from the same device is allowed only after a specific time period has passed. For each failure, the waiting period doubles up until a maximum period.

The following *app_preinclude.h* macros support this feature:

- gRepeatedAttempts_d
  - Set to 1 to enable the feature. By default, it is disabled (0).
- gRepeatedAttemptsNoOfDevices_c
  - Number of remote devices to keep track of. By default, the value is 4.
  - If a new device needs to be added and the list is full, one of the oldest entries will be replaced.
- gRepeatedAttemptsTimeoutMin_c
  - Minimum waiting period in seconds – default 10.
- gRepeatedAttemptsTimeoutMax_c

– Maximum waiting period in seconds – default 640. The waiting period doubles after each failed pairing with the same device.

Parent topic:*Application Structure*

Advanced Secure Mode This section describes the advanced security capabilities of the Bluetooth LE Host Stack which are available on the KW45/K32W1/KW47/MCXW72 platform via the EdgeLock Secure Enclave (ELKE).

The security capabilities are enabled at application, Host and Controller level by setting Advanced Secure Mode to active. To do this, the user must set the *gAppSecureMode_d* macro to *1* in the project's *app_preinclude.h* file. This macro is defined by default as *0* in *app_preinclude_common.h*:

```
#if (gAppSecureMode_d == 1U)
#define gHostSecureMode_d               (1U)
#else
#define gHostSecureMode_d               (0U)
#endif
```

At application level, when Advanced Secure Mode is enabled, the security mode and level for pairing is automatically enforced as Mode 1 Level 4, ensuring LE Secure Connections pairing. Legacy pairing is not supported in this mode.

When enabled, the main benefit of Advanced Secure Mode is the secured storage and handling of Bluetooth LE security keys. The EdgeLock Secure Enclave is capable of generating, importing, and exporting security keys as plain text or as encrypted blobs. All encrypted blobs are created by the EdgeLock Secure Enclave using a die unique key, which makes them impossible to decrypt by devices other than the one that created them. The Bluetooth LE security keys are managed in Advanced Secure Mode as follows:

- IRK
  - The peer IRKs received after pairing and bonding are no longer stored into NVM as plaintext, but as encrypted blobs 40 bytes in length (or ELKE blobs). They can still be retrieved and converted to plaintext.
  - The local IRK is no longer generated using the default method of hashing over the board's UID at every startup. It is instead generated once using the EdgeLock Secure Enclave and stored into a new NVM dataset as an ELKE blob.
  - Local and peer IRKs are no longer transmitted through HCI in plaintext but as EIRK (Encrypted IRK) blobs, 16 bytes in length, which can be decrypted by the Controller.
- LTK
  - The LTK is no longer stored into NVM as plaintext, but as an ELKE blob. Furthermore, the plaintext of the LTK is never available to the Host/application. Generating the LTK via the ECDH process and generating the Session Key for individual connections is done by the Host via the EdgeLock Secure Enclave and custom vendor HCI messages which are transparent to the application.
- CSRK
  - The local CSRK is no longer generated using the default method of hashing over the board's UID at every startup. It is instead generated once using the EdgeLock Secure Enclave and stored into a new NVM dataset as an ELKE blob.

At startup, Advanced Secure Mode for the Controller is enabled dynamically by calling:

```
#if (defined(gAppSecureMode_d) && (gAppSecureMode_d > 0U))
    (void) PLATFORM_EnableBleSecureKeyManagement();
#endif
```

This call can be found in BluetoothLEHost_Init, as part of the initialization sequence.

Additional considerations

- The Gap_LoadKeys API will return the IRK in plaintext, having converted it from the ELKE blob which is stored in the NVM. However, it cannot convert the LTK to plaintext - it will thus return the LTK in ELKE blob form.
- As mentioned in the section above, the plaintext LTK is never available. Generating the Session Key for individual connections is done via the EdgeLock Secure Enclave by the Host, using the LTK in ELKE blob form. The Session Key is passed to the Controller via a custom vendor HCI message. Be aware that the content of the HCI_LE_Long_Term_Key_Request_Reply command is not the actual LTK and cannot be used for debugging.
- The Host works with IRKs in ELKE blob form, therefore APIs such as Gap_EnableHostPrivacy and Gap_CreateRandomAddress take ELKE blobs as arguments. The Controller works with IRKs in EIRK blob form, therefore the Gap_EnableControllerPrivacy takes an EIRK blob as argument. This management is done in *ble_conn_manager.c* when Advanced Secure Mode is enabled.

Parent topic:*Application Structure*

Low-Power Management

System considerations The KW45/K32W1 has a dual-core architecture and has two separated power domains:

- The main domain for the Cortex M33
- The Radio domain which comprises the Cortex M3 core and the NBU (Narrow Band Unit).

The two power domains can go into or exit independently different low-power modes, namely Wait for instruction (WFI), Deep-sleep mode, Power-down mode, and Deep Power-down mode.

In Wait For Interrupt (WFI) mode, the CPU core is powered ON but is in an idle mode with the clock turned OFF.

In Deep Sleep mode, the fast clock is turned off, and the CPU along with the main power domain are placed into a retention state, with the voltage being scaled down to support state retention only. As no high frequency clock runs in this mode, the voltage applied on the power domain can be reduced to reduce leakage on the hardware logic. This reduces the overall power consumption in the Deep Sleep mode. When waking up from Deep Sleep mode, the core voltage is increased back to nominal voltage, the fast clock (FRO) is turned back on, and the peripheral in this domain can be reused as normal.

In Power-down mode, both the clock, and power are shut off to the CPU and the main peripheral domain. SRAM is retained, but register values are lost. The SDK power manager handles the restore of the processor registers and dependencies such as interrupt controller and similar ones transparently from the application.

In Deep Power-down mode the SRAM is not retained. This is the lowest power mode available. It is exited through the reset sequence.

Parent topic:*Low-Power Management*

When/how to enter low power To enable low power at application level, the gAppLowpowerEnabled_d define should be set to 1 in app_preinclude.h file.

The system should enter low power when the entire system is idle, and all software layers agree on that. The device enters low power by calling the PWR_EnterLowPower function.

For FreeRTOS applications, the low-power entry point is placed in the FreeRTOS idle task, which has the lowest priority in the system. From that task, the vPortSuppressTicksAndSleep function is called, which at its turn, calls the PWR_EnterLowPower to enter low power.

For the BareMetal examples, the application low power entry point is placed in the main function and is called when there are no messages to be processed by other tasks.

The wake-up sources that can be configured for the application are UART or button. Note that Low-power timer wake-up source and wake-up from the Radio domain are directly enabled from the Connectivity framework.

Each software layer/entity running on the system can prevent it from entering low power by calling *PWR_LowPowerEnterCritical* function. The system stays awake until all software layers that called *PWR_LowPowerEnterCritical* call back *PWR_LowPowerExitCritical* and the system reaches the low-power entry point.

When going to low power, the SDK Power Manager selects the best low-power mode that fits all the constraints.

The default low-power mode for each application is Deep-sleep mode. Users can change the behavior by setting a new low-power constraint for the application.

For example, if the low power constraint set from the application is Deep-sleep mode, and no other constraint is set, the**SDK Power Manager** selects Deep-sleep the next time the device enters low power. However, there might be a case when a WFI constraint is set (*PWR_SetLowPowerModeConstraint (PWR_WFI)*) by another component, such as the SecLib module that operates Hardware encryption. In such cases, the **SDK Power Manager** selects this WFI mode until the constraint is released by the SecLib module (*PWR_ReleaseLowPowerModeConstraint(PWR_WFI)*).

If it is required to change the mode from Deep-sleep to Power-down mode, the deep sleep constraint (*PWR_ReleaseLowPowerModeConstraint(PWR_DeepSleep)*) must be released and the power-down constraint (*PWR_SetLowPowerModeConstraints(PWR_PowerDown)*) must be set. The **SDK Power Manager** selects the Power-down mode when the device enters low power.

Parent topic:*Low-Power Management*

Over the Air Programming (OTAP) This chapter contains a detailed description of the Over The Air Programming capabilities of the Bluetooth Low Energy Host Stack enabled by dedicated GATT Service/Profile and of the support modules needed for OTA programming.

The image transfer is done using a dedicated protocol which is designed to run on both the Bluetooth Low Energy transport and serial transport.

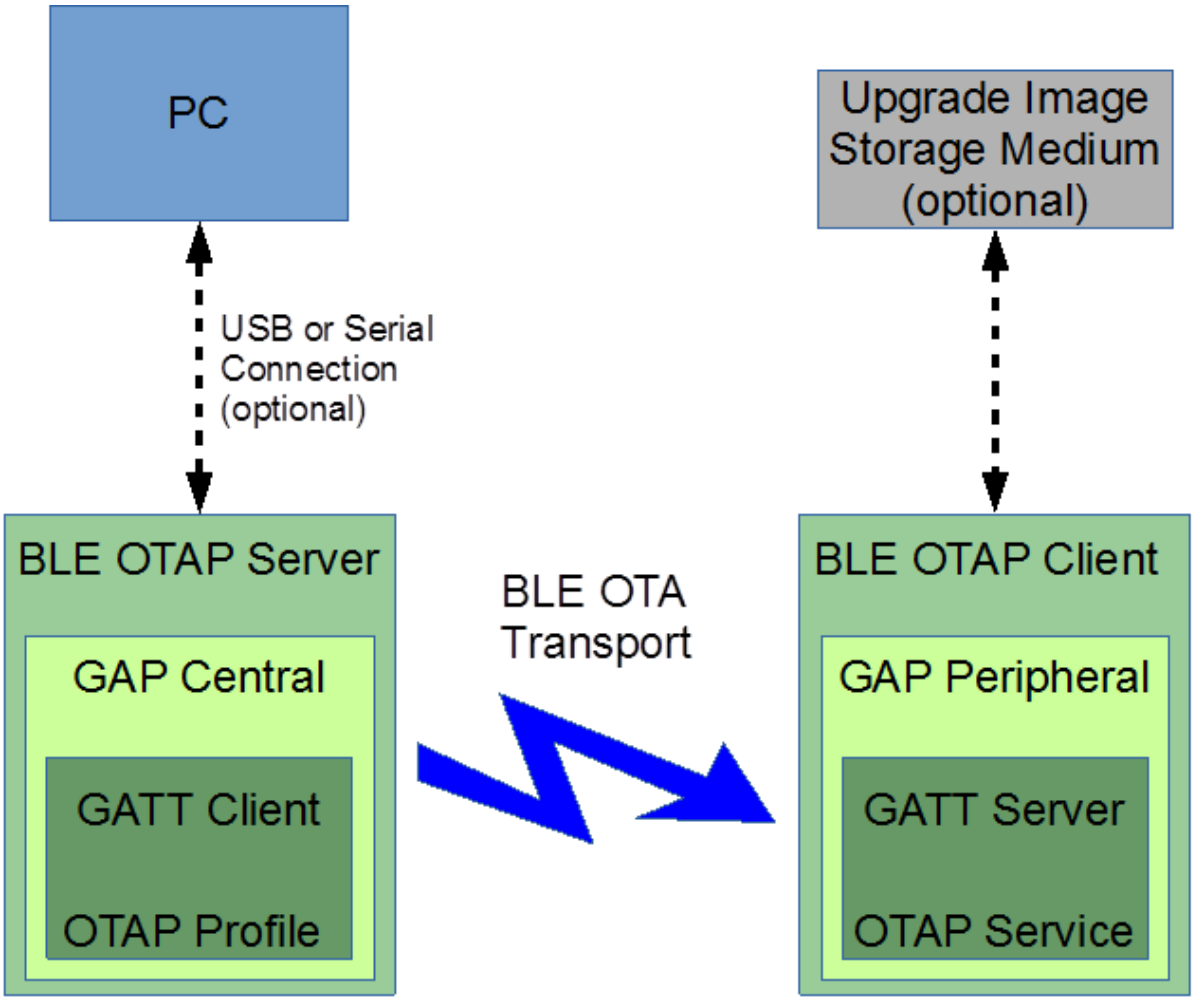
The container for the upgrade image is an image file which has a predefined format which is described in detail. The image file format is independent of the protocol but must contain information specific to the image upgrade infrastructure on an OTAP Client device. Detailed information on how to build an image file starting from a generic format executable generated by an embedded cross-compiling toolchain is shown.

The demo applications implement a typical scenario where a new image is sent from a PC via serial interface to a Bluetooth Low Energy OTAP Server and then over the air to an OTAP Client which is the target of the upgrade image. There are 3 applications involved in the OTAP demo: 1 PC application which builds the image file and serves it to the embedded OTAP Server and 2 embedded applications (OTAP Server and OTAP Client). This chapter contains enough information for building Bluetooth Low Energy OTAP applications which implement different image upgrade scenarios specific to other use cases.

General functionality A Bluetooth Low Energy OTAP system consists of an OTAP Server and an OTAP Client which exchange an image file over the air using the infrastructure provided by Bluetooth Low Energy (GAP, GATT, SM) via a custom GATT Service and GATT Profile. Additionally, a third application may be used to serve an image to the embedded OTAP Server.

The OTAP Server runs on the GATT Client via the Bluetooth Low Energy OTAP Profile and the OTAP Client runs on the GATT Server via the Bluetooth Low Energy OTAP Service. For the moment the OTAP Server runs on the GAP Central and the OTAP Client runs on the GAP Peripheral.

The Figure shows a typical image upgrade scenario.



Parent topic:*Over the Air Programming (OTAP)*

Bluetooth Low Energy OTAP service-profile The Bluetooth Low Energy OTAP Service is implemented using the GATT Server which runs on the OTAP Client (GAP Peripheral).

The Bluetooth LE OTAP Service does not require any other Bluetooth LE services because it is a custom service it has a 128-bit UUID. The service has 2 custom characteristics which also have 128-bit UUIDs.

The service must be included in the GATT database of the GATT Server as described in *Creating GATT database* of this document.

OTAP service and characteristics The OTAP Service has a custom 128-bit UUID which is shown below. The UUID is based on a base 128-bit UUID used for Bluetooth LE custom services and characteristics. These are shown in the tables below.

Service	128-bit UUID
Base Bluetooth Low Energy	00000000 -ba5e-f4ee-5ca1-eb1e5e4b1ce0

The OTAP Service custom 128-bit UUID is built using the base UUID by replacing the most significant 4 bytes which are 0 with a value specific to the OTAP Service which is 01FF5550 in hexadecimal format.

Service	UUID (128-bit)
Bluetooth LE OTAP Service	01ff5550 -ba5e-f4ee-5ca1-eb1e5e4b1ce0

The Bluetooth LE OTAP Service Characteristics UUIDs are built the same as the Bluetooth LE OTAP Service UUID starting from the base 128-bit UUID but using other values for the most significant 4 bytes.

Characteristic	UUID (128-bit)	Properties	Descriptors
Bluetooth LE OTAP Control Point	01ff5551 -ba5e-f4ee-5ca1-eb1e5e4b1ce0	Write, Indicate	CCC
Bluetooth LE OTAP Data	01ff5552 -ba5e-f4ee-5ca1-eb1e5e4b1ce0	Write Without Response	-

Both characteristics are implemented as variable length characteristics.

The Bluetooth LE OTAP Control Point Characteristic is used for exchanging OTAP commands between the OTAP Server and the OTAP Client. The OTAP Client sends commands to the OTAP Server using ATT Notifications for this characteristic and the OTAP Server sends commands to the OTAP Client by making ATT Write Requests to this characteristic. Both ATT Writes and ATT Notifications are acknowledged operations via ATT Write Responses and ATT Confirmations.

The Bluetooth LE OTAP Data characteristic is used by the OTAP Server to send parts of the OTAP image file to the OTAP Client when the ATT transfer method is chosen by the application. The ATT Write Commands (GATT Write Without Response operation) is not an acknowledged operation.

The Bluetooth LE OTAP service and characteristics 128-bit UUIDs are defined in the *gatt_uuid128.h* just as shown below.

```
UUID128(uuid_service_otap, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA,␣
↪0x50, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_control_point, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4,␣
↪0x5E, 0xBA, 0x51, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_data, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E,␣
↪0xBA, 0x52, 0x55, 0xFF, 0x01)
```

The service is included into the GATT database of the device. It is declared in the *gatt_db.h* file as shown below.

```
PRIMARY_SERVICE_UUID128(service_otap, uuid_service_otap)
CHARACTERISTIC_UUID128(char_otap_control_point, uuid_char_otap_control_point,␣
↪(gGattCharPropWrite_c | gGattCharPropIndicate_c))
VALUE_UUID128_VARLEN(value_otap_control_point, uuid_char_otap_control_point,␣
↪(gPermissionFlagWritable_c), 16, 16, 0x00)
CCCD(cccd_otap_control_point)
CHARACTERISTIC_UUID128(char_otap_data, uuid_char_otap_data, (gGattCharPropWriteWithoutRsp_
↪c))
VALUE_UUID128_VARLEN(value_otap_data, uuid_char_otap_data, (gPermissionFlagWritable_c),␣
↪gAttMaxMtu_c - 3, gAttMaxMtu_c - 3, 0x00)
```

The Bluetooth LE OTAP Control Point characteristic should be large enough for the longest command which can be exchanged between the OTAP Server and The OTAP Client.

The Bluetooth LE OTAP Data characteristic should be large enough for the longest data chunk command the OTAP Client expects from the OTAP Server to be sent via ATT. The maximum length of the OTAP Data Characteristic value is ATT_MTU- 3. 1 byte is used for the ATT OpCode and 2 bytes are used for the Attribute Handle when performing a Write Without Response, the only operation permitted for this characteristic value.

Parent topic: Bluetooth Low Energy OTAP service-profile

OTAP server and OTAP client interactions The OTAP Server application scans for devices advertising the OTAP Service. When it finds one it connects to that device and notifies it of the available image files or waits for requests regarding available image files. The behavior is specific to each application which needs the OTAP functionality. The Bluetooth LE OTAP Protocol described below details how to do this.

After an OTAP Server (GAP Central, GATT Client) connects to an OTAP Client (GAP Peripheral, GATT Server) it scans the device database and identifies the handles of the OTAP Control Point and OTAP Data characteristics and their descriptors. Then it writes the CCC Descriptor of the OTAP Control point to allow the OTAP Client to send it commands via ATT Indications. It can send commands to the OTAP Client by using ATT Write Commands to the OTAP Control Point characteristic.

After the connection is established, if the OTAP Client wants to use the L2CAP CoC transfer method it must register a L2CAP PSM with the OTAP Server.

The OTAP Client only starts any image information request or image transfer request procedures only after the OTAP Server writes the OTAP Control Point CCCD to ensure there is bidirectional communication between the devices.

Parent topic: Bluetooth Low Energy OTAP service-profile

Parent topic:*Over the Air Programming (OTAP)*

Bluetooth LE OTAP protocol The protocol consists of a set of commands (messages) which allow the OTAP Client to request or be notified about the available images on an OTAP Server and to request the transfer of parts of images from the OTAP Server.

All commands with the exception of the image data transfer commands are exchanged through the OTAP Control Point characteristic of the OTAP Service. The data transfer commands are sent only from the OTAP Server to the OTAP Client either via the OTAP Data characteristic of the OTAP Service or via a dedicated Credit Based Channel assigned to a L2CAP PSM.

Protocol design considerations The OTAP Client is a GAP Peripheral device, and therefore has limited resources. This is why the OTAP Protocol was designed in such a way that it is at the discretion of the OTAP Client if, when, how fast and how much of an available upgrade image is transferred from the OTAP Server. The OTAP Client also decides which is the image transfer method based on its capabilities. Two image transfer methods are supported at this moment: the ATT Transfer Method and the L2CAP PSM CoC Transfer Method.

The ATT Transfer Method is supported by all devices which support Bluetooth LE but it has the disadvantage of a small data payload size and a larger Bluetooth LE stack protocols overhead leading to a lower throughput. This disadvantage has been somewhat reduced by the introduction of the Long Frames feature in the Bluetooth LE specification 4.2 Link Layer which allows for a larger ATT_MTU value. The L2CAP PSM CoC Transfer Method is an optional feature available for devices running a Bluetooth stack version 4.1 and later. The protocol overhead is smaller and the data payload is higher leading to a high throughput. The L2CAP PSM Transfer Method is the preferred transfer method and it is available on all Bluetooth LE Devices if the application requires it.

Based on application requirements and device resources and capabilities the OTAP Clients can request blocks of OTAP images divided into chunks. To minimize the protocol overhead and maximize throughput an OTAP Client makes a data block request specifying the block size and the chunk size and the OTAP Server sends the requested data chunks (which have a sequence number) without waiting for confirmation. The block size, chunk size and number of chunks per block are limited and suitable values must be used based on application needs.

The OTAP Client can stop or restart an image block transfer at any time if the application requires it or a transfer error occurs. The OTAP Server implementation can be almost completely stateless. The OTAP Server does not need to remember what parts of an image have been transferred, this is the job of the OTAP Client which can request any part of an image at any time. This allows it

to download parts of the image whenever and how fast its resources allow it. The OTAP Server simply sends image information and image parts on request.

The Bluetooth LE OTAP Protocol is designed to be used not only on Bluetooth LE transport medium but on any transport medium, for example a serial communication interface or another type of wireless interface. This may be useful when transferring an upgrade image from a PC or a mobile device to the OTAP Server to be sent via Bluetooth LE to the OTAP Clients which require it. In the OTAP Demo Applications the embedded OTAP Server relays OTAP commands between an OTAP Client and a PC via a serial interface and using a FSCI type protocol. Effectively the OTAP Client downloads the upgrade image from the PC and not from the OTAP Server. Other transfer methods may be used based on application needs.

Parent topic: Bluetooth LE OTAP protocol

Bluetooth Low Energy OTAP commands The Bluetooth LE OTAP Commands general format is shown below. A command consists of two parts, a Command ID, and a Command Payload as shown in the table below.

Field Name	CmdId	CmdPayload
Size (Bytes)	1	variable

Commands are sent over the transport medium starting with the Command ID and continuing with the Command Payload.

All multibyte command parameters in the Command Payload are sent in a least significant octet first order (little endian).

A summary of the commands supported by the Bluetooth LE OTAP Protocol is shown in the table below. Each of the commands is then detailed in its own section.

CmdId	Command Name
0x01	New Image Notification
0x02	New Image Info Request
0x03	New Image Info Response
0x04	Image Block Request
0x05	Image Chunk
0x06	Image Transfer Complete
0x07	Error Notification
0x08	Stop Image Transfer

New image notification command This command can be sent by an OTAP Server to an OTAP Client, usually immediately after the first connection, to notify the OTAP Client of the available images on the OTAP Server.

Table 14: New Image Notification Command Parameters

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId + Payload)
0x01	New Image Notification	S->C	ImageId	2	Short image identifier used for transactions between the OTAP Server and OTAP Client. Should be unique for all images on a server.	15
			ImageVersion	8	Image file version. Contains sufficient information to identify the target hardware, stack version and build version.	
			ImageFileSize	4	Image file size in bytes.	

The *ImageId* parameter should not be '0x0000', which is the reserved value for the current running image or 0xFFFF, which is the reserved value for "no image available".

Parent topic: Bluetooth Low Energy OTAP commands

New image info request command This command can be sent by an OTAP Client to an OTAP Server to inquire about available upgrade images on the OTAP Server.

CmdId	Name	Dir	Parameters	Param Size (Bytes	Description	Total Size (CmdId+Payload)
0x02	New Image Info Request	C->S	CurrImageId	2	Id of the currently running image. Should be 0x0000.	11
0x02	New Image Info Request	C->S	CurrImageV	8	Version of the currently running image. A value of all zeroes signals that the client is looking for all images available on an OTAP Server. A value of all zeroes requests information about all images on the server.	11

- The *CurrImageId* parameter should be set to 0x0000 to signify the current running image.
- The *CurrImageVer* parameter should contain sufficient information about the target device for the OTAP Server to determine if it has an upgrade image available for the requesting OTAP Client.
- A value of all zeroes for the *CurrImageVer* means that an OTAP Client is requesting information about all available images on an OTAP Server and the OTAP Server should send a New Image Info Response for each image.

Parent topic: Bluetooth Low Energy OTAP commands

New image info response command This command is sent by the OTAP Server to the OTAP Client as a response to a New Image Information Request Command.

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId+Payload)
0x03	New Image Info Response	S->C	ImageId	2	Image Id. Value 0xFFFF is reserved as “no image available”	15
0x03	New Image Info Response	S->C	ImageVersion	8	Image file version	15
0x03	New Image Info Response	S->C	4 Image file size.	4	Image file size	15

The *ImageId* parameter with a value of 0xFFFF is reserved for the situation where no upgrade image is available for the requesting device.

Parent topic: Bluetooth Low Energy OTAP commands

Image block request command This command is sent by the OTAP Client to the OTAP Server to request a part of the upgrade image after it has determined the OTAP Server has an upgrade image available.

When an OTAP Server Receives this command it should stop any image file chunk transfer sequences in progress.

CmdId	Nam	Dir	Parameters	Para Size	Description	Total Size (CmdId+Payload)
0x04	Image Bloc Reques	C->S	StartPositionBlockSizeChunkSizeTransferMethodL2ca	4421	Start position of the image block to be transferred.Requested total block size in bytes.Should be optimized to the Transfer Channel type. The maximum number of chunks per block is 256. Value is inbytes.0x00 - ATT 0x01 – L2CAP PSM Credit based channel0x0004 - ATT Other values – PSM for credit based channels	16

The *ImageId* parameter contains the ID of the upgrade image.

The *StartPosition* parameter specifies the location in the image upgrade file at which the requested block starts.

The *BlockSize* and *ChunkSize* parameters specify the size in bytes of the block to be transferred and the size of the chunks into which a block is separated. The *ChunkSize* value must be chosen in such a way that the total number of chunks can be represented by the *SeqNumber* parameter of the Image Chunk Command. At the moment this parameter is 1 byte in size so there are a maximum of 256 chunks per block. The chunk sequence number goes from 0 to 255 (0x00 to 0xFF). If this condition is not met or the requested block is not entirely into the image file bounds an error is sent to the OTAP Client when the OTAP Server receives this misconfigured Image Block Request Command.

The maximum value of the *ChunkSize* parameter depends on the maximum ATT_MTU and L2CAP_MTU supported by the Bluetooth LE stack version and implementation.

The *TransferMethod* parameter is used to select the transfer method which can be ATT or L2CAP PSM CoC. The *L2capChannelOrPsm* parameter must contain the value 0x0004 for the ATT transfer method and another value representing the chosen PSM for the L2CAP PSM transfer method. The default PSM for the Bluetooth LE OTAP demo applications is 0x004F for both the OTAP Server and the OTAP Client although the specification allows different values at the 2 ends of the L2CAP PSM connection. The PSM must be in the range reserved by the Bluetooth specification which is 0x0040 to 0x007F.

The optimal value of the *ChunkSize* parameter depends on the chosen transfer method and the Link Layer payload size. Ideally it must be chosen in such a way that full packets are sent for every chunk in the block.

The default Link Layer payload is 27 bytes form which we subtract 4 for the L2CAP layer and 3 for the ATT layer (1 for the ATT Cmd Opcode and 2 for the Handle) leaving us with a 20 byte OTAP protocol payload. From these 20 bytes we subtract 1 for the OTAP CmdId and 1 for the chunk sequence number leaving us with an optimum chunk size of 18 for the ATT transfer method – which is the default in the demo applications. For the L2CAP PSM transfer method the chosen default chunk size is 111. This was chosen so as a chunk fits exactly 5 link layer packets. The default L2CAP payload of 23 (27 - 4) multiplied by 5 gives us 115 from which we subtract 2 bytes for the SDU Length (which is only sent in the first packet), 1 byte for the OTAP CmdId and 1 byte for the chunk sequence number which leaves exactly 111 bytes for the actual payload.

If the Link layer supports Long Frames feature then the chunk size should be set according to the negotiated ATT MTU for the ATT transfer method. From the negotiated ATT MTU (*att_mtu*) subtract 3 bytes for the ATT layer (1 for the ATT Cmd Opcode and 2 for the Handle) then subtract 2 bytes for the OTAP protocol (1 for the CmdId and 1 for the chunk sequence number) to determine the optimum chunk size (*optimum_att_chunk_size = att_mtu – 3 – 2)*. For the L2CAP PSM transfer method the chunk size can be set based on the maximum L2CAP SDU size (*max_l2cap_sdu_size*) from which 4 bytes should be subtracted, 2 for the SDU Length and 2 for the OTAP protocol (*optimum_l2cap_chunk_size = max_l2cap_sdu_size – 3 – 2*). In some particular cases reducing the L2CAP chunk size could lead to better performance. If the L2CAP chunk size needs to be reduced it should be reduced so it fits exactly a number of link layer packets. An example of how to compute an optimal reduced L2CAP chunk size is given in the previous paragraph.

Parent topic: Bluetooth Low Energy OTAP commands

Image chunk command One or more Image Chunk Commands are sent from the OTAP Server to the OTAP Client after an Image Block Request is received by the former. The image chunks are sent via the ATT Write Without Response mechanism if the ATT transfer method is chosen and directly via L2CAP if the L2CAP PSM CoC transfer method is chosen.

CmdId	Name	Dir	Parameters	Parameter Size	Description	Total Size (CmdId+Payload)
0x05	Image Chunl	S->C	SeqNumber-Data	1variable	In the range 0 -> BlockSize/ChunkSize - calculated by Server, checked by Client. The command code is present even when ATT is used.Actual data	3 or more

The *SeqNumber* parameter is the chunk sequence number and it has incremental values from 0 to 255 (0x00 to 0x FF) for a maximum of 256 chunks per block.

The *Data* parameter is an array containing the actual image part being transferred starting from the *BlockStartPosition + SeqNumber ChunkSize* position in the image file and containing *ChunkSize* or less bytes depending on the position in the block. Only the last chunk in a block can have less than *ChunkSize*bytes in the Image Chunk Command data payload.

Parent topic: Bluetooth Low Energy OTAP commands

Image transfer complete command This command is sent by the OTAP Client to the OTAP Server when an image file has been completely transferred and its integrity has been checked.

CmdId	Name	Dir	Parameters	Param Size	Description	Total Size (CmdId+Payload)
0x06	Image Transfer Complete	C->S	ImageIdStatus	21	Image IdStatus of the image transfer. 0x00 - Success	4

The *ImageId* parameter contains the ID of the image file that was transferred.

The *Status* parameter is 0x00 (Success) if image integrity and possibly other checks have been successfully made after the image is transferred and another value if integrity or other kind of errors have occurred.

If the status is 0x00 the OTAP Client can trigger the Bootloader to start flashing the new image. The image flashing should take about 15 seconds for a 160 KB flash memory.

Parent topic: Bluetooth Low Energy OTAP commands

Error notification command This command can be sent by both the OTAP Server and the OTAP Client when an error of any kind occurs. When an OTAP Server receives this command, it should stop any image file chunk transfer sequences in progress.

CmdId	Name	Dir	Parameters	Param Size (Bytes	Description	Total Size (CmdId+Payload)
0x07	Error Notification	Bidi	CmdIdErrorStatus	11	Id of the command which generated the error.Error Status: Examples: out of image bounds, chunk too small, chunk too large, image verification failure, bad command format, image not available, unknown command	3

The *CmdId* parameter contains the ID of the command which caused the error (if applicable).

The *ErrorStatus* parameter contains the source of the error. All error statuses are defined in the *otapStatus_t* enumerated type in the *otap_interface.h* file.

Parent topic: Bluetooth Low Energy OTAP commands

Stop image transfer command This command is sent from the OTAP Client to the OTAP Server whenever the former wants to stop the transfer of an image block which is currently in progress, or from OTAP Server to the OTAP Client when the image transfer is stopped from application (Test Tool).

CmdId	Name	Dir	Parameters	Param Size	Description	Total Size (CmdId+Payload)
0x08	Stop Image Transfer	C->S	ImageId	2	Image Id	3

The *ImageId* parameter contains the ID of the image being transferred.

Parent topic: Bluetooth Low Energy OTAP commands

Parent topic: Bluetooth LE OTAP protocol

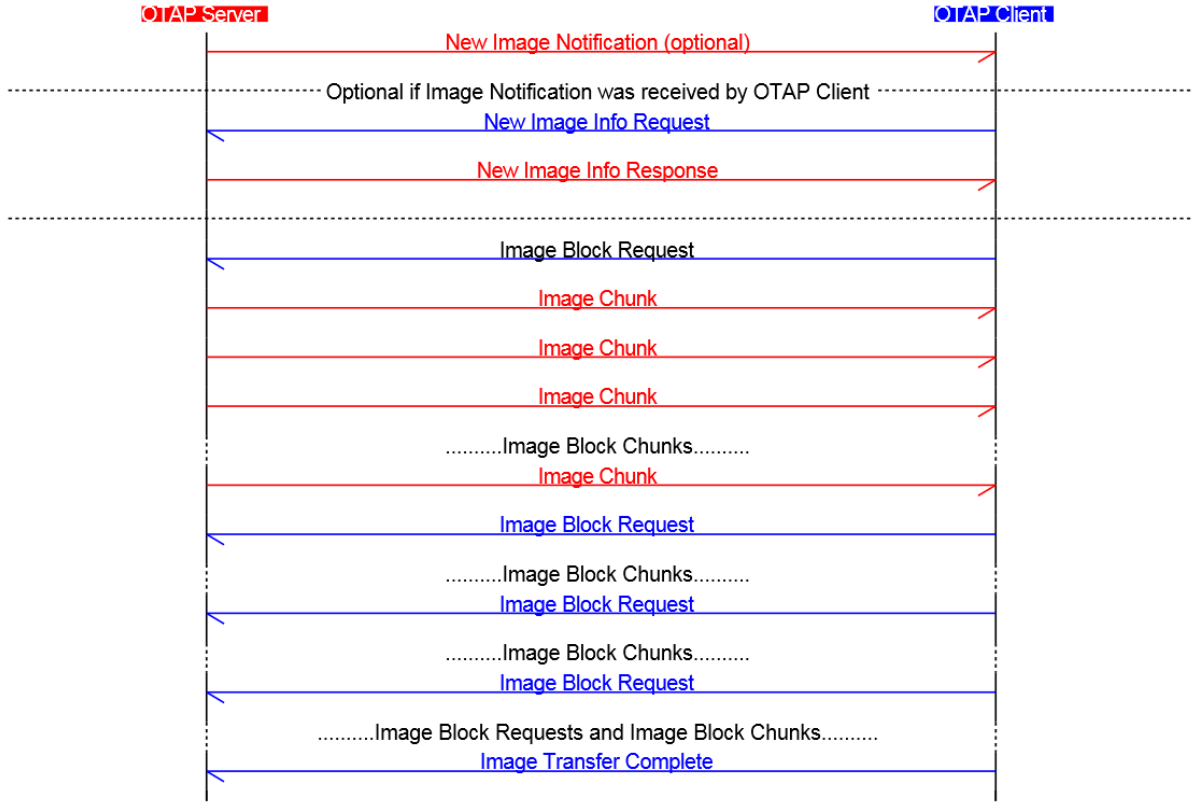
OTAP client–server interactions The interactions between the OTAP Server and OTAP Client start immediately after the connection, discovery of the OTAP Service characteristics and writing of the OTAP Control Point CCC Descriptor by the OTAP Server.

The first command sent could be a New Image Notification sent by the OTAP Server to the OTAP Client or a New Image Info Request sent by the OTAP Client. The OTAP Server can respond with a New Image Info response if it has a new image for the device which sent the request (this can be determined from the *ImageVerison* parameter). The best strategy depends on application requirements.

After the OTAP Client has determined that the OTAP Sever has a newer image it can start downloading the image. This is done by Sending Image Block Request commands to retrieve parts of the image file. The OATP Server answers to these requests with one or more Image Chunk Commands via the requested transfer method or with an Error Notification if there are improper parameters in the Image Block Request. The OTAP Client makes as many Image Block Requests as it is necessary to transfer the entire image file.

The OTAP Client decides how often Image Block Request Commands are sent and can even stop a block transfer which is in progress via the Stop Image Transfer Command. The OTAP Client is in complete control of the image download process and can stop it and restart it at any time based on its resources and application requirements.

A typical **Bluetooth LE OTAP Image Transfer** scenario is shown in the message sequence chart Figure.



Parent topic: Bluetooth LE OTAP protocol

Parent topic:*Over the Air Programming (OTAP)*

Bluetooth Low Energy OTAP image file format The Bluetooth LE OTAP Image file has a binary file format. It is composed of a header followed by a number of sub-elements. The header describes general information about the file. There are some predefined sub-elements of a file but an end manufacturer could add manufacturer-specific sub-elements. The header does not have details of the sub-elements. Each element is described by its type.

The general format of an image file is shown in the table below.

Image File Element	Value Field Length (bytes)	Description
Header	Variable	The header contains general information about the image file.
Upgrade Image Sub-element	Variable	This sub-element contains the actual binary executable image, which is copied into the flash memory of the target device. The maximum size of this sub-element depends on the target hardware.
Image File CRC Sub-element	2	This is a 16-bit CCITT type CRC which is calculated over all elements of the image file with the exception of the Image File CRC sub-element itself. This must be the last sub-element in an image file.

Each sub-element in a Bluetooth LE OTAP Image File has a Type-Length-Value (TLV) format. The type identifier provides forward and backward compatibility as new sub-elements are introduced. Existing devices that do not understand newer sub-elements may ignore the data.

The following table shows the general format of a Bluetooth LE Image File sub-element.

Sub-field	Size (Bytes)	Format	Description
Type	2	uint16	Type Identifier – determines the format of the data contained in the value field
Length	4	uint32	Length of the *Value* field of the sub-element.
Value	variable	uint8[]	Data payload

Some sub-element type identifiers are reserved while others are left for manufacturer-specific use. The table below shows the reserved type identifiers and the manufacturer-specific ranges.

Type Identifiers	Description
0x0000	Upgrade Image
0x0001 – 0xefff	Reserved
0xf000 – 0xffff	Manufacturer-Specific Use

The OTAP Demo applications use two of the manufacturer-specific sub-element type identifiers while the rest remain free to use. The two are shown in the table below along with a short description.

Manufacturer-specific Type Identifiers	Sub-element Name	Notes
0xf000	Sector Bitmap	Bitmap that signals the bootloader the sectors of the internal flash, which should be overwritten and which should remain as is.
0xf100	Image File CRC	16-bit CRC that is computed over the image file with the exception of the CRC sub-element itself.

Bluetooth Low Energy OTAP header The format and fields of the Bluetooth Low Energy OTAP Header are summarized in the table below.

Octets	Data Types	Field Name	Mandatory/Optional
4	Unsigned 32-bit integer	Upgrade File Identifier	M
2	Unsigned 16-bit integer	Header Version	M
2	Unsigned 16-bit integer	Header Length	M
2	Unsigned 16-bit integer	Header Field Control	M
2	Unsigned 16-bit integer	Company Identifier	M
2	Unsigned 16-bit integer	Image ID	M
8	8 byte array	Image Version	M
32	Character string	Header String	M
4	Unsigned 32-bit integer	Total Image File Size	M

The fields are shown in the order they are placed in memory from the first location to the last.

The total size of the header without the optional fields (if defined by the *Header Field Control*) is 58 bytes.

All the fields in the header have a little endian format with the exception of the *Header String* field which is an ASCII character string.

A packed structure type definition for the contents of the Bluetooth LE OTAP Header can be found in the *otap_interface.h* file.

Upgrade file identifier Fixed value 4 byte field used to identify the file as being a Bluetooth LE OTAP Image File. The predefined value is "**0x0B1EF11E**".

Parent topic: Bluetooth Low Energy OTAP header

Header version This 2 byte field contains the major and minor version number. The high byte contains the major version and the low byte contains the minor version. The current value is "**0x0100**" with the major version "**01**" and the minor version **"00**". A change to the minor version means the OTA upgrade file format is still backward compatible, while a change to the major version suggests incompatibility.

Parent topic: Bluetooth Low Energy OTAP header

Header length Length of all the fields in the header including the ***Upgrade File Identifier*** field, ***Header Length*** field and all the optional fields. The value insulates existing software against new fields that may be added to the header. If new header fields added are not compatible with current running software, the implementations should process all fields they understand and then skip over any remaining bytes in the header to process the image or CRC sub-element. The value of the ***Header Length*** field depends on the value of the Header Field Control field, which dictates which optional header fields are included.

Parent topic: Bluetooth Low Energy OTAP header

Header field control This is a 2-byte bit mask that specifies the optional fields present in the OTAP Header.

In case no optional fields are defined, this whole field is reserved and should be set to "**0x0000**".

Parent topic: Bluetooth Low Energy OTAP header

Company identifier This is the company identifier assigned by the Bluetooth SIG. The Company Identifier used for the OTAP demo applications is "**0x01FF**".

Parent topic: Bluetooth Low Energy OTAP header

Image ID This is a unique short identifier for the image file. It is used to request parts of an image file. This number should be unique for all images available on a Bluetooth LE OTAP Server.

- The value 0x0000 is reserved for the current running image.
- The value 0xFFFF is reserved as a "no image available" code for New Image Info Response commands.

This field value must be used in the *ImageID* field in the *New Image Notification* and *New Image Info Response* commands.

Parent topic: Bluetooth Low Energy OTAP header

Image version This is the full identifier of the image file. It should allow a Bluetooth LE OTAP Client to identify the target hardware, stack version, image file build version, and other parameters if necessary. The recommended format of this field (which is used by the OTAP Demo applications) is shown below but an end device manufacturer could choose different format. The subfields are shown in the order they are placed in memory from the first location to the last. Each subfield has a little-endian format, if applicable.

Subfield	Size (bytes)	Format	Description
Build Version	3	uint8[]	Image build version.
Stack Version	1	uint8	0x41 for example for Bluetooth Low Energy Stack version 4.1.
Hardware ID	3	uint8[]	Unique hardware identifier.
End Manufacturer Id	1	uint8	ID of the hardware–specific to the end manufacturer

This field value must be used in the *ImageVersion* field in the New Image Notification and New Image Info Response commands.

Parent topic: Bluetooth Low Energy OTAP header

Header string This is a manufacturer-specific string that may be used to store other necessary information as seen fit by each manufacturer. The idea is to have a human readable string that can prove helpful during the development cycle. The string is defined to occupy 32 bytes of space in the OTAP Header. The default string used for the Bluetooth LE OTAP demo application is "BLE OTAP Demo Image File".

Parent topic: Bluetooth Low Energy OTAP header

Total image file size The value represents the total image size in bytes. This is the total of data in bytes that is transferred over-the-air from the server to the client. In most cases, the total image size of an OTAP upgrade image file is the sum of the sizes of the OTAP Header and all the other sub-elements on the file. If the image contains any integrity and/or source identity verification fields then the Total Image File Size also includes the sizes of these fields.

Parent topic: Bluetooth Low Energy OTAP header

Parent topic: Bluetooth Low Energy OTAP image file format

Parent topic:*Over the Air Programming (OTAP)*

Building Bluetooth Low Energy OTAP image file from SREC file A SREC (Motorola S-record) file is an ASCII format file which contains binary information. Common file extensions are: .srec, .s19, .s28, .s37 and others. Most modern compiler toolchains can generate an SREC format executable.

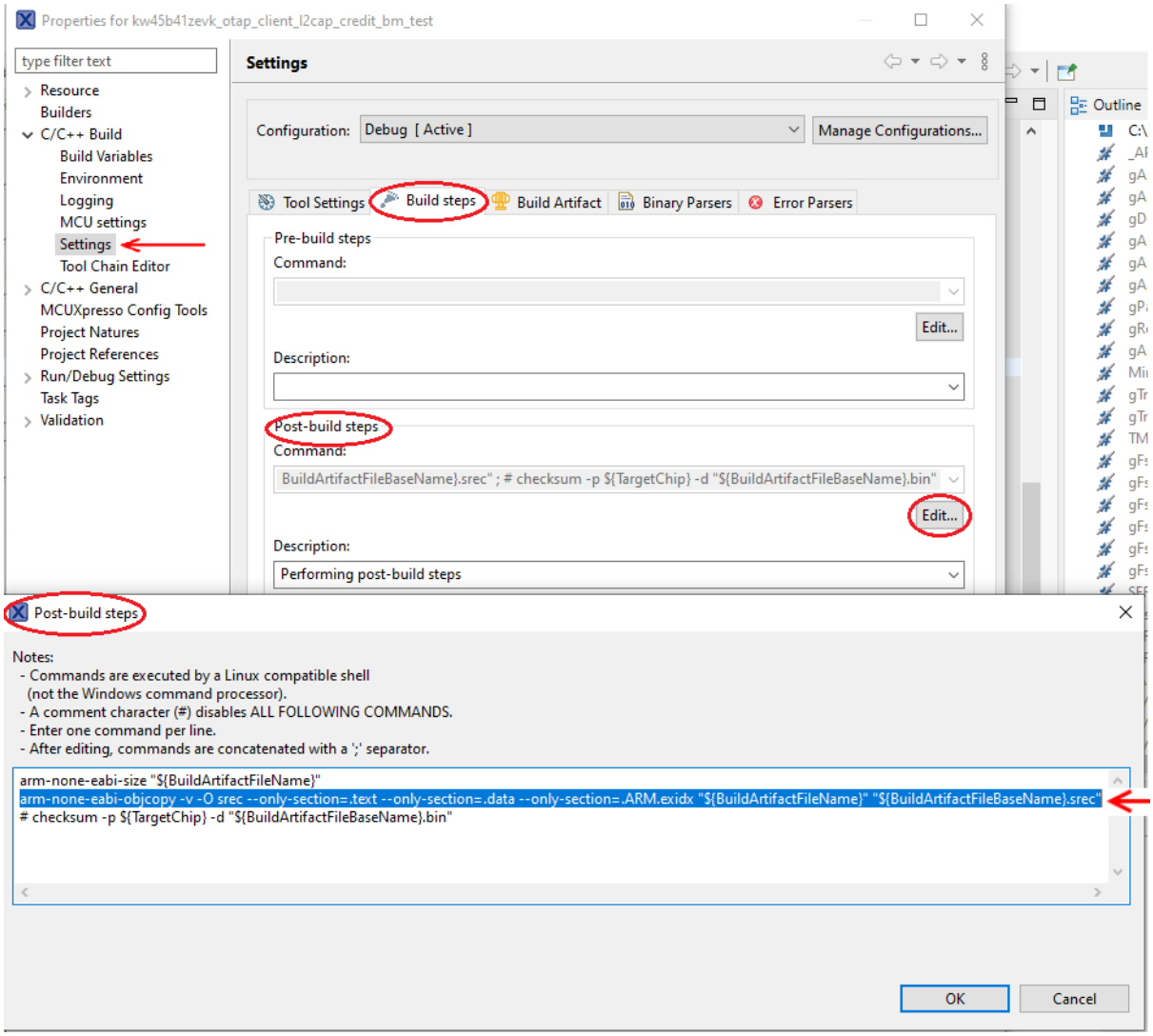
The steps described in this section enable the creation of a SREC file for your embedded application in IAR Embedded Workbench.

For this, open the target properties and go to the **Output Converter** tab. Activate the **Generate additional output** checkbox and choose the **Motorola** option from the **Output format** drop down menu. From the same pane you can also override the name of the output file. A screenshot of the described configuration is shown in Figure.

In MCUXpresso IDE, go to **Project properties** -> **Settings** -> **Build steps** window and press the “**Edit**” button for the Post-build steps. A Post-build steps window shows up in which the following command must be added:

```
arm-none-eabi-objcopy -v -O srec --only-section=.text --only-section=.data --only-section=.ARM.exidx
"${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.srec"
```

A snapshot of this window is shown in the Figure below.



The format of the SREC file is shown in table below. It contains lines of text called records which have a specific format. An example of the contents of a SREC file is shown below.

```
S02000006F7461705F636C69656E745F6174745F4672656552544F532E73726563A1
```

(continues on next page)

(continued from previous page)

```
S1130000F83F0020EB0500007506000075060000AF
S1130010750600007506000075060000075060000F0
S1130020750600007506000075060000075060000E0
S1130030750600007506000075060000075060000D0
S113004000000000000000000000000000000000AC
S113005000000000000000000000000000000000009C
.....
S2140117900121380004F05FF8002866D12A003100E4
S2140117A06846008804F022F8A689002E16D0002884
S2140117B014D12569278801A868A11022F7F782FCB1
S2140117C06B4601AA0121380004F045F800284CD1E7
S2140117D02A0031006846008804F008F8A68A002E20
```

All records start with the ASCII letter ‘S’ followed by an ASCII digit from ‘0’ to ‘9’. These two characters from the record type identify the format of the data field of the record.

The next 2 ASCII characters are 2 hex digits that indicate the number of bytes (hex digit pairs) which follow the rest of the record (address, data, and checksum).

The address that follows next can have 4, 6, or 8 ASCII hex digits, depending on the record type.

The data field is placed after the address and it contains 2 * n ASCII hex digits for ‘n’ bytes of actual data.

The last element of the S record is the checksum, which comprises 2 ASCII hex digits. The checksum is computed by adding all the bytes of the byte count, address, and data fields. Then the ones complement of the least significant octet of the sum is computed to determine the checksum.

Field	Record Type	Count	Address	Data	Checksum	Line Terminator
Format	“Sn”, n=0..9	ASCIIhex digits	ASCIIhex digits	ASCIIhex digits	ASCIIhex digits	“\r\n”
Length (characters)	2	2	4,6,8	*Count –*len*(Address) –*len(*Checksum)*	2	2

More details about the SREC file format can be found at this location: en.wikipedia.org/wiki/SREC_(file_format).

We are only interested in records that contain actual data. These are S1, S2, and S3 records. The other types of records can be ignored.

The S1, S2, and S3 records are used to build the Upgrade Image Sub-element of the image file simply by placing the record data at the location specified by the record address in the *Value* field of the Sub-element. It is recommended to fill all gaps in S record addresses with 0xFF.

To build an OTAP Image File from a SREC file, follow the procedure described below:

- Generate the SREC file by correctly configuring your toolchain to do so.
- Create the image file header.
  - Set the Image ID field of the header to be unique on the OTAP Server.
  - Leave the Total Image File Size Field blank for the moment.
- Create the Upgrade Image Sub-element
  - Read the S1, S2, and S3 records from the SREC file and place the binary record data to the record addresses in the *Value* filed of the sub-element. Fill all address gaps in the S records with 0xFF.

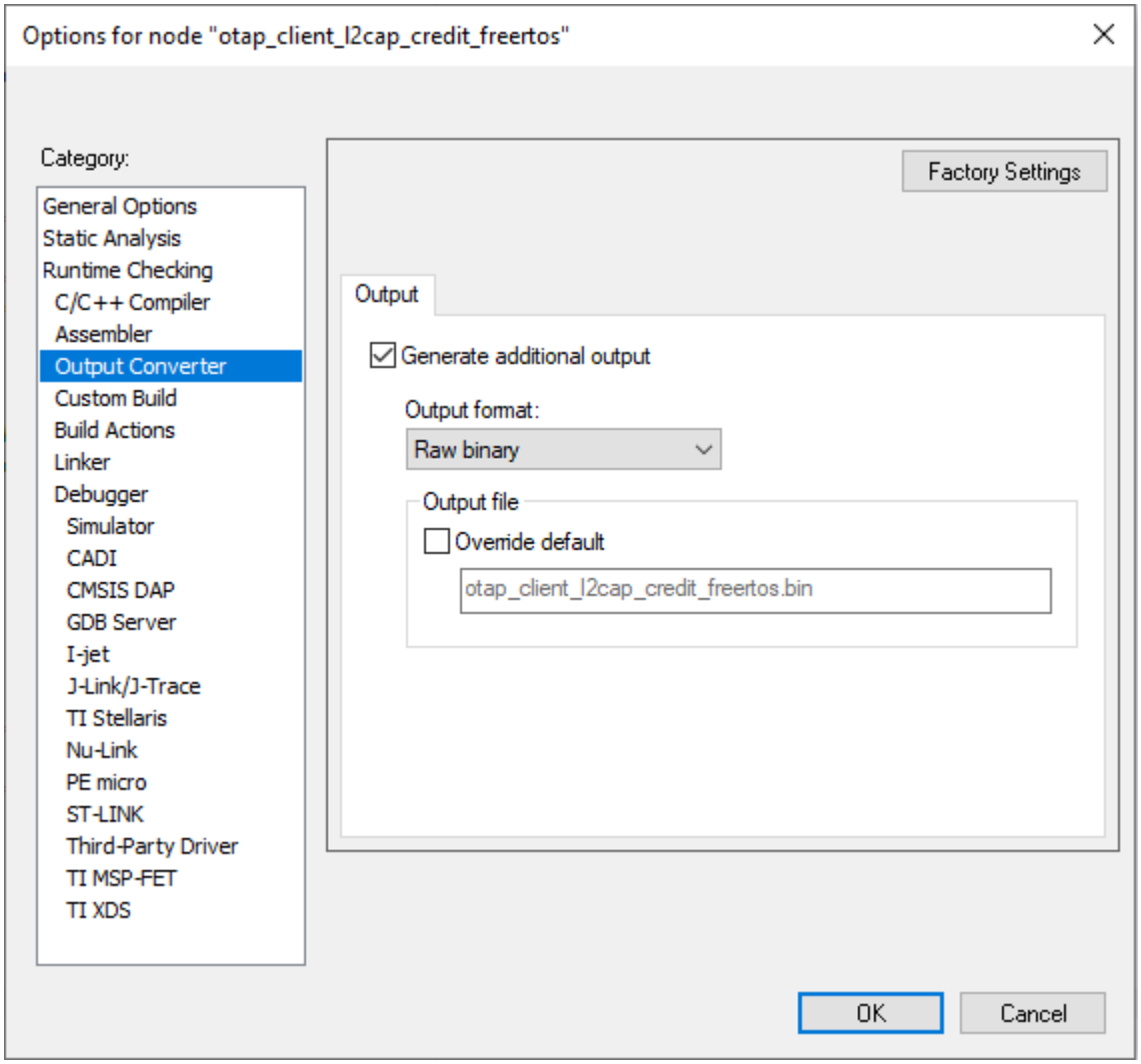
  - Fill in the *Length* field of the sub-element with the length of the written *Value* field.
- Create the Sector Bitmap Sub-element
  - A default working setting would be all byes 0xFF for the *Value* field of this sub-element.
- Create the Image File CRC Sub-element
  - Compute the total image file size as the length of the header + the length of all 3 sub-elements and fill in the appropriate filed in the header with this value.
  - Compute and write the *Value*field of this sub-element using the header and all sub-elements except this one.
  - The *OTA_CrcCompute()* function in the *OtaSupport.c* file can be used to incrementally compute the CRC.

If the Image ID is not available when the image file is created, then the CRC cannot be computed. It can be computed later after the Image ID is established and written in the appropriate field in the header.

Parent topic:*Over the Air Programming (OTAP)*

Building Bluetooth Low Energy OTAP image file from BIN file A BIN file is an binary file which contains an executable image. The most common extension for this type of file is .bin. Most modern compiler toolchains can output a BIN format executable.

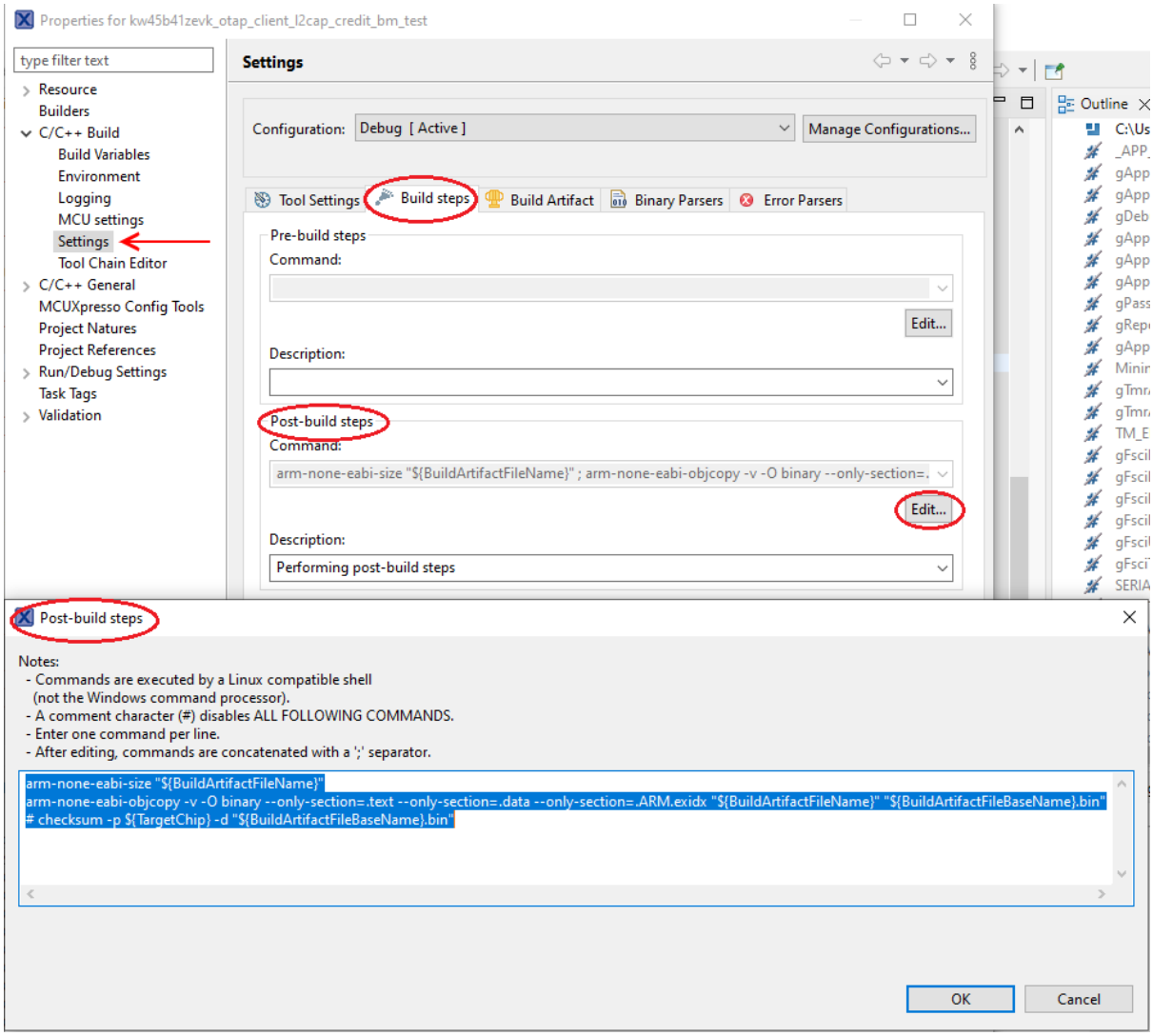
To enable the creation of a BIN file for your embedded application in IAR Embedded Workbench open the target properties and go to the *Output Converter* tab. Activate the *"Generate additional output"* checkbox and choose the *binary* option from the *"Output format"* drop down menu. From the same pane you can also override the name of the output file. The Figure shows a screenshot of the described configuration.



In MCUXpresso IDE, go to **Project properties** -> **Settings** -> **Build steps** window and press the "**Edit**" button for the Post-build steps. A Post-build steps window shows up in which the following command must be added:

```
arm-none-eabi-objcopy -v -O binary --only-section=.text --only-section=.data --only-section=.ARM.exidx
  "${BuildArtifactFileName}"
  "${BuildArtifactFileBaseName}.bin"
```

The Figure below shows the Build steps and Post-build steps in **Settings** window.



The format of the BIN file is very simple. It contains the executable image in binary format as is, starting from address 0 and up to the highest address. This type of file does not have any explicit address information.

To build an OTAP Image File from a BIN file, follow the procedure below:

- Generate the BIN file by correctly configuring your toolchain to do so.
- Create the image file header
  - Set the Image ID field of the header to be unique on the OTAP Server.
  - Leave the Total Image File Size field blank for the moment.
- Create the Upgrade Image Sub-element
  - Copy the entire contents of the BIN file as is into the *Value* filed of the sub-element.
  - Fill in the *Length* field of the sub-element with the length of the written *Value* filed.
- Create the Sector Bitmap Sub-element
  - A default working setting would be all bytes 0xFF for the *Value* field of this sub-element.
- Create the Image File CRC Sub-element
  - Compute the total image file size as the length of the header + the length of all 3 sub-elements and fill in the appropriate filed in the header with this value.

- Compute and write the *Value*field of this sub-element using the header and all sub-elements except this one.
- The *OTA_CrcCompute()* function in the *OtaSupport.c* file can be used to incrementally compute the CRC.

If the Image ID is not available when the image file is created, then the CRC cannot be computed. It can be computed later after the Image ID is established and written in the appropriate field in the header.

Parent topic:*Over the Air Programming (OTAP)*

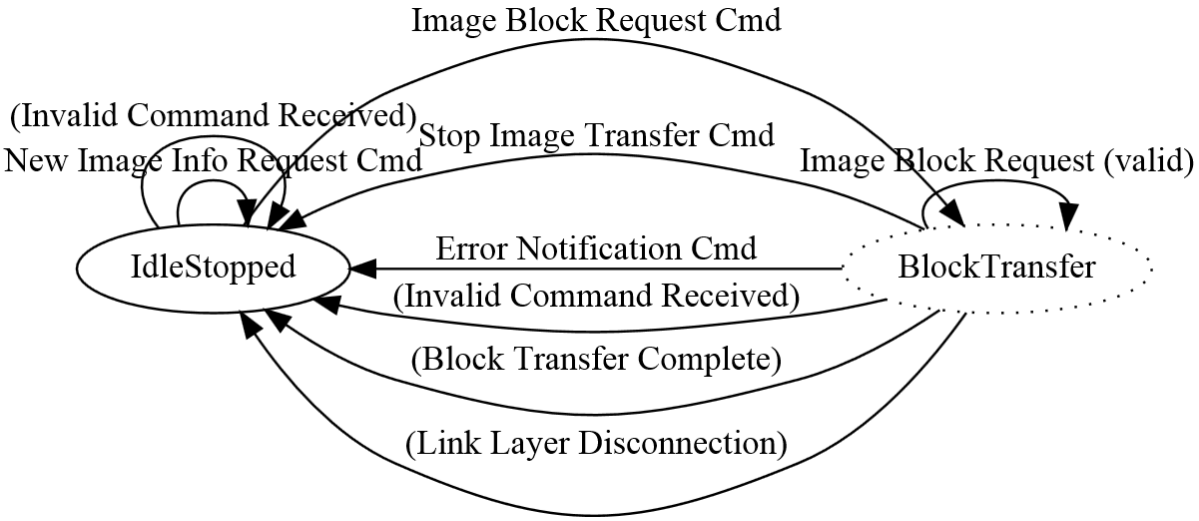
Bluetooth Low Energy OTAP application integration The Bluetooth Low Energy OTAP demo applications are standalone applications that only run the OTAP Server and the OTAP Client. In practice, however the OTAP Server and OTAP Client are used alongside with other functions. The OTAP functionality is used as a tool along with the main application on a device.

This section contains some guidelines on how to integrate OTAP functionality into other Bluetooth Low Energy applications.

OTAP server Before any OTAP transactions can be done the application which acts as an OTAP Server must connect to a peer device and perform ATT service and characteristic discovery. Once the handles of the OTAP Service, OTAP Control Point and OTAP Data characteristics and their descriptors are found then OTAP communication can begin.

A good starting point for OTAP transactions for both the OTAP Server and The OTAP client is the moment the Server writes the OTAP Control Point CCCD to receive ATT Indications from the OTAP Client. At that point the Server can send a New Image Notification to the Client if it finds out what kind of device the client is through other means than the OTAP server. How this can be done is entirely application-specific. If the OTAP Server does not know exactly what kind of device is the OTAP Client it can wait for the Client to send a New Image Info Request. Again, the best behavior depends on application requirements.

Once OTAP communication begins then the OTAP Server just has to wait for commands from the OTAP Client and answer them. This behavior is almost completely stateless. An example state diagram for the OTAP Server application is shown in Figure.



The OTAP Server waits in an idle state until a valid Image Block Request command is received and then moves to a pseudo-state and starts sending the requested block. The transfer can be interrupted by some commands (Error Notification, Stop Image Transfer, and so on) or other events (disconnection, user interruption, and so on).

The *otap_interface.h* file contains infrastructure for sending and receiving OTAP Commands and parsing OTAP image files. Packed structure types are defined for all OTAP commands and type enumerations are defined for command parameter values and some configuration values like the data payloads for the different transfer methods.

To receive ATT Indications and ATT Write Confirmations from the OTAP Client the OTAP Server application registers a set of callbacks in the stack. This is done in the *BluetoothLEHost_Initialized* function.

```
App_RegisterGattClientProcedureCallback (BleApp_GattClientCallback);
App_RegisterGattClientIndicationCallback (BleApp_GattIndicationCallback);
```

This *BleApp_GattIndicationCallback()* function is called when any attribute is indicated so the handle of the indicated attribute must be checked against a list of expected handles. In our case, we are looking for the OTAP Control Point handle that was obtained during the discovery procedure.

The *BleApp_GattIndicationCallback()* function from the demo calls an application-specific function called *BleApp_AttributeIndicated()* in which the OTAP Commands are handled.

```
static void BleApp_AttributeIndicated
(
    deviceId_t     deviceId,
    uint16_t       handle,
    uint8_t*       pValue,
    uint16_t       length
)
{
    if (handle == mPeerInformation.customInfo.otapServerConfig.hControlPoint)
    {
       otapCommandVars.pValueTemp = pValue;
        otapCommand_t*  pOtaCmd = otapCommandVars.otapCommandTemp;
        /* ... Missing code here ... */
        /* If the OTAP Server does not have internal storage then all commands must be forwarded
          *  via the serial interface. */
           FsciBleOtap_SendPkt (&(pOtaCmd->cmdId),
                       (uint8_t*)(&(pOtaCmd->cmd)),
                       length - gOtap_CmdIdFieldSize_c);
    }
    elseif (handle == otherHandle)
    {
        /* Handle other attribute indications here */
        /* ... Missing code here ... */
    }
    else
    {
        /*! A GATT Client is trying to GATT Indicate an unknown attribute value.
         * This should not happen. Disconnect the link. */
        Gap_Disconnect (deviceId);
    }
}
```

OTAP Server demo does not have internal storage, so all commands are forwarded via the serial interface.

To send OTAP Commands to the OTAP Client the application running the OTAP Server calls the *OtapServer_SendCommandToOtapClient()* function, which performs an ATT Write operation on the OTAP Control Point attribute.

```
static void OtapServer_SendCommandToOtapClient
        (deviceId_t  otapClientDevId,
         void*       pCommand,
         uint16_t    cmdLength)
```

(continues on next page)

(continued from previous page)

```
{
    /* GATT Characteristic to be written - OTAP Client Control Point */
    gattCharacteristic_t    otapCtrlPointChar;
    bleResult_t             bleResult;

    /* Only the value handle element of this structure is relevant for this operation. */
    otapCtrlPointChar.value.handle = mPeerInformation.customInfo.otapServerConfig.hControlPoint;
    otapCtrlPointChar.value.valueLength = 0;
    otapCtrlPointChar.cNumDescriptors = 0;
    otapCtrlPointChar.aDescriptors = NULL;

    bleResult = GattClient_SimpleCharacteristicWrite (mPeerInformation.deviceId,
                                                      &otapCtrlPointChar,
                                                      cmdLength,
                                                      pCommand);

    if (gBleSuccess_c == bleResult)
    {
        otapServerData.lastCmdSentToOtapClient = (otapCmdIdt_t)(((otapCommand_t*)pCommand)->
↪cmdId);
    }
    else
    {
        /*! A Bluetooth Low Energy error has occurred - Disconnect */
        (void)Gap_Disconnect (otapClientDevId);
    }
}
```

The ATT Confirmation for the ATT Write is received in the *BleApp_GattClientCallback()* set up earlier which receives a GATT procedure success message for a *gGattProcWriteCharacteristicValue_c* procedure type.

```
static void BleApp_GattClientCallback(
    deviceId_t              serverDeviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
   union
    {
        uint8_t                 errorTemp;
        attErrorCode_t              attErrorCodeTemp;
    }attErrorCodeVars;

    if (procedureResult == gGattProcError_c)
    {
        attErrorCodeVars.errorTemp = (uint8_t)error & 0xFFU;
        attErrorCode_t attError = attErrorCodeVars.attErrorCodeTemp;
        if (attError == gAttErrCodeInsufficientEncryption_c     ||
            attError == gAttErrCodeInsufficientAuthorization_c  ||
            attError == gAttErrCodeInsufficientAuthentication_c)
        {
    #if gAppUsePairing_d
            /* Start Pairing Procedure */
            (void)Gap_Pair (serverDeviceId, &gPairingParameters);
    #endif
        }

        BleApp_StateMachineHandler (serverDeviceId, mAppEvt_GattProcError_c);
```

(continues on next page)

(continued from previous page)

```
    }
    else if (procedureResult == gGattProcSuccess_c)
    {
        switch(procedureType)
        {
            /* ... Missing code here... */
            case gGattProcWriteCharacteristicValue_c:
            {
                BleApp_HandleValueWriteConfirmations (serverDeviceId);
            }
            break;

            default:
                ; /* For MISRA compliance */
            break;
        }

        BleApp_StateMachineHandler(serverDeviceId, mAppEvt_GattProcComplete_c);
    }
    else
    {
        ; /* For MISRA compliance */
    }
}
```

The *BleApp_HandleValueWriteConfirmations()* function deals with ATT Write Confirmations based on the requirements of the application.

There are two possible transfer methods for Image Chunks, the ATT transfer method and the L2CAP transfer method. The OTAP server is prepared to handle both, as requested by the OTAP Client.

To be able to use the L2CAP transfer method, the OTAP Server application must register a L2CAP LE PSM and 2 callbacks: a data callback and a control callback. This is done by using the *BluetoothLEHost_Initialized()* function.

```
/* Register OTAP L2CAP PSM */
  L2ca_RegisterLePsm (gOtap_L2capLePsm_c,
                      gOtapCmdImageChunkCocMaxLength_c); /*!< The negotiated MTU must be higher than␣
↪the biggest data chunk that is sent fragmented */
...
  App_RegisterLeCbCallbacks(BleApp_L2capPsmDataCallback, BleApp_L2capPsmControlCallback);
```

The data callback *BleApp_L2capPsmDataCallback()* is not used by the OTAP Server.

The control callback is used to handle L2CAP LE PSM connection requests from the OTAP Client and other events: PSM disconnections, No peer credits, and so on. The OTAP Client must initiate the L2CAP PSM connection if it wants to use the L2CAP transfer method.

```
static void BleApp_L2capPsmControlCallback(l2capControlMessage_t* pMessage)
{
    switch (pMessage->messageType)
    {
        case gL2ca_LePsmConnectRequest_c:
        {
            l2caLeCbConnectionRequest_t *pConnReq = &pMessage->messageData.connectionRequest;

            /* Respond to the peer L2CAP CB Connection request - send a connection response. */
            (void)L2ca_ConnectLePsm (gOtap_L2capLePsm_c,
                                     pConnReq->deviceId,
                                     mAppLeCbInitialCredits_c);
            break;
```

(continues on next page)

(continued from previous page)

```
        }
        case gL2ca_LePsmConnectionComplete_c:
        {
            l2caLeCbConnectionComplete_t *pConnComplete = &pMessage->messageData.connectionComplete;

            if (pConnComplete->result == gSuccessful_c)
            {
                /* Set the application L2CAP PSM Connection flag to TRUE because there is no gL2ca_
↪LePsmConnectionComplete_c
                 * event on the responder of the PSM connection. */
                otapServerData.l2capPsmConnected = TRUE;
                otapServerData.l2capPsmChannelId = pConnComplete->cId;

                if (pConnComplete->peerMtu > gOtap_l2capCmdMtuDataChunkOverhead_c)
                {
                    otapServerData.negotiatedMaxL2CapChunkSize = pConnComplete->peerMtu - gOtap_
↪l2capCmdMtuDataChunkOverhead_c;
                }
            }
            break;
        }
        case gL2ca_LePsmDisconnectNotification_c:
        {
            l2caLeCbDisconnection_t *pCbDisconnect = &pMessage->messageData.disconnection;

            /* Call App State Machine */
            BleApp_StateMachineHandler (pCbDisconnect->deviceId, mAppEvt_CbDisconnected_c);

            otapServerData.l2capPsmConnected = FALSE;
            break;
        }
        case gL2ca_NoPeerCredits_c:
        {
            l2caLeCbNoPeerCredits_t *pCbNoPeerCredits = &pMessage->messageData.noPeerCredits;
            (void)L2ca_SendLeCredit (pCbNoPeerCredits->deviceId,
                                     otapServerData.l2capPsmChannelId,
                                     mAppLeCbInitialCredits_c);
            break;
        }
        case gL2ca_Error_c:
        {
            /* Handle error */
            break;
        }
        default:
            ; /* For MISRA compliance */
            break;
    }
}
```

The ATT transfer method is supported by default but the L2CAP transfer method only works if the OTAP Client opens an L2CAP PSM credit-oriented channel.

To send data chunks to the OTAP Client the OTAP Server application calls the *OtapServer_SendCImgChunkToOtapClient()* function which delivers the chunk via the selected transfer method. For the ATT transfer method the chunk is sent via the *GattClient_CharacteristicWriteWithoutResponse()* function and for the L2CAP transfer method the chunk is sent via the *L2ca_SendLeCbData()* function.

```
static void OtapServer_SendCImgChunkToOtapClient(deviceId_t otapClientDevId,
                                                 void       pChunk,
```

(continues on next page)

(continued from previous page)

```
                                        uint16_t    chunkCmdLength)
{
    bleResult_t bleResult = gBleSuccess_c;
    if (otapServerData.transferMethod == gOtapTransferMethodAtt_c)
    {
        /* GATT Characteristic to be written without response - OTAP Client Data */
        gattCharacteristic_t otapDataChar;
        /* Only the value handle element of this structure is relevant for this operation. */
        otapDataChar.value.handle = mPeerInformation.customInfo.otapServerConfig.hData;
        bleResult = GattClient_CharacteristicWriteWithoutResponse
                                            (mPeerInformation.deviceId,
                                             &otapDataChar,
                                             chunkCmdLength,
                                             pChunk);
    }
    else if (otapServerData.transferMethod == gOtapTransferMethodL2capCoC_c)
    {
        bleResult = L2ca_SendLeCbData (mPeerInformation.deviceId,
                                       otapServerData.l2capPsmChannelId,
                                       pChunk,
                                       chunkCmdLength);
    }
    if (gBleSuccess_c != bleResult)
    {
        /*! A Bluetooth Low Energy error has occurred - Disconnect */
        Gap_Disconnect (otapClientDevId);
    }
}
```

The OTAP Server demo application relays all commands received from the OTAP Client to a PC through the FSCI type protocol running over a serial interface. It also directly relays all responses from the PC back to the OTAP Client.

Other implementations can bring the image to an external memory through other means of communication and directly respond to the OTAP Client requests.

Parent topic: Bluetooth Low Energy OTAP application integration

OTAP client An application running an OTAP Client must wait for an OTAP Server to connect and perform service and characteristic discovery before performing any OTAP-related operations. OTAP transactions can begin only after the OTAP Server writes the OTAP Control point CCC Descriptor to receive ATT Notifications. After this is done, bidirectional communication is established between the OTAP Server and Client and OTAP transactions can begin.

The OTAP Client can advertise the OTAP Service via the demo application. Optionally, the OTAP Server may already know the advertising device has an OTAP Service based on application-specific means. In both situations, the OTAP Server must discover the handles of the OTAP Service and its characteristics.

In addition to the OTAP Service instantiated in the GATT Database, the OTAP Client needs to have some storage capabilities for the downloaded image file.

How to put the OTAP Service in the GATT Database is described in The OTAP Service and Characteristics.

The upgrade image storage capabilities in the demo OTAP Client applications are handled by the *OtaSupport* module from the Framework, which contains support modules and drivers. The *OtaSupport* module has support for both internal storage (a part of the internal flash memory is reserved for storing the upgrade image) and external storage (a SPI flash memory chip).

The demo applications use internal storage by default. The internal storage is viable only if there is enough space in the internal flash for the upgrade image – the flash in this case should be at

least twice the size of the largest application. The *OtaSupport* module also needs the *Eeprom* module from the Framework to work correctly.

The OtaSupport module also includes functionality for configuring the OTACFG IFR sections after the image is received in order to enable the ROM bootloader to perform the actual image update.

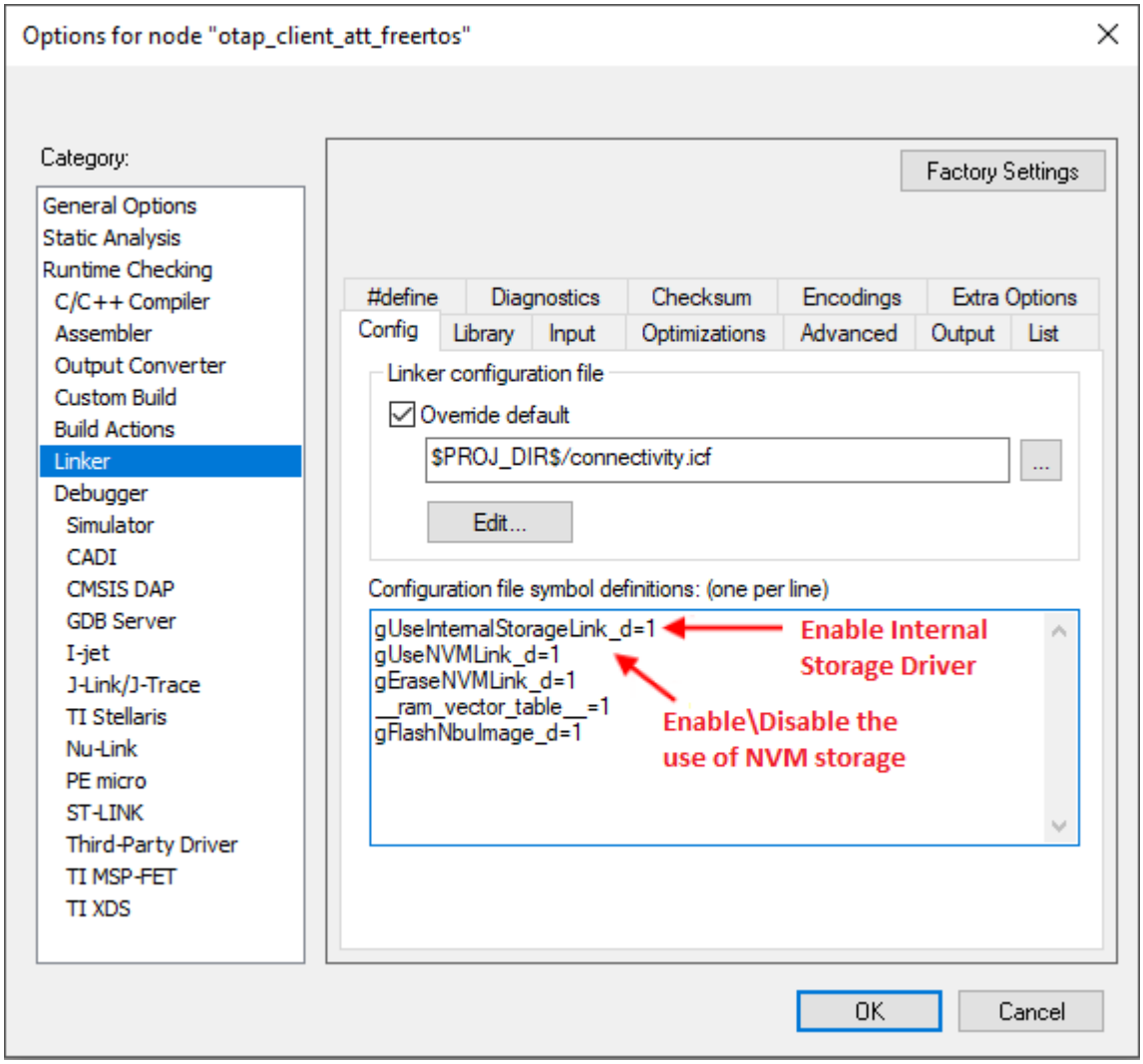
To use the *OtaSupport* module several configuration options must be set up in both the source files and the linker options of the toolchain.

To use internal storage, set up the gUseInternalStorageLink_d=1 symbol in the linker configuration window (**Linker->Config tab** in the IAR project properties) and set the gAppOtaExternalStorage_c value to (0) in the app_preinclude.hfile:

```
/*! Define as 1 to place OTA storage in external flash */
  #define gAppOtaExternalStorage_c (0)
```

The OTAP demo applications for the IAR EW IDE have some settings in the Linker options tab which must be configured to use OtaSupport and the OTAP Bootloader. In the **Project Target Options->Linker->Config** tab, 3 symbols must be correctly defined. To use NVM storage, thegUseNVMLink_d symbol must be set to 1. The gUseInternalStorageLink_d symbol must be set to 0 when OTAP external storage is used and to 1when the internal storage is used. The gEraseNVMLink_d must be set to 0.

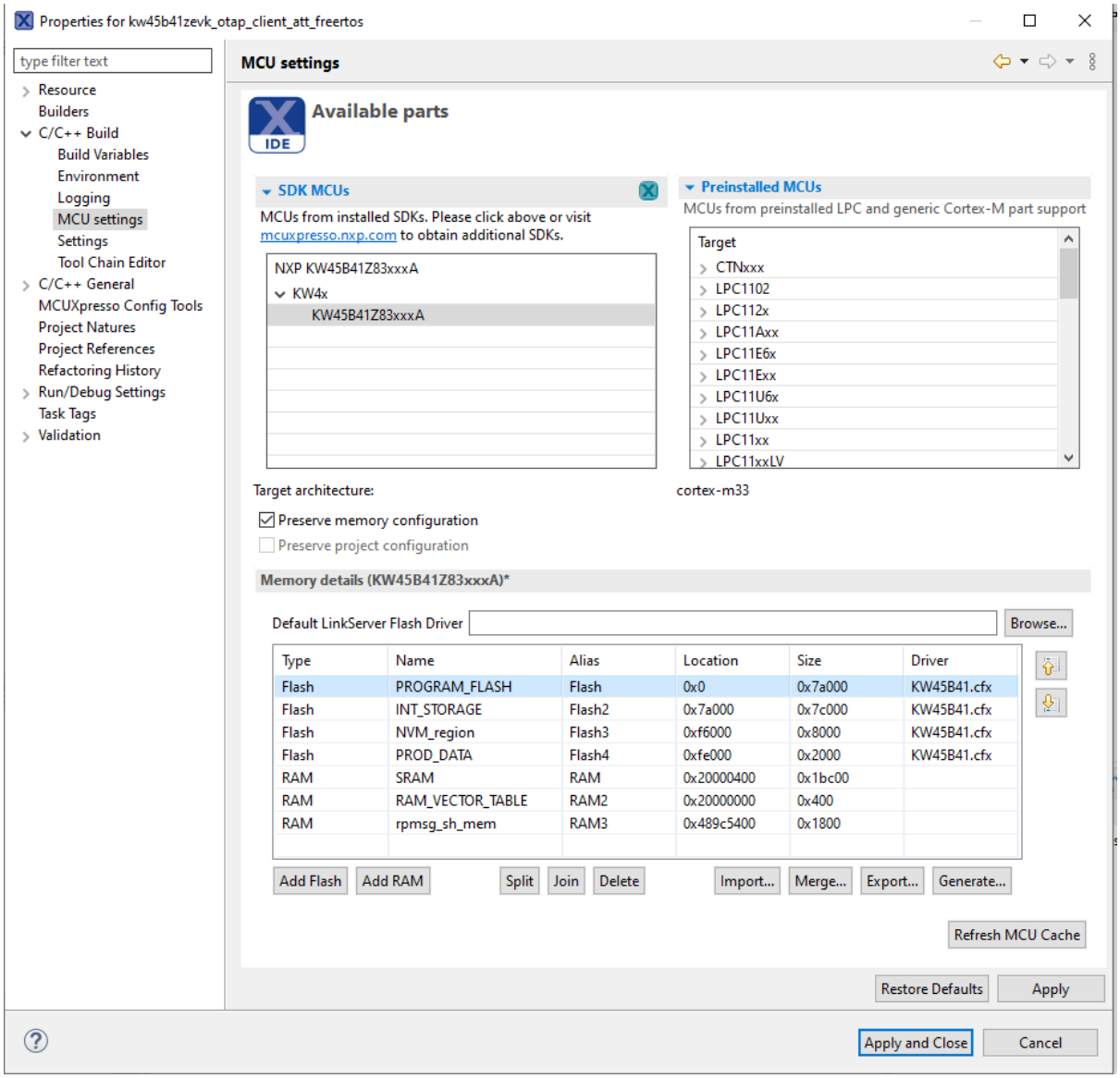
An example linker configuration window for IAR is shown below.



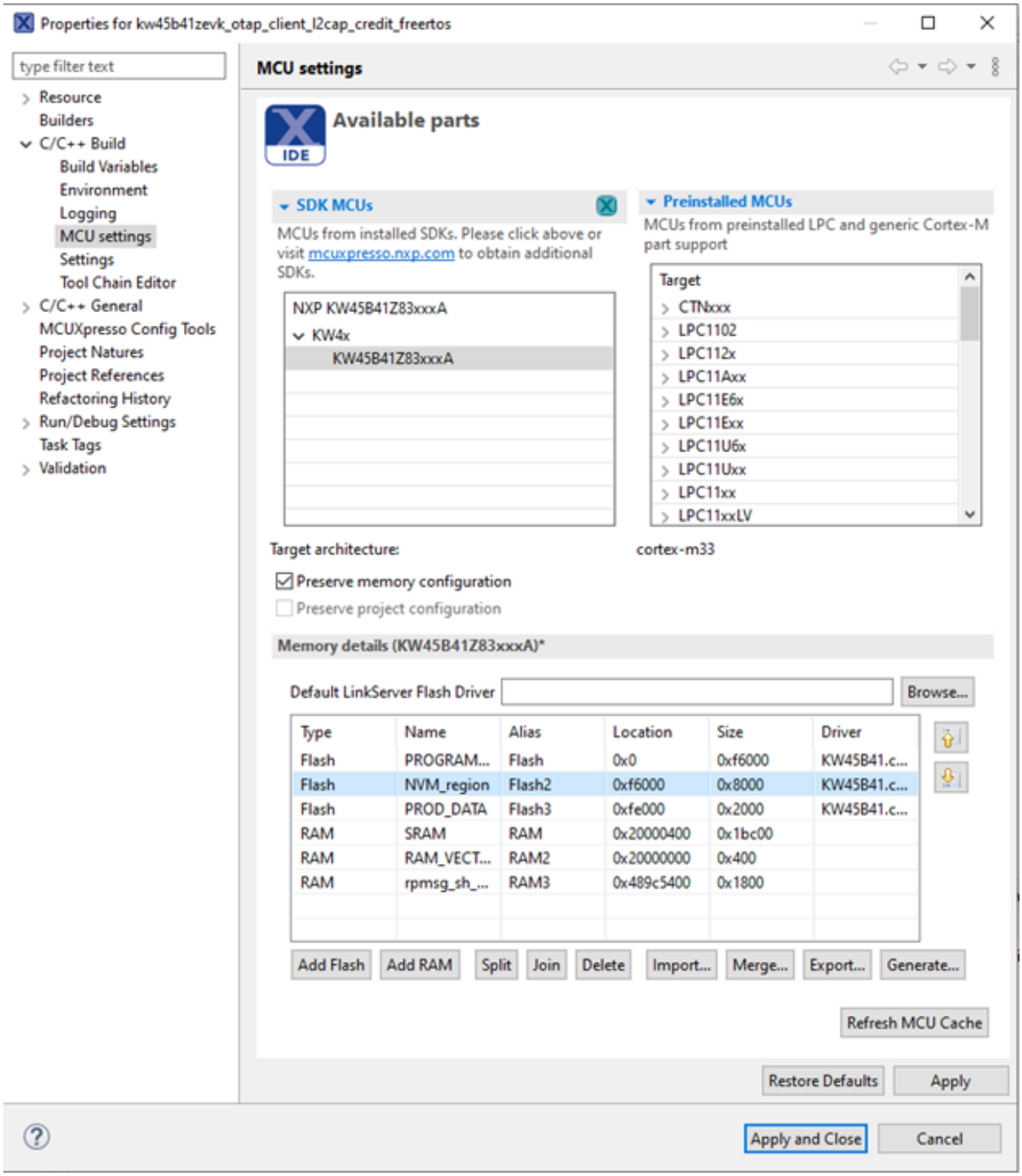
Note: The gEraseNVMLink_d=1 IAR linker flag places some dummy bytes into the NVM region to invalidate the data and force the application to erase the entire NVM region. When generating

an image for the OTA upgrade, this flag must be set to 0. This results in a smaller image size being transferred and lower power consumption. If the NVM region must be erased after the upgrade process, the “Preserve NVM” checkbox (from the Over The Air programming tool) should be unchecked.

For MCUXpresso IDE, the linker settings required for OTAP applications can be set up from the “**SDK Import Wizard**” or from the “**Project Properties** ->**MCU settings**”. Refer to Figure.



The demo applications use internal storage by default. To enable external storage support for MCUX, set the gAppOtaExternalStorage_c value to (1) in the app_preinclude.hfile. Also remove the INT_STORAGE section (from **Project Properties**-> **MCU settings**) and extend the PROGRAM_FLASHsection as shown in the Figure.



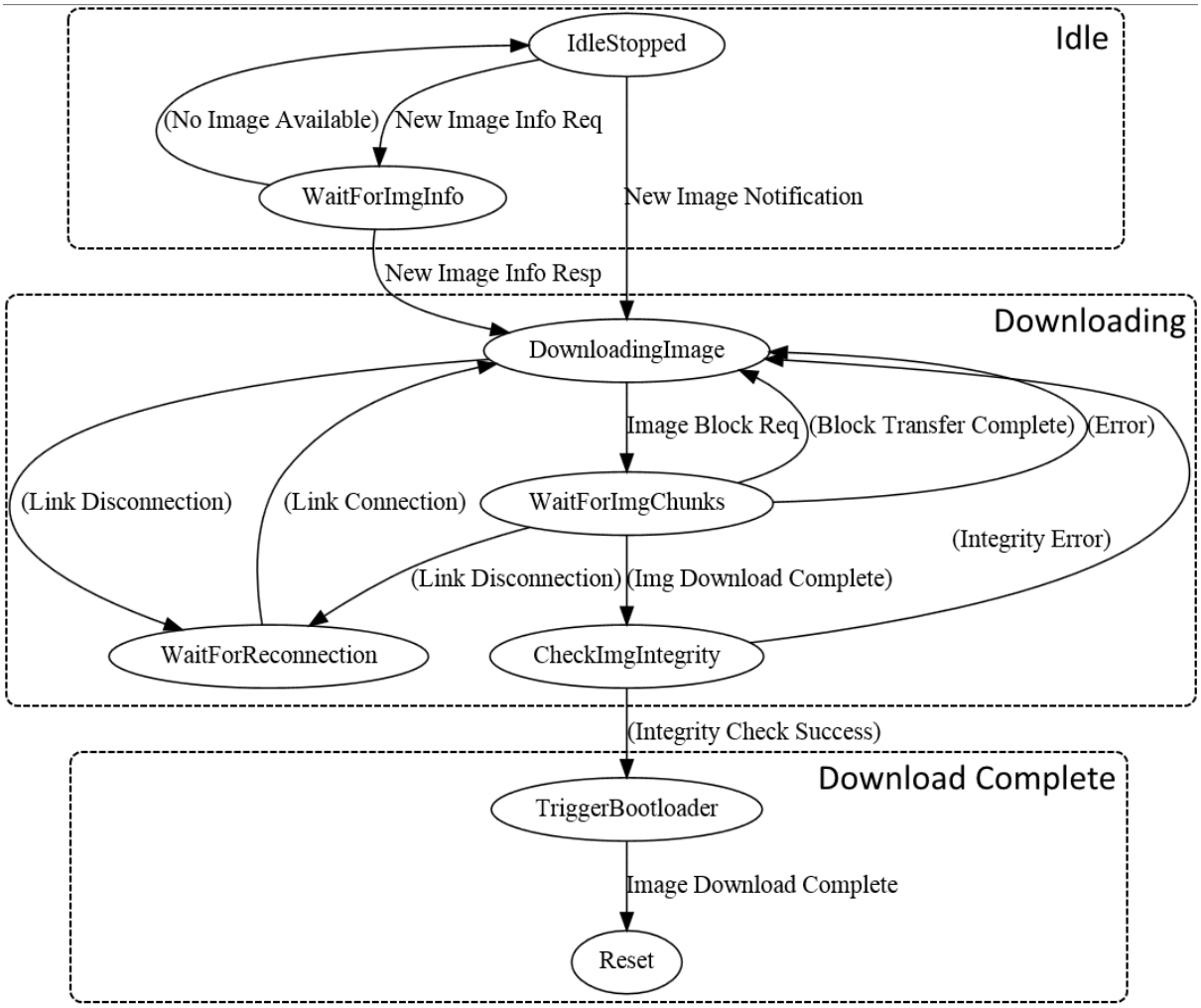
Once the application starts and bidirectional OTAP communication is established via the OTAP Service, then the OTAP Client must determine if the connected OTAP Server has a newer image than the one currently present on the device. This can be done in two ways:

- The OTAP Server knows by some application-specific means that it has a newer image and sends a New Image Notification to the OTAP Client or
- The OTAP Client sends a New Image Info Request to the OTAP Server and waits for a response. This example application uses the second method.

The New Image Info Request contains enough information about the currently running image to allow the OTAP Server to determine if it has a newer image for the requesting device. The New Image Info Response contains enough information for the OTAP Client to determine if the “dead-vertised” image is newer and it wants to download it. The best method is entirely dependent on application requirements.

An example function that checks if an *ImageVerison* field from a New Image Notification or a New Image Info Response corresponds to a newer image (based on the suggested format of this field) is provided in the OTAP Client demo applications. The function is called *OtapClient_IsRemoteImageNewer()*.

The OTAP Client application is a little more complicated than the OTAP Server application because more state information needs to be handled (current image position, current chunk sequence number, image file parsing information, and so on). An example state diagram for the OTAP Client is shown below. The Figure briefly lists the steps of the image download process. Note that some of the states may not be explicitly present in the demo applications.



After the OTAP Client determines that the peer OTAP Server has a suitable upgrade image available, it can start the download process. This is done by sending multiple Image Block Request messages and waiting for the Image Chunks via the selected transfer method.

While receiving the image file blocks, the OTAP Client application parses the image file. In case any parameter of an image file sub-element is invalid or the image file format is invalid, it sends an Error Notification to the OTAP Server and tries to restart the download process from the beginning or a known good position.

When an Image Chunk is received, its sequence number is checked and its content is parsed in the context of the image file format. If the sequence number is not as expected, then the block transfer is restarted from the last known good position. When all chunks of an Image Block are received, the next block is requested, if there are more blocks to download. When the last Image Block in an image file is received, then the image integrity is checked (the received CRC from the Image File CRC sub-element is compared to the computed CRC).

The computed image integrity initialization and intermediary value must be reset to '0' before

starting or restarting an image download. If the image integrity check fails then the image download process is restarted from the beginning. If the image integrity check is successful, then the Image Download Complete message is sent to the OTAP Server, the OTACFG IFR is updated and the MCU is restarted. After the restart, the ROM bootloader kicks in and writes the new image to the flash memory, afterwards giving CPU control to the newly installed application.

If at any time during the download process, a Link Layer disconnection occurs, then the image download process is restarted from the last known good position when the link is re-established.

As noted earlier, the OTAP Client application needs to handle a lot of state information. In the demo application, all this information is held in the *otapClientData* structure of the *otapClientAppData_t* type. The type is defined and the structure is initialized in the *otap_client.c* file of the application. This structure is defined and initialized differently for the OTAP Client ATT and L2CAP example applications. Mainly, the *transferMethod* member of the structure is constant and has different values for the two example applications and the L2CAP application structure has an extra member.

To receive write notifications when the OTAP Server writes the OTAP Control Point attribute and ATT Confirmations when it indicates the OTAP Control Point attribute, the OTAP Client application must register a GATT Server callback and enable write notifications for the OTAP Control Point attribute. This is done in the *BluetoothLEHost_Initialized()* function in the *otap_client_att.c/otap_client_l2cap_credit.c* file.

```
static void BluetoothLEHost_Initialized(void)
{
    /* ... Missing code here ... */

    /* Register stack callbacks */
    (void)App_RegisterGattServerCallback (BleApp_GattServerCallback);

    /* ... Missing code here ... */
}
```

The *BleApp_GattServerCallback()* function handles all incoming communication from the OTAP Server.

```
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent)
{
    switch (pServerEvent->eventType)
    {
        /* ... Missing code here ... */

        case gEvtCharacteristicCccdWritten_c:
        {
            OtapClient_CccdWritten (deviceId,
                                    pServerEvent->eventData.charCccdWrittenEvent.handle,
                                    pServerEvent->eventData.charCccdWrittenEvent.newCccd);
        }
        break;

       case gEvtAttributeWritten_c:
        {
            OtapClient_AttributeWritten (deviceId,
                                         pServerEvent->eventData.attributeWrittenEvent.handle,
                                         pServerEvent->eventData.attributeWrittenEvent.cValueLength,
                                         pServerEvent->eventData.attributeWrittenEvent.aValue);
        }
       break;

        case gEvtAttributeWrittenWithoutResponse_c:
        {
            OtapClient_AttributeWrittenWithoutResponse (deviceId,
```

(continues on next page)

(continued from previous page)

```
                                        pServerEvent->eventData.attributeWrittenEvent.handle,
                                        pServerEvent->eventData.attributeWrittenEvent.cValueLength,
                                        pServerEvent->eventData.attributeWrittenEvent.aValue);
        }
        break;

        case gEvtHandleValueConfirmation_c:
        {
            OtapClient_HandleValueConfirmation (deviceId);
        }
       break;*

        /* ... Missing code here ... */

        default:
            ; /* For MISRA compliance */
        break;
    }
}
```

When the OTAP Server Writes a CCCD the *BleApp_GattServerCallback()* function calls the *OtapClient_CccdWritten()* function which sends a New Image Info Request when the OTAP Control Point CCCD is written it – this is the starting point of OTAP transactions in the demo applications.

When an ATT Write Request is made by the OTAP Server the the *BleApp_GattServerCallback()* function calls the *OtapClient_AttributeWritten()* function which handles the data as an OTAP command. Only writes to the OTAP Control Point are handled as OTAP commands. For each command received from the OTAP Server there is a separate handler function which performs required OTAP operations. These are:

- *OtapClient_HandleNewImageNotification()*
- *OtapClient_HandleNewImageInfoResponse()*
- *OtapClient_HandleErrorNotification()*

When an ATT Write Command (GATT Write Without Response) is sent by the OTAP Server the *BleApp_GattServerCallback()* function calls the *OtapClient_AttributeWrittenWithoutResponse()* function which handles Data Chunks if the selected transfer method is ATT and returns an error if any problems are encountered. Data chunks are handled by the *OtapClient_HandleDataChunk()* function.

```
static void BleApp_AttributeWrittenWithoutResponse (deviceId_t deviceId,
                                                    uint16_t handle,
                                                    uint16_t length,
                                                    uint8_t* pValue)
{
    /* ... Missing code here ... */
    f (handle == value_otap_data)
    {
        /* ... Missing code here ... */
        if (otapClientData.transferMethod == gOtapTransferMethodAtt_c)
        {
            if (((otapCommand_t*)pValue)->cmdId == gOtapCmdIdImageChunk_c)
            {
                OtapClient_HandleDataChunk (deviceId,
                                            length,
                                            pValue);
            }
        }
        /* ... Missing code here ... */
    }
```

(continues on next page)

(continued from previous page)

```
    /* ... Missing code here ... */
}
```

Finally, when an ATT Confirmation is received for a previously sent ATT Indication the *BleApp_GattServerCallback()* function calls the *OtapClient_HandleValueConfirmation()* function, which performs the necessary OTAP operations based on the last sent command to the OTAP Server. This is done using separate confirmation handling functions for each command that is sent to the OTAP Server. These functions are:

- *OtapClient_HandleNewImageInfoRequestConfirmation()*
- *OtapClient_HandleImageBlockRequestConfirmation()*
- *OtapClient_HandleImageTransferCompleteConfirmation()*
- *OtapClient_HandleErrorNotificationConfirmation()*
- *OtapClient_HandleStopImageTransferConfirmation()*

Outgoing communication from the OTAP Client to the OTAP Server is done using the *OtapCS_SendCommandToOtapServer()* function. This function writes the value to be indicated to the OTAP Control Point attribute in the GATT database and then calls the *OtapCS_SendControlPointIndication()*which checks if indications are enabled for the target device and sends the actual ATT Indication. Both functions are implemented in the *otap_service.c* file.

```
bleResult_t OtapCS_SendCommandToOtapServer (uint16_t serviceHandle,
                                            void* pCommand,
                                            uint16_t cmdLength)
{
    union
    {
        uint8_t*              uuid_char_otap_control_pointTemp;
        bleUuid_t*             bleUuidTemp;
    }bleUuidVars;

    uint16_t  handle;
    bleResult_t result;
    bleUuidVars.uuid_char_otap_control_pointTemp = uuid_char_otap_control_point;
    bleUuid_t* pUuid = bleUuidVars.bleUuidTemp;

    /* Get handle of OTAP Control Point characteristic */
    result = GattDb_FindCharValueHandleInService(serviceHandle,
                                                 gBleUuidType128_c, pUuid, &handle);

    if (result == gBleSuccess_c)
    {
        /* Write characteristic value */
        result = GattDb_WriteAttribute(handle,
                                       cmdLength,
                                       (uint8_t*)pCommand);

       if (result == gBleSuccess_c)
        {
            /* Send Command to the OTAP Server via ATT Indication */
            result = OtapCS_SendControlPointIndication (handle);
        }
    }

   return result;
}
```

(continues on next page)

(continued from previous page)

```
static bleResult_t OtapCS_SendControlPointIndication (uint16_t handle)
{
    uint16_t     hCccd;
    bool_t       isIndicationActive;
    /* Get handle of CCCD */
    GattDb_FindCccdHandleForCharValueHandle (handle, &hCccd);
    Gap_CheckIndicationStatus (...);
    return GattServer_SendIndication (...);
}
```

The *otap_interface.h* file contains all the necessary information for parsing and building OTAP commands (packed command structures type definitions, command parameters enumerations, and so on).

For the two possible image transfer methods (ATT and L2CAP) there are two separate demo applications. To be able to use the L2CAP transfer method the OATP Client application must register a L2CAP LE PSM and 2 callbacks: a data callback and a control callback. This is done in the *OtapClient_Config()* function.

```
/* Register OTAP L2CAP PSM */
L2ca_RegisterLePsm (gOtap_L2capLePsm_c,
gOtapCmdImageChunkCocMaxLength_c); /*!< The negotiated MTU must be higher than the biggest data␣
↪chunk that is sent fragmented */
...
App_RegisterLeCbCallbacks(BleApp_L2capPsmDataCallback, BleApp_L2capPsmControlCallback);
```

The control callback is used to handle L2CAP LE PSM-related events: PSM disconnections, PSM Connection Complete, No peer credits, and so on.

```
static void BleApp_L2capPsmControlCallback(l2capControlMessage_t* pMessage)
{
    switch (pMessage->messageType)
    {
        case gL2ca_LePsmConnectRequest_c:
        {
            l2caLeCbConnectionRequest_t *pConnReq = &pMessage->messageData.connectionRequest;

            /* This message is unexpected on the OTAP Client, the OTAP Client sends L2CAP PSM connection
             * requests and expects L2CAP PSM connection responses.
             * Disconnect the peer. */
            (void)Gap_Disconnect (pConnReq->deviceId);

            break;
        }
        case gL2ca_LePsmConnectionComplete_c:
        {
            l2caLeCbConnectionComplete_t *pConnComplete = &pMessage->messageData.connectionComplete;

            /* Call the application PSM connection complete handler. */
            OtapClient_HandlePsmConnectionComplete (pConnComplete);

            break;
        }
        case gL2ca_LePsmDisconnectNotification_c:
        {
            l2caLeCbDisconnection_t *pCbDisconnect = &pMessage->messageData.disconnection;

            /* Call the application PSM disconnection handler. */
            OtapClient_HandlePsmDisconnection (pCbDisconnect);

            break;
```

(continues on next page)

(continued from previous page)

```
        }
        case gL2ca_NoPeerCredits_c:
        {
            l2caLeCbNoPeerCredits_t *pCbNoPeerCredits = &pMessage->messageData.noPeerCredits;
            (void)L2ca_SendLeCredit (pCbNoPeerCredits->deviceId,
                                     otapClientData.l2capPsmChannelId,
                                     mAppLeCbInitialCredits_c);
            break;
        }
        case gL2ca_Error_c:
        {
            /* Handle error */
            break;
        }
        default:
            ; /* For MISRA compliance */
            break;
    }
}
```

The OTAP Client must initiate the L2CAP PSM connection if it wants to use the L2CAP transfer method; this can be done using the *L2ca_ConnectLePsm()* function. The *L2ca_ConnectLePsm()* function is called by the *OtapClient_ContinueImageDownload()* if the transfer method is L2CAP and the PSM is found to be disconnected.

```
static void OtapClient_ContinueImageDownload (deviceId_t deviceId)
{
    /* ... Missing code here ... */
    /* Check if the L2CAP OTAP PSM is connected and if not try to connect and exit immediately. */
    if ((otapClientData.l2capPsmConnected == FALSE) &&
            (otapClientData.state != mOtapClientStateImageDownloadComplete_c))
    {
        L2ca_ConnectLePsm (gOtap_L2capLePsm_c,
                           deviceId,
                           mAppLeCbInitialCredits_c);
        bValidState = FALSE;;
    }
    /* ... Missing code here ... */
}
```

The PSM data callback *BleApp_L2capPsmDataCallback()* is used by the OTAP Client to handle incoming image file parts from the OTAP Server.

```
static void BleApp_L2capPsmDataCallback (deviceId_t     deviceId,
                                         uint16_t       lePsm,
                                         uint8_t*       pPacket,
                                         uint16_t       packetLength)
{
    OtapClient_HandleDataChunk (deviceId,
                                packetLength,
                                pPacket);
}
```

All data chunks regardless of their source (ATT or L2CAP) are handled by the *OtapClient_HandleDataChunk()* function. This function checks the validity of Image Chunk messages, parses the image file, requests the continuation or restart of the image download and triggers the bootloader when the image download is complete.

```
static void OtapClient_HandleDataChunk (deviceId_t deviceId, uint16_t length, uint8_t* pData);
```

The Image File CRC Value is computed on the fly as the image chunks are received using

the *OTA_CrcCompute()* function from the *OtaSupport* module which is called by the *OtapClient_HandleDataChunk()* function. The *OTA_CrcCompute()* function has a parameter for the intermediary CRC value which must be initialized to 0 every time a new image download is started.

The actual write of the received image parts to the storage medium is also done in the *OtapClient_HandleDataChunk()* function using the *OtaSupport* module. This is achieved using the following functions:

- *OTA_StartImage()* – called before the start of writing a new image to the storage medium.
- *OTA_CancelImage()* – called whenever an error occurs and the image download process needs to be stopped/restarted from the beginning.
- *OTA_PushImageChunk()* – called to write a received image chunk to the storage medium. Note that only the Upgrade Image Sub-element of the image file is actually written to the storage medium.
- *OTA_CommitImage()* - called to set up what parts of the downloaded image are written to flash and other information for the bootloader. The Value field of the Sector Bitmap Sub-element of the Image File is given as a parameter to this function.
- *OTA_SetNewImageFlag()* - called to configure the OTACFG IFR when a new image has been successfully received. When the MCU is reset, the ROM bootloader transfers the new image from the storage medium to the program flash.

To continue the image download process after a block is transferred or to restart it after an error has occurred the *OtapClient_ContinueImageDownload()* function is called. This function is used in multiple situations during the image download process.

To summarize, an outline of the steps required to perform the image download process is shown below:

- Wait for a connection from an OTAP Server
- Wait for the OTAP Server to write the OTAP Control Point CCCD
- Ask or wait for image information from the server
- If a new image is available on the server, start the download process using the *OtapClient_ContinueImageDownload()* function.
  - If the transfer method is L2CAP CoC, then initiate a PSM connection to the OTAP Server
- Repeat while image download is not complete.
  - Wait for image chunks.
  - *Call the OtapClient_HandleDataChunk()* function for all received image chunks regardless of the selected transfer method.
    - Check image file header integrity using the *OtapClient_IsImageFileHeaderValid()* function.
    - Write the Upgrade Image Sub-element to the storage medium using *OtaSupport* module functions.
    - When the download is complete, check image integrity.
      - If the integrity check is successful, commit the image using the Sector Bitmap Sub-element and trigger the bootloader
      - If integrity check fails, restart the image download from the beginning
    - If the download is not complete, ask for a new image chunk.
  - If any error occurs during the processing of the image chunk, restart the download from the last known good position.

- If an image was successfully downloaded and transferred to the storage medium and the bootloader triggered, then reset the MCU to start the flashing process of the new image.

Parent topic: Bluetooth Low Energy OTAP application integration

Parent topic:*Over the Air Programming (OTAP)*

Secured OTAP The security features of the KW45/K32W1 devices enable them to use secured OTAP, meaning the new images can be authenticated and encrypted. The decryption/authentication keys are programmed into hardware fuses. For information on how to prepare the board, refer to the accompanying document related to board provisioning. For information on how to obtain the secured image, refer to the *Bluetooth Low Energy Demo Applications User's Guide (BLEDAUG)*.

Parent topic:*Over the Air Programming (OTAP)*

Creating a Bluetooth LE application when the Host Stack runs on another processor This section describes how to create a Bluetooth Low Energy application (host), when the Bluetooth Low Energy Host Stack is running on another processor (blackbox). The section alsp provides sample code to explain how to achieve this.

The supported serial interfaces between the two chips (application and the Bluetooth Low Energy Host Stack) are UART, SPI, or USB.

Typical applications employing Bluetooth LE Host Stack blackboxes are host systems such as a PC tool or an embedded system that has an application implementation. This chapter describes an embedded application.

For more information, refer to *Bluetooth Low Energy Host Stack FSCI Reference Manual*. This document provides explicit information on exercising the Bluetooth Low Energy Host Stack functionality through a serial communication interface to a host system.

Serial manager and FSCI configuration For creating an embedded application that communicates with the Bluetooth Low Energy Host Stack using the serial interface, the following steps must be done:

Serial manager initialization The function that must be called for Serial Manager initialization is located in *SerialManager.h*:

```
/* Init serial manager */
SerialManager_Init();
```

Parent topic: Serial manager and FSCI configuration

FSCI configuration and initialization By default, the FSCI module is disabled. It must be enabled by setting gFsciIncluded_c to 1. Also, gFsciLenHas2Bytes_c must be set to 1 because Bluetooth Low Energy Host Stack interface commands and events need serial packets bigger than 255 octets.

For more information on the following configuration parameters, refer to the FSCI chapter of the *Connectivity Framework Reference Manual*.

To configure the FSCI module, the following parameters can be set on both the Bluetooth Low Energy Application project and the Bluetooth Low Energy FSCI blackbox:

```
/* Enable/Disable FSCI */
#define gFsciIncluded_c                 1

/* Enable/Disable FSCI Low Power Commands*/
#define gFSCI_IncludeLpmCommands_c      0

/* Defines FSCI length - set this to FALSE is FSCI length has 1 byte */
#define gFsciLenHas2Bytes_c             1

/* Defines FSCI maximum payload length */
#define gFsciMaxPayloadLen_c            1660

/* Enable/Disable Ack transmission */
#define gFsciTxAck_c                    0

/* Enable/Disable Ack reception */
#define gFsciRxAck_c                    0

/* Enable FSCI Rx restart with timeout */
#define gFsciRxTimeout_c                1
#define mFsciRxTimeoutUsePolling_c      1

/* Use Misra Compliant version of FSCI module */
#define gFsciUseDedicatedTask_c         1

/* FSCI task size */
#if defined(DEBUG)
#define gFsciTaskStackSize_c            4600
#else
#define gFsciTaskStackSize_c            2600
#endif
```

To perform the FSCI module initialization, the following code can be used:

```
/* Define fsci serial manager handle */
  #if defined(gFsciIncluded_c) && (gFsciIncluded_c > 0)
    extern serial_handle_t g_fsciHandleList[gFsciIncluded_c];
  #endif /*gFsciIncluded_c > 0*/

void BluetoothLEHost_AppInit(void)
{
    /* Init FSCI */
    FSCI_commInit( g_fsciHandleList );

    /* Register BLE handlers in FSCI */
    fsciBleRegister(0);
    ...
}
```

Parent topic: Serial manager and FSCI configuration

FSCI handlers (GAP, GATT, and GATTDB) registration For receiving messages from all the Bluetooth Low Energy Host Stack serial interfacing layers (GAP, GATT, and GATTDB), a function handler must be registered in FSCI for each layer:

```
fsciBleRegister(0);
```

Parent topic: Serial manager and FSCI configuration

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

Bluetooth Low Energy Host Stack initialization The Bluetooth Low Energy Host Stack must be initialized when platform setup is complete and all RTOS tasks have been started. This initialization is done by restarting the blackbox using a FSCI CPU Reset Request command. This is performed automatically by the ***Ble_Initialize(App_GenericCallback)*** function.

```
/* Send FSCI CPU reset command to BlackBox */
FSCI_transmitPayload(gFSCI_ReqOpcodeGroup_c, mFsciMsgResetCPUReq_c, NULL, 0, fsciInterface);
```

The completion of the Bluetooth Low Energy Host Stack initialization is signaled by the reception of the *GAP-GenericEventInitializationComplete.Indication* event (over the serial communication interface, in FSCI). The *Bluetooth Low Energy-HostInitialize.Request* command is not required to be sent to the blackbox (the entire initialization is performed by the blackbox, when it resets).

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

GATT database configuration The GATT database always resides on the same processor as the entire Bluetooth Low Energy Host Stack, so the attributes must be added by the host application using the serial communication interface.

To create a GATT database remotely, *GATTDBDynamic* commands must be used. The GATTDBDynamic API is provided to the user that performs all the required memory allocations and sends the FSCI commands to the blackbox. The result of the operation is returned, including optionally the service, characteristic, and 'cccd' handles returned by the blackbox.

Current supported API for adding services is the following:

```
bleResult_t GattDbDynamic_AddGattService (gattServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddGapService (gapServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddIpssService (ipssServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddHeartRateService (heartRateServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddBatteryService (batteryServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddDeviceInformationService (deviceInfoServiceHandles_t*␣
↪pOutServiceHandles);
```

The service handles are optional.

Also, a generic function is provided, so that the user can add any generic service to the database:

```
bleResult_t GattDbDynamic_AddServiceInDatabase (serviceInfo_t* pServiceInfo);
```

Usually, a Bluetooth Low Energy Application is ported from a single chip solution, where the Bluetooth Low Energy Application and the Bluetooth Low Energy stack reside on the same processor and the GATT database is populated statically. The user should remove all the attribute handles from any structure and replace them with *gGattDbInvalidHandle_d*. The attribute handles should be populated after the services are added dynamically to the database with the handles returned by the previous API.

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

FSCI host layer The Bluetooth Low Energy GAP, GATT, GATTDB, and L2CAP APIs are included in the Bluetooth Low Energy interface. When these APIs reside on a separate processor than the Bluetooth Low Energy stack, they are implemented as an FSCI Host Layer that should be added to the Bluetooth Low Energy Application project.

This layer is responsible for serializing API to the corresponding FSCI commands. The layer also sends these APIs to the blackbox, receives and deserializes FSCI statuses and events, presents them to the Bluetooth Low Energy Application, and arbitrates access from multiple tasks to the serial interface.

All the GAP, GATT, GATTDB, and L2CAP APIs are executed asynchronously, so the user context blocks waiting for the response from the blackbox. The response can be the status of the request or optionally an FSCI event, which includes the output parameters of a synchronous function.

There are also functions without parameters that are not executed synchronously and they are provided asynchronously through a later FSCI event. It is the responsibility of the FSCI Host layer to keep the application-allocated memory between the time of the request and the completion of the event with the actual values of the output parameters and populate them accordingly.

The Bluetooth Low Energy API execution inside the FSCI Host layer first waits for gaining access to the serial interface through a mutex. Once the access is gained, the FSCI request is sent to the serial interface to the blackbox. Then, by default, the serial interface response is received by polling until the whole FSCI packet is received. The other option available is to block the user task to wait for an OS event that is set by the FSCI module when the status is received. For more information on the FSCI module, see the Connectivity Framework Reference Manual. See *References*.

The API can have output parameters that are to be received immediately after the status of the request. In such as case, if the status of the request is 'success', the polling mechanism continues to receive the whole FSCI packet of the Bluetooth Low Energy event. The output parameters are obtained and the values are filled in the memory space provided by the application. After obtaining the status and optionally the event, the execution of the request is considered completed, the mutex to the serial interface is unlocked, and the execution flow is returned to the user calling context.

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

References For more information, refer to the following documents:

- Bluetooth Low Energy Demo Applications User's Guide (*KW45_K32W1_BLEDAUG*)
- Bluetooth Low Energy Host Stack API Reference Manual (*KW45_K32W1_BLEHSAPIRM*)
- Bluetooth Low Energy Host Stack FSCI (Framework Serial Connectivity Interface) API Reference Manual (*KW45_K32W1_BLEHSFSCIAPIRM*)
- Connectivity Framework Reference Manual (*KW45_K32W1_CONNFWRM*)
- Bluetooth Low Energy CCC Digital Key R3 Application Note (*AN12791*)

Acronyms and abbreviations The following acronyms are used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
CCCD	Client Characteristic Configuration Descriptor
CSRK	Connection Signature Resolving Key
ELKE	EdgeLock Secure Enclave
FSCI	Framework Serial Connectivity Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GATTDB	Generic Attribute Profile Database
HCI	Host Controller Interface
IRK	Identity Resolving Key
LTK	Long Term Key
LL	Link Layer
L2CAP	Logical Link Control and Adaptation Protocol
MTU	Maximum Transmission Unit
PDU	Protocol Data Unit
PAwR	Periodic Advertising with Responses
RPA	Resolvable Private Address
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System
RX	Receiver
SDK	Software Development Kit
TX	Transmitter
WFI	Wait For Interrupt

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2022-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision history This table summarizes revisions to this document.

Document ID	Release date	Description
UG10184 v.1.0	26 November 2024	Document is aligned to KW47 EAR 2.1 24.12.00-pvw2 release

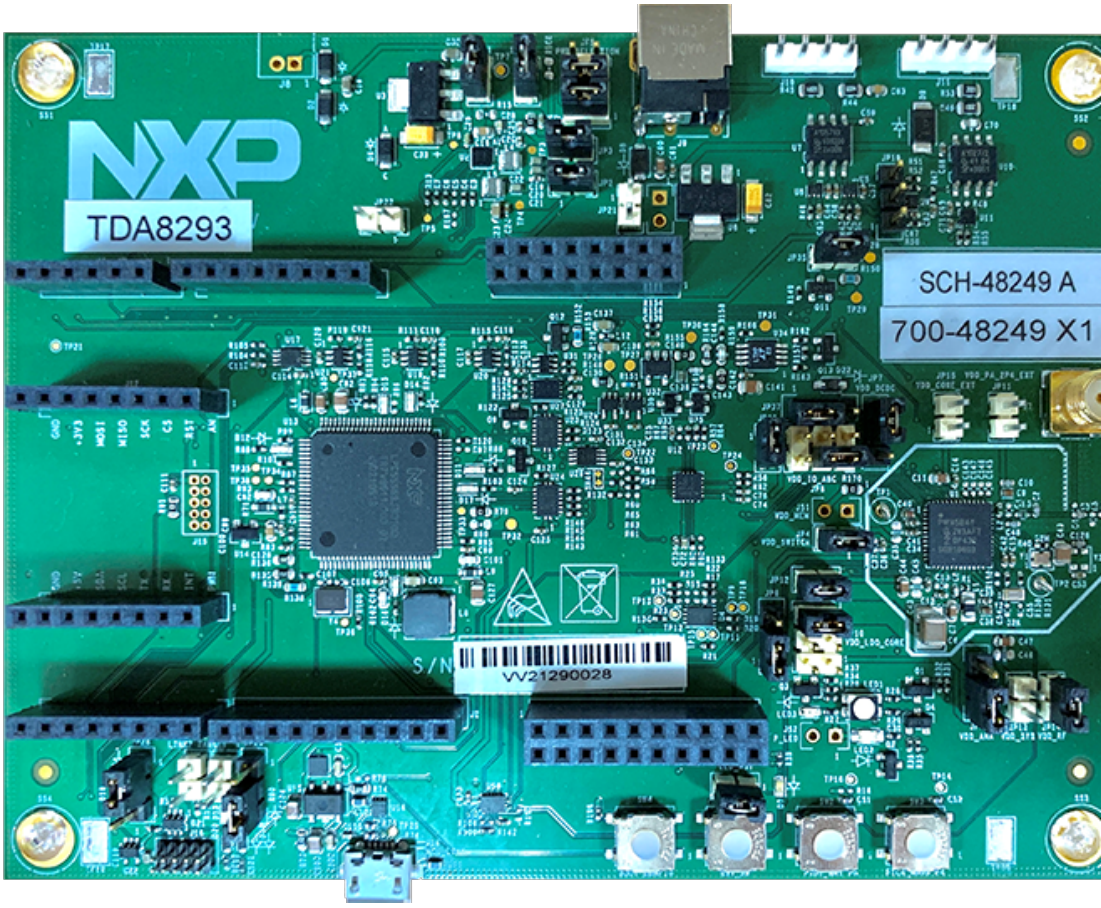
Legal information

KW45, K32W1, MCXW71, KW47, MCXW72 Bluetooth Low Energy Software Quick Start Guide

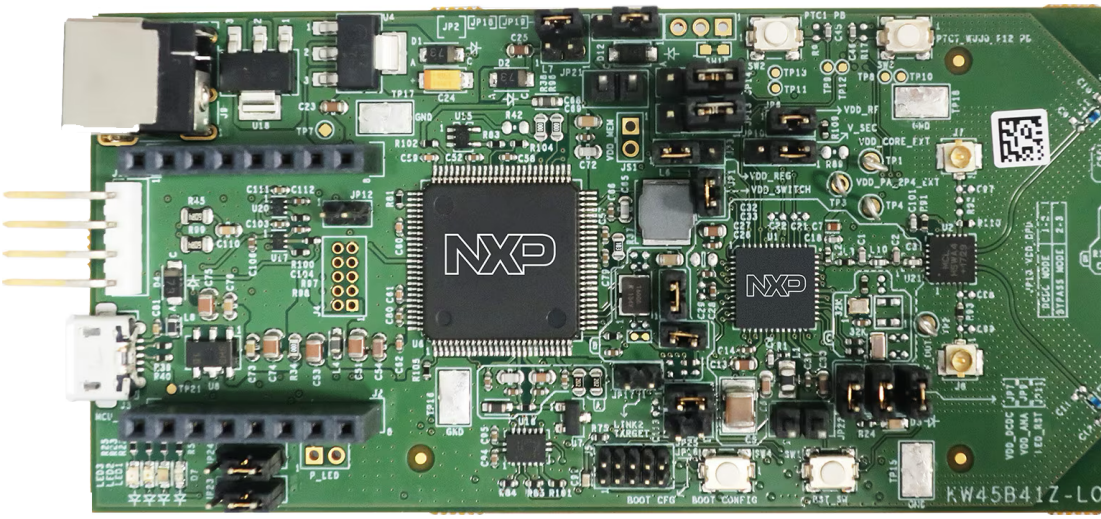
Introduction This document briefly describes the process of using NXP Bluetooth Low Energy Software for the KW45, K32W1, MCXW71, KW47, or MCXW72 wireless microcontroller platforms (version 1.1.0). It lists the hardware setup and steps for building and usage of the provided demo applications.

Hardware setup The examples described in this document use a KW45B41Z-EVK, KW45B41Z-LOC, K32W148-EVK, MCX-W71-EVK, FRDM-MCXW71, KW47-EVK, KW47-LOC, MCX-W72-EVK, or FRDM-MCXW72 as the development platform, as shown in the figures below.

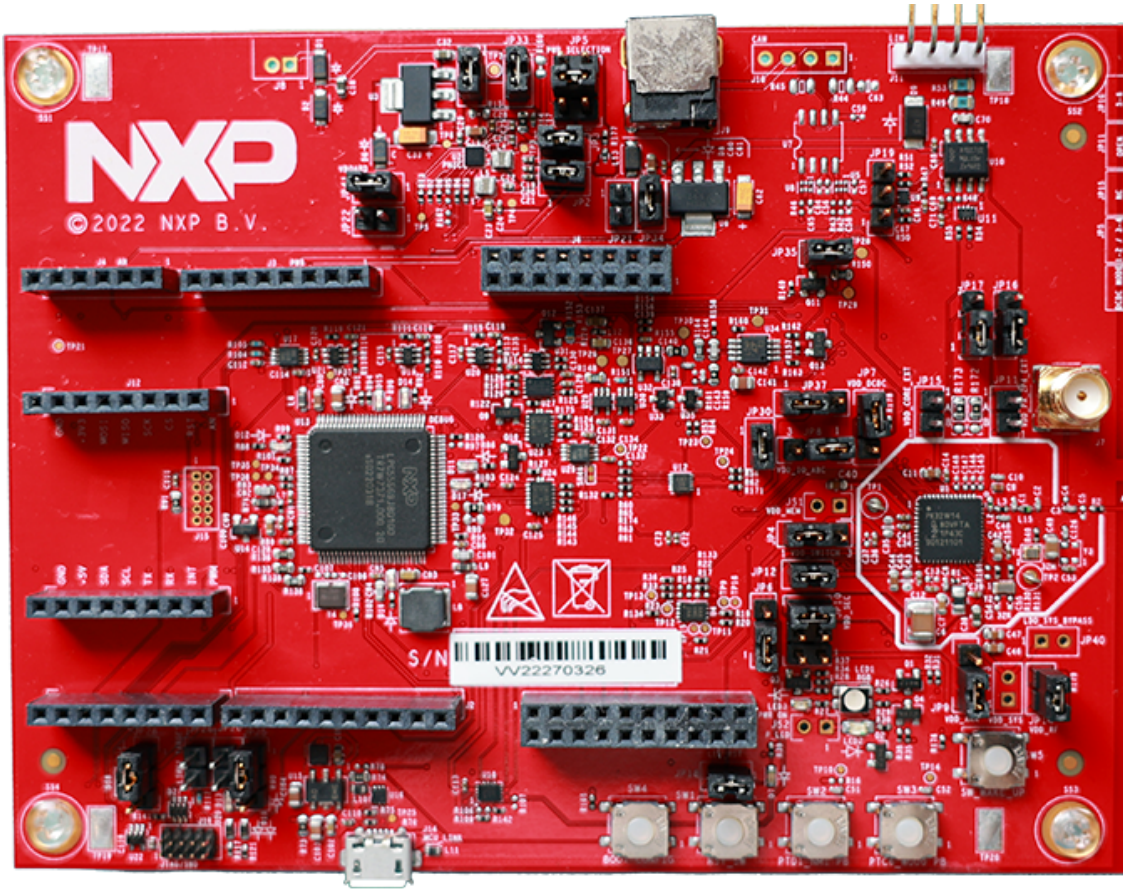
- The default interface selected in the IAR Embedded Workbench for Arm projects included in this release is below:
  - CMSIS-DAP for kw45b41zevk, kw45b41zloc, k32w148evk, mcxw71evk, frdmmcxw71
  - JLink for kw47evk, kw47loc, mcxw72evk, and frdmmcxw72 platforms
- Use jumpers to configure the boards in one of the available power configurations (refer to the specific board documentation).
- On all boards, the OpenSDA USB port is connected to a Windows PC. The OpenSDA chip on the board requires flashing with appropriate firmware with debugging and virtual serial COM port capabilities.



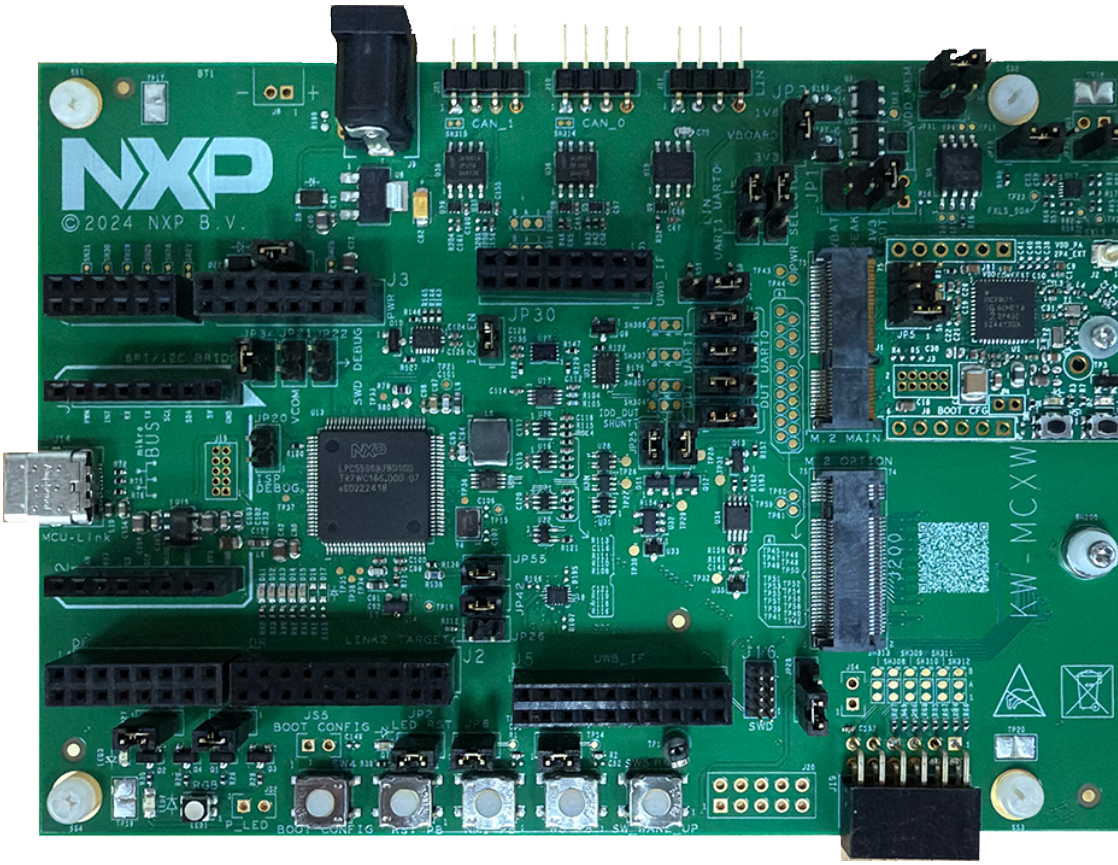
KW45B41Z-EVK board



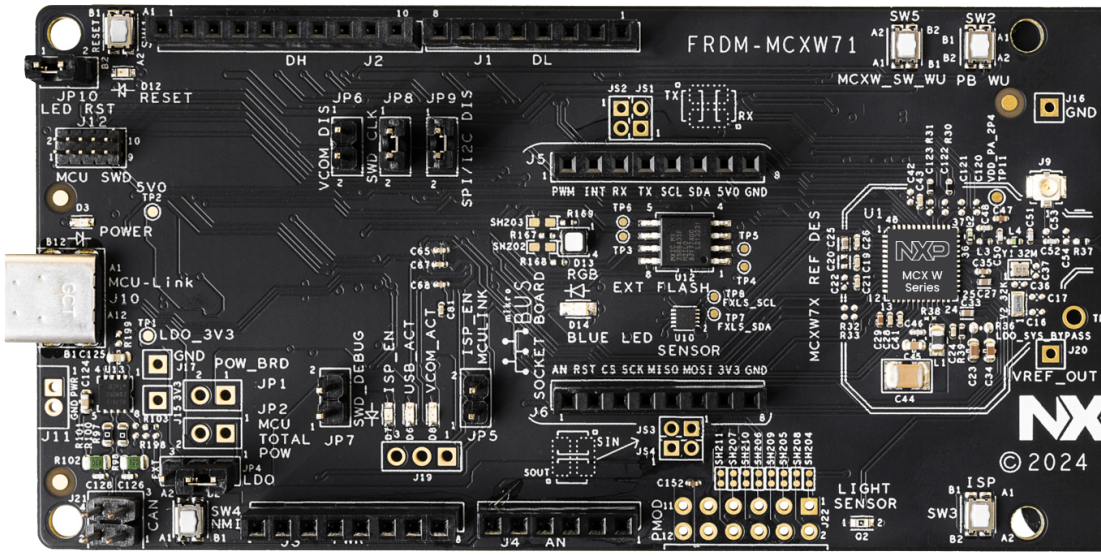
KW45B41Z-LOC board



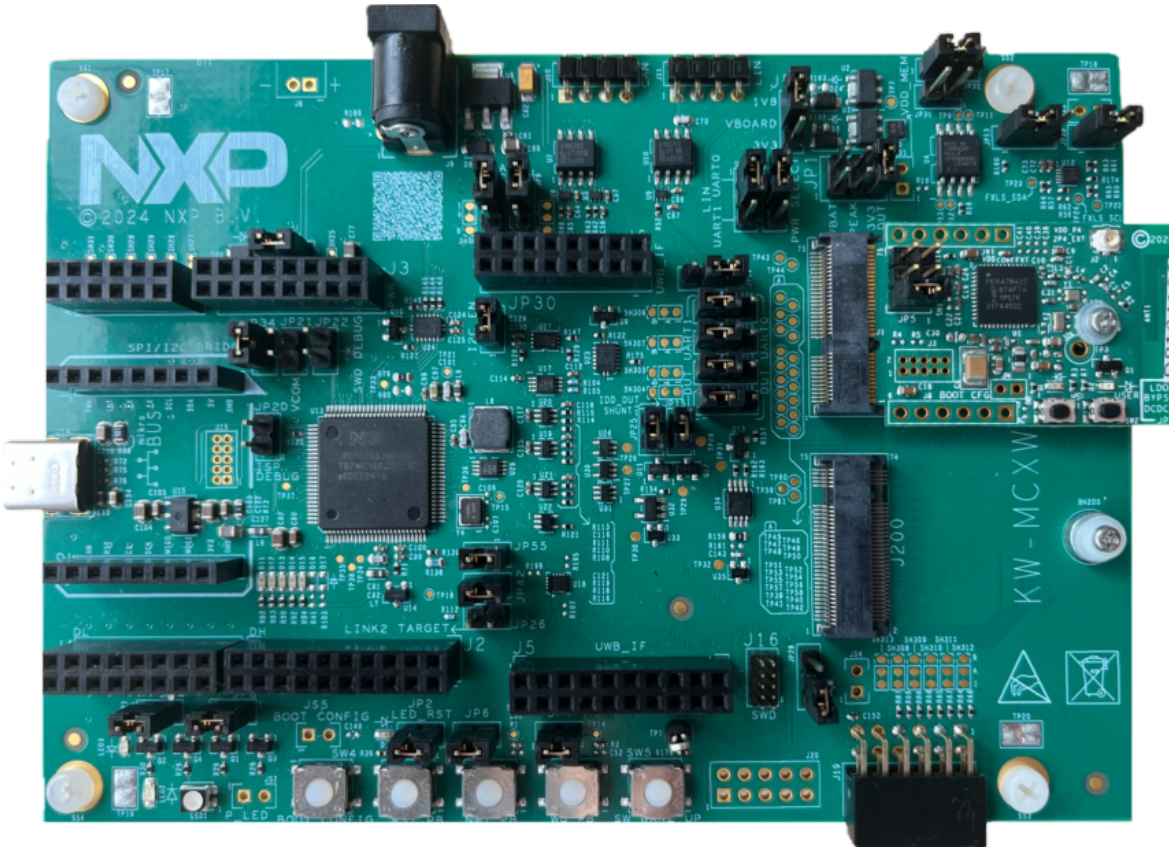
K32W148-EVK board



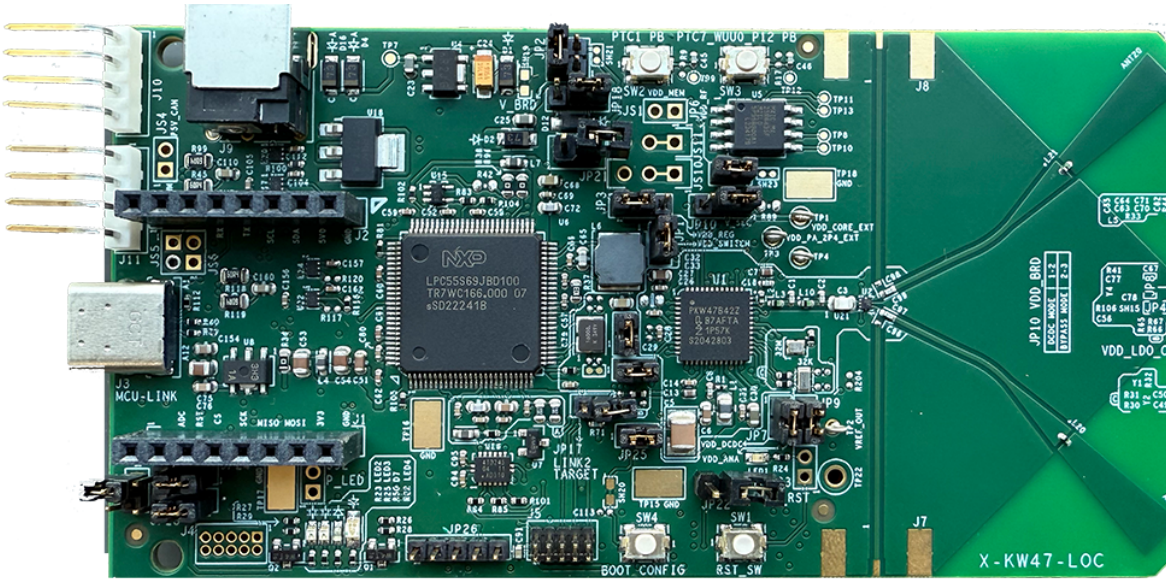
MCX-W71-EVK board



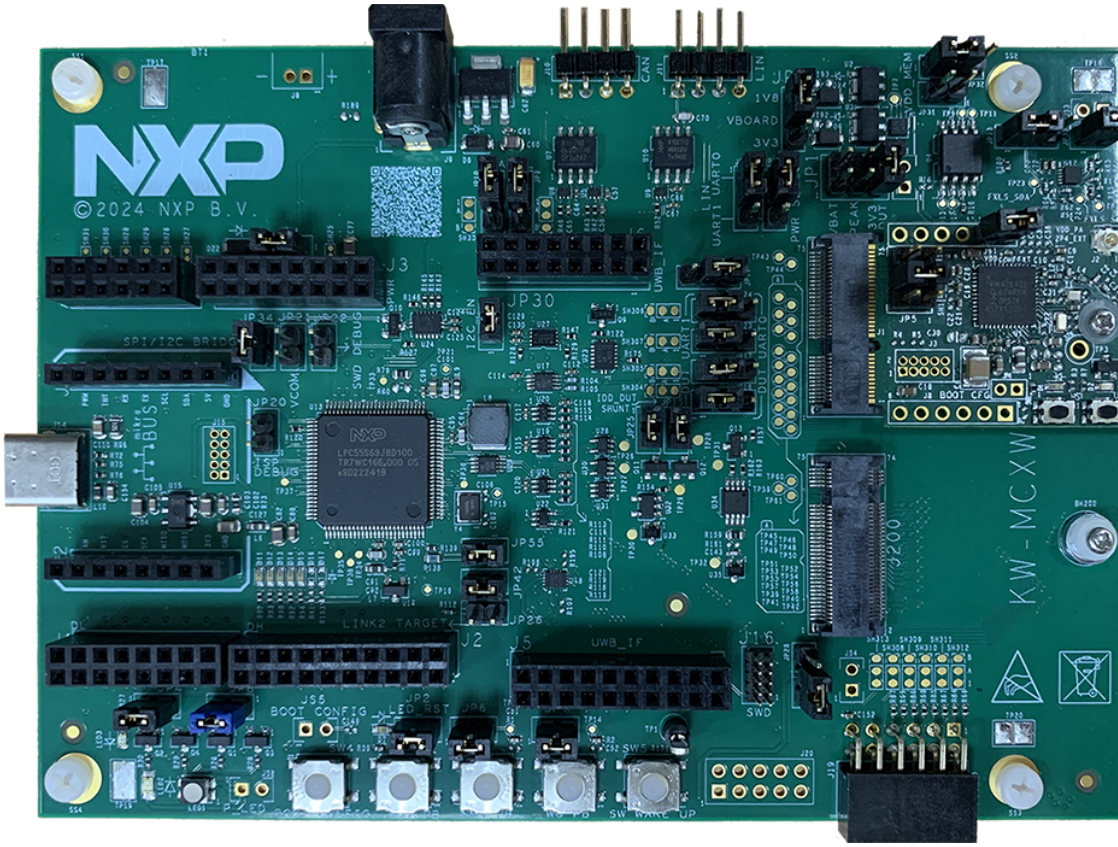
FRDM-MCXW71 board



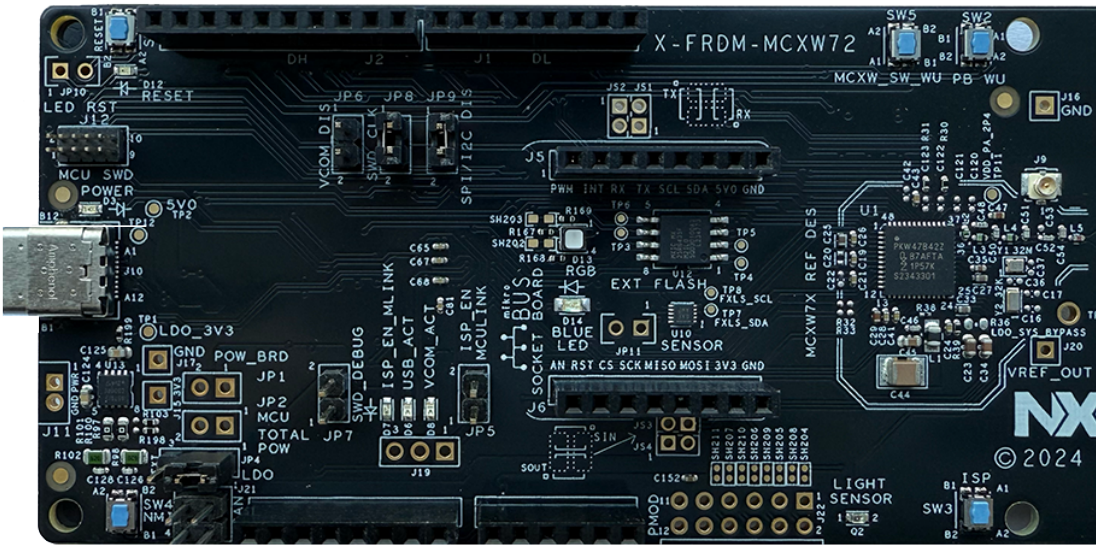
KW47-EVK board



KW47-LOC board



MCX-W72-EVK board



FRDM-MCXW72 board

Getting Started with west Build Environment These steps are intended for **GitHub users** working with Bluetooth Low Energy (BLE) demo applications from the GitHub MCUXpresso SDK repository.

Follow the steps in the order presented in Getting Started with MCUXpresso SDK Repository to install the west environment.

Command Line to Generate/Build Projects

Example: w_uart **on** kw47evk

To build the w_uart example for the kw47evk board using the IAR toolchain and generate an IDE project, use the following command:

```
west build -b kw47evk examples/wireless_examples/bluetooth/w_uart/bm/ \ --toolchain=iar -t guiproject -
↪Dcore_id=cm33_core0 --pristine
```

Argument	Description
-b kw47evk	Specifies the board
--toolchain=iar	Specifies the toolchain (default is armgcc)
-t guiproject	Creates an IDE project
-Dcore_id=cm33_core0	Core ID, if available
--pristine	Cleans the build folder before building

Use the following steps in order to build and flash the BLE software demo applications on all toolchains presented in:

- Building and flashing the BLE software demo applications using IAR Embedded Workbench
- Building and flashing the BLE Software Demo applications using MCUXpresso IDE
- Building and flashing the BLE software demo applications using Visual Studio Code

Installing the Connectivity Package To install the Connectivity Package, configure and download the package archive from the staging system on the https://mcuxpresso.nxp.com website. You can simply download the precreated package archive if it is available on the same website.

Note: Use the default location for the package (C:\NXP) and create a subfolder there specific to each device and release.

Note: Prior to loading any wireless SDK example, update your NBU image with the provided binaries in the following folder of the SDK: ../middleware/wireless/ble_controller/bin

Building the binaries This section describes the necessary steps for obtaining the binary files for usage with the boards.

Prerequisites To build any of the demo applications, you need the following toolchain:

- IAR Embedded Workbench for Arm (details in release note)
- MCUXpresso IDE (details in release note)
- Visual Studio Code with "*MCUXpresso for Visual Studio Code*" extension (details in release note)
- Teraterm (version 4.105 or higher)

The Connectivity Software Package does not include support for any other toolchains. The packages must be built with the debug configuration to enable debugging information. This package includes various sample applications that can be used as a starting point.

Conventions for building the *wireless_UART* application. The following sections present the steps required for building the *wireless_UART* application. All applications can be found using the following placeholders for text:

- <connectivity_path>: represents the root path for the SDK.
- <board>: represents the target board for the demo app, "kw47evk" in this case.
- <RTOS>: represents the scheduler or RTOS used by the app; it can be either "bm" or "freertos".
- <demo_app>: represents the demo application name.
- <IDE>: represents the integrated development environment used to build projects; "iar" in this case.
- <core_id>: represents the target CPU on which the application will run, "cm33_core0" in this case (applicable only on KW47-EVK, KW47-LOC, MCX-W72-EVK and FRDM-MCXW72 boards).
- The general folder structure of the demo applications is the following:

  <connectivity_path>\boars\<board>\wireless_examples\bluetooth\<demo_app>\
  <core_id>\<RTOS>\<IDE>

Selected application: w_uart

Board: One of the following boards:

- **kw45b41zevk** (for this guide)
- kw45b41zloc
- k32w148evk
- frdmmcxw71
- kw47evk
- kw47loc
- mcxw72evk
- frdmmcxw72

RTOS: FreeRTOS

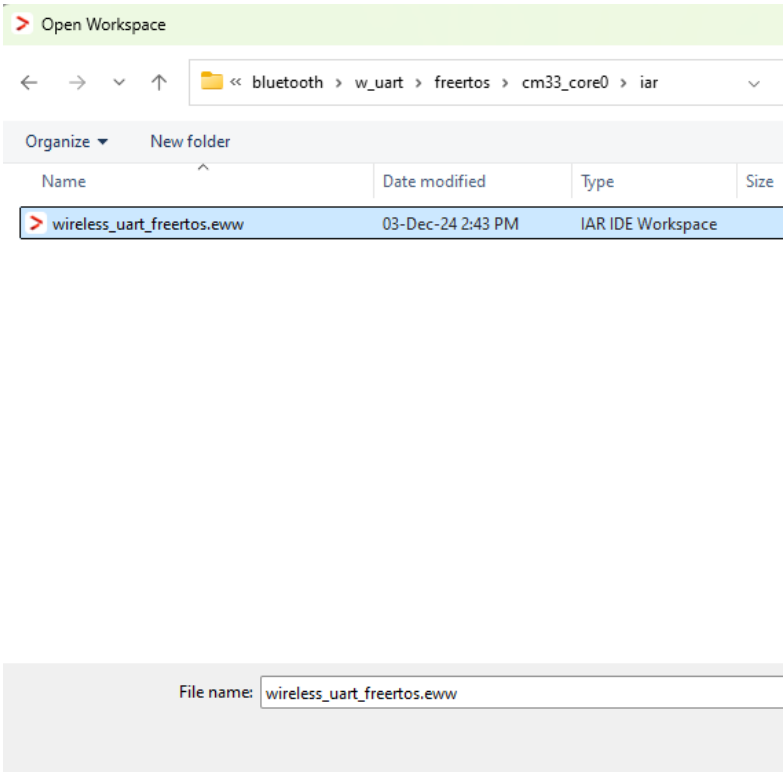
Resulting location:

<connectivity_path>\boards\<kw45b41zevk / kw45b41zloc / k32w148evk / frdmmcxw71 / kw47evk / kw47loc / mcxw72evk / frdmmcxw72>\wireless_examples\bluetooth\w_uart\freertos\<IDE>

Building and flashing the BLE software demo applications using IAR Embedded Workbench

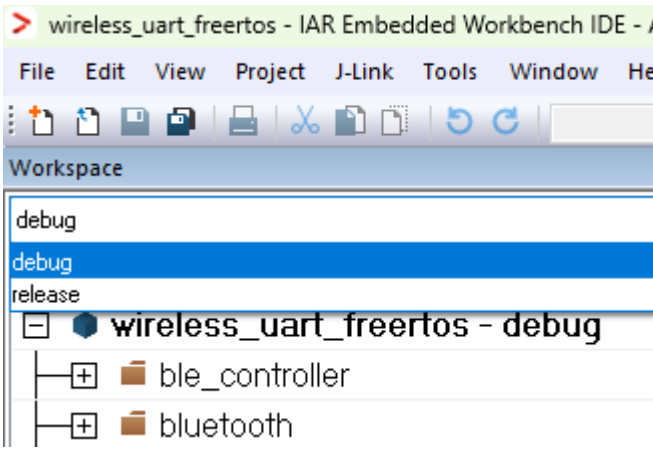
Use the following steps in order to build and flash the BLE software demo applications using the IAR Embedded Workbench:

1. First unpack the contents of the archive to a folder on the local disk. Then, navigate to the resulting location starting from the SDK root directory.
2. Open the IAR workspace file (*.eww file format) highlighted file in the figure below.



Wireless UART IAR demo project location

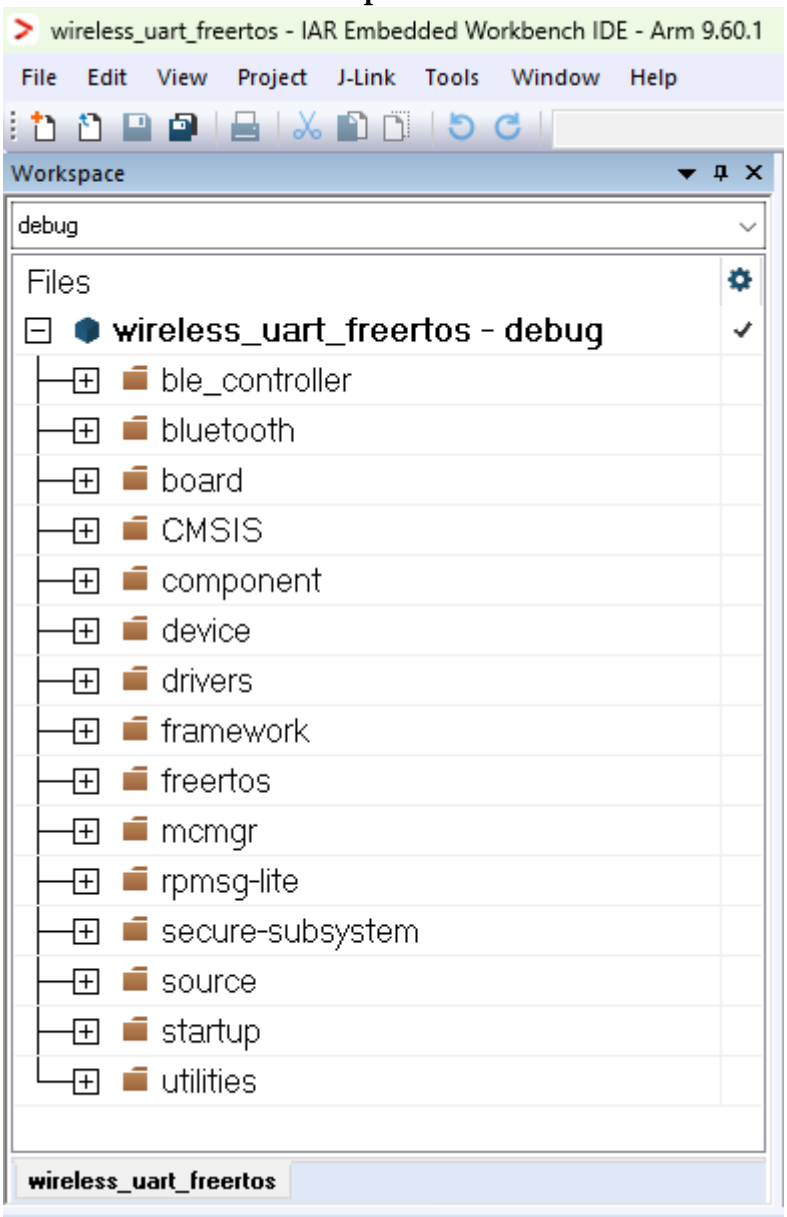
3. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.



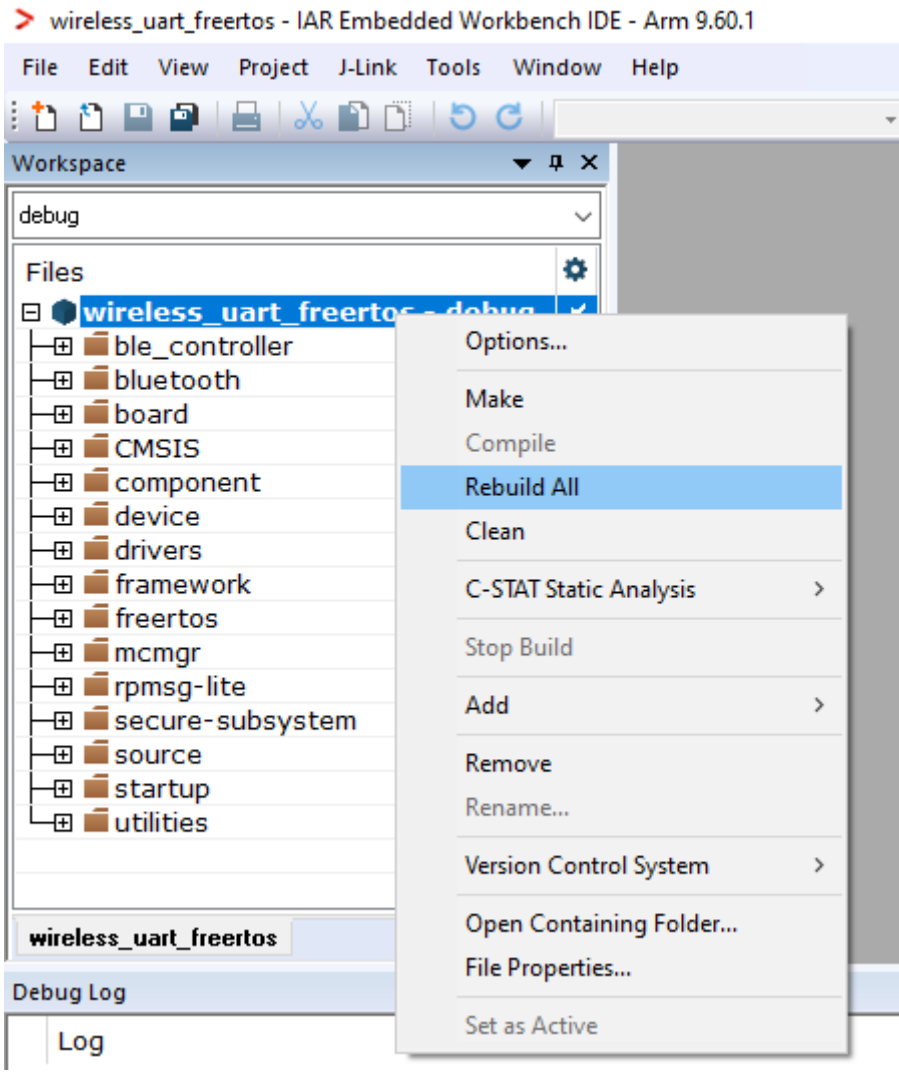
Select the desired configuration (Debug or Release)

The figure below shows the Wireless UART - IAR workspace.

Wireless UART - IAR workspace



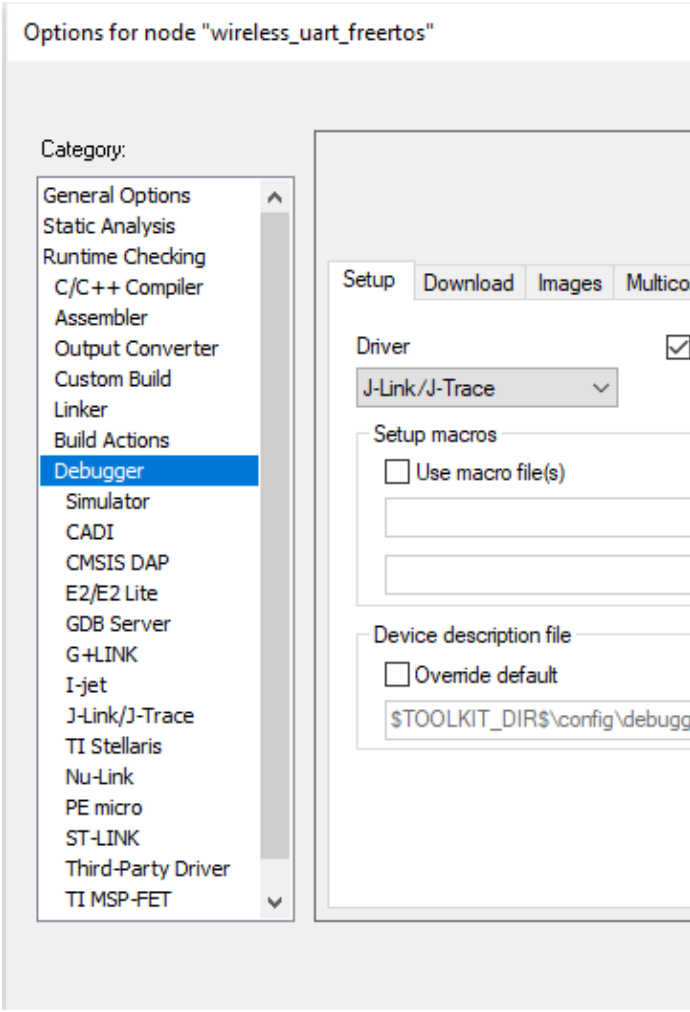
4. Build the Wireless UART project using the options shown in the figure.



Build Wireless UART application

5. Make the appropriate debugger settings in the project options window, as seen in the next figure.

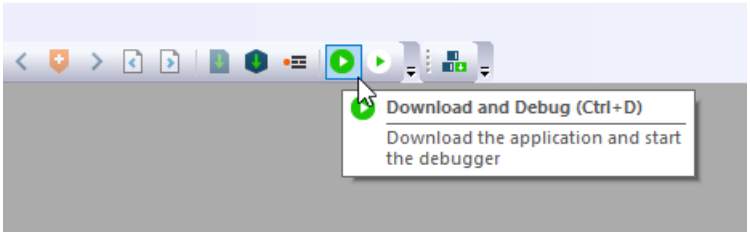
   Go to: **Project** > **Options (Alt+F7)** > **Debugger** > **Setup (tab)** > **Driver** > **J-Link/J-Trace**



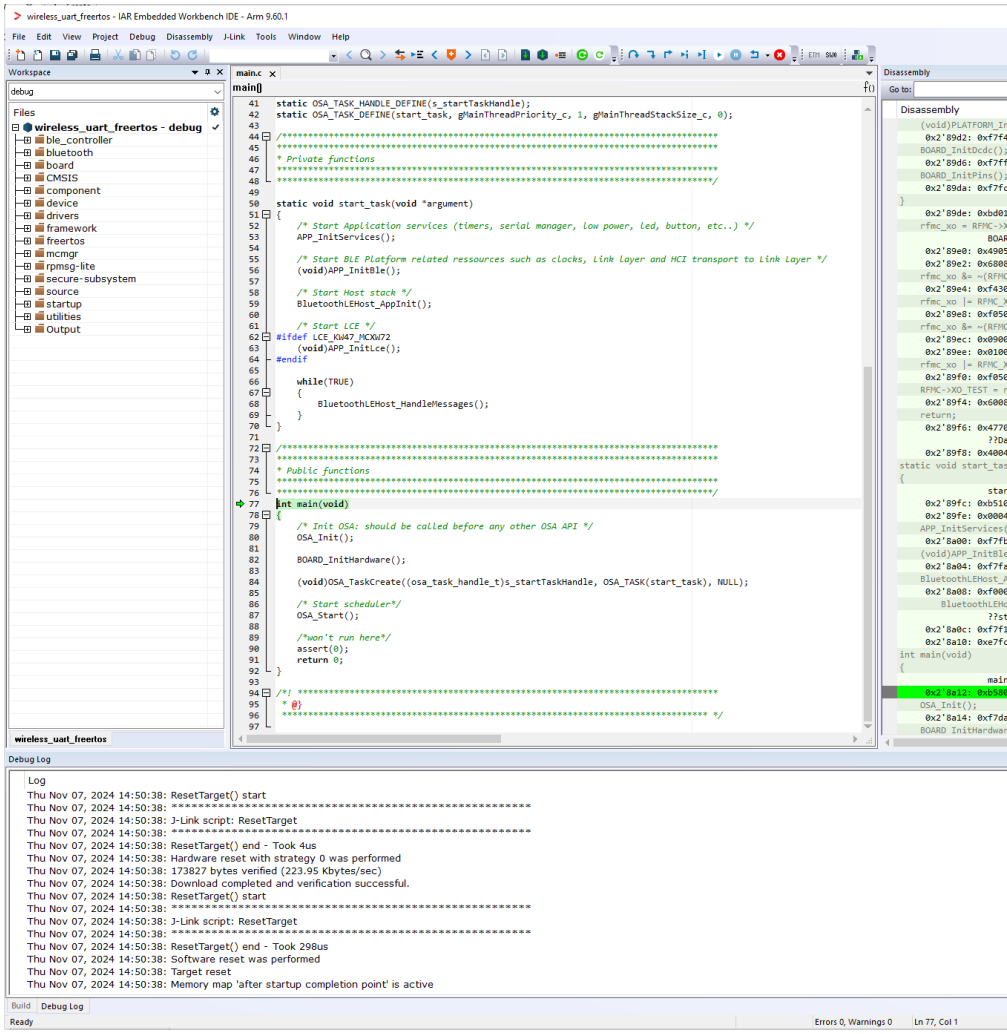
Debugger Settings for the Wireless UART project

6. Click the "**Download and Debug**" button (or **CTRL+D**) to flash the executable onto the board.

Download and Debug the Wireless UART application



7. Press **Go** (**F5**). At this moment, the board starts running the application.

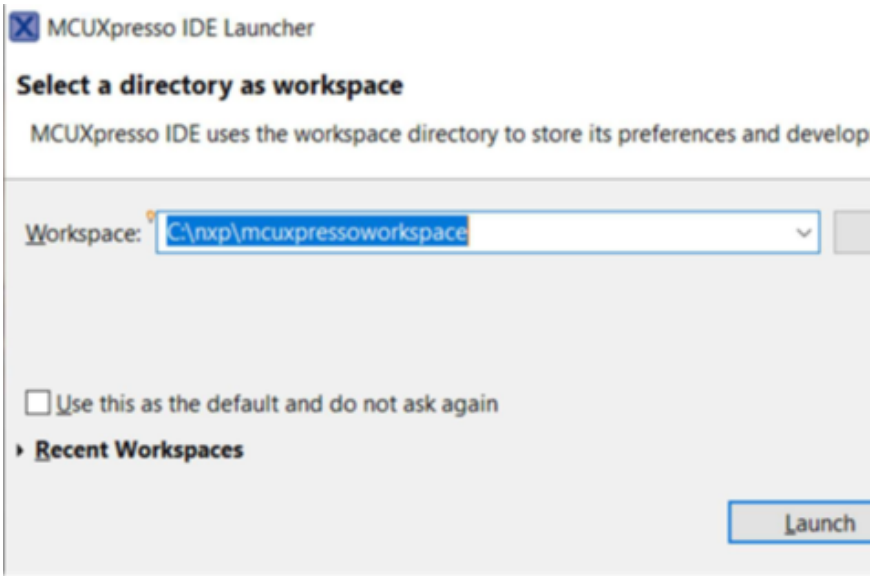


Running the code on IAR

Parent topic:*Building the binaries*

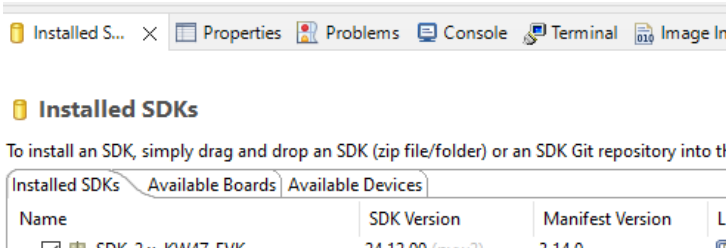
Building and flashing the BLE Software Demo applications using MCUXpresso IDE To build and flash the BLE software demo applications using MCUXpresso IDE, follow the steps listed below:

1. Open MCUXpresso IDE and open an existing or new workspace location.



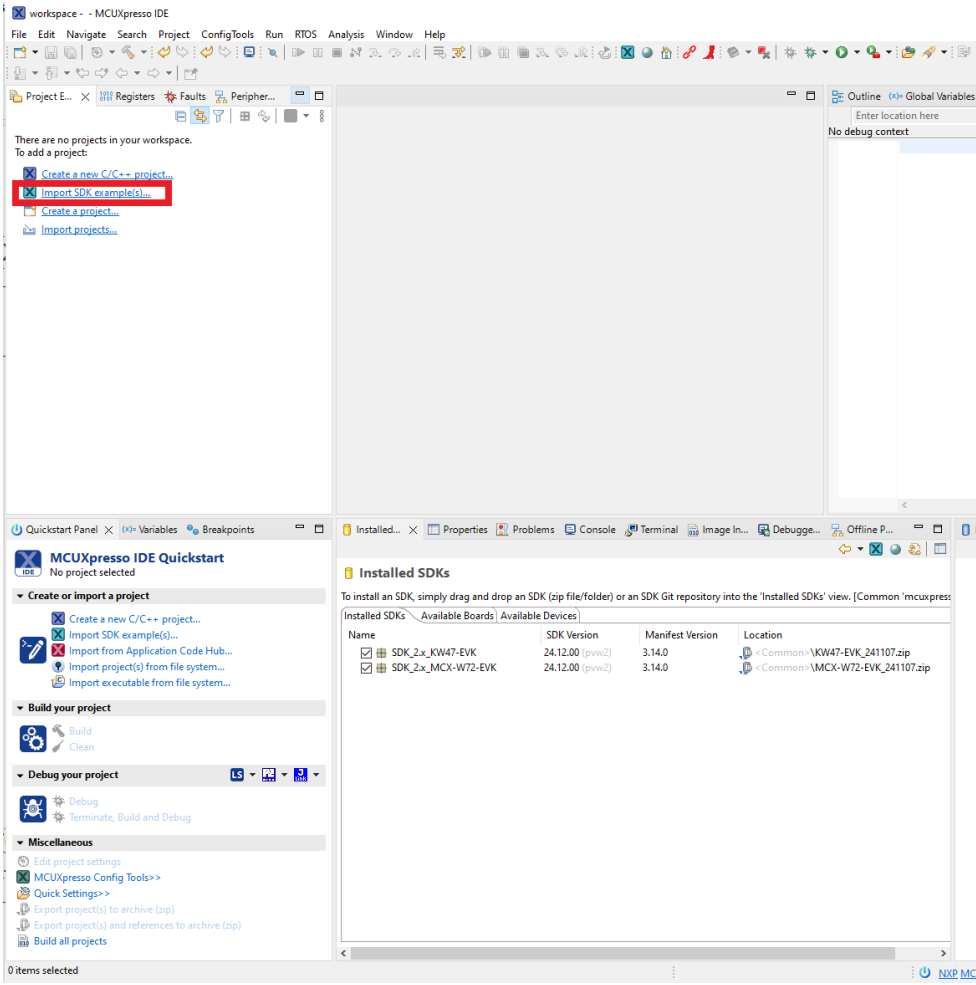
Select MCUXpresso IDE workspace

2. Drag and drop the package archive into the **MCUXpresso Installed SDKs** area in the lower right of the main window.



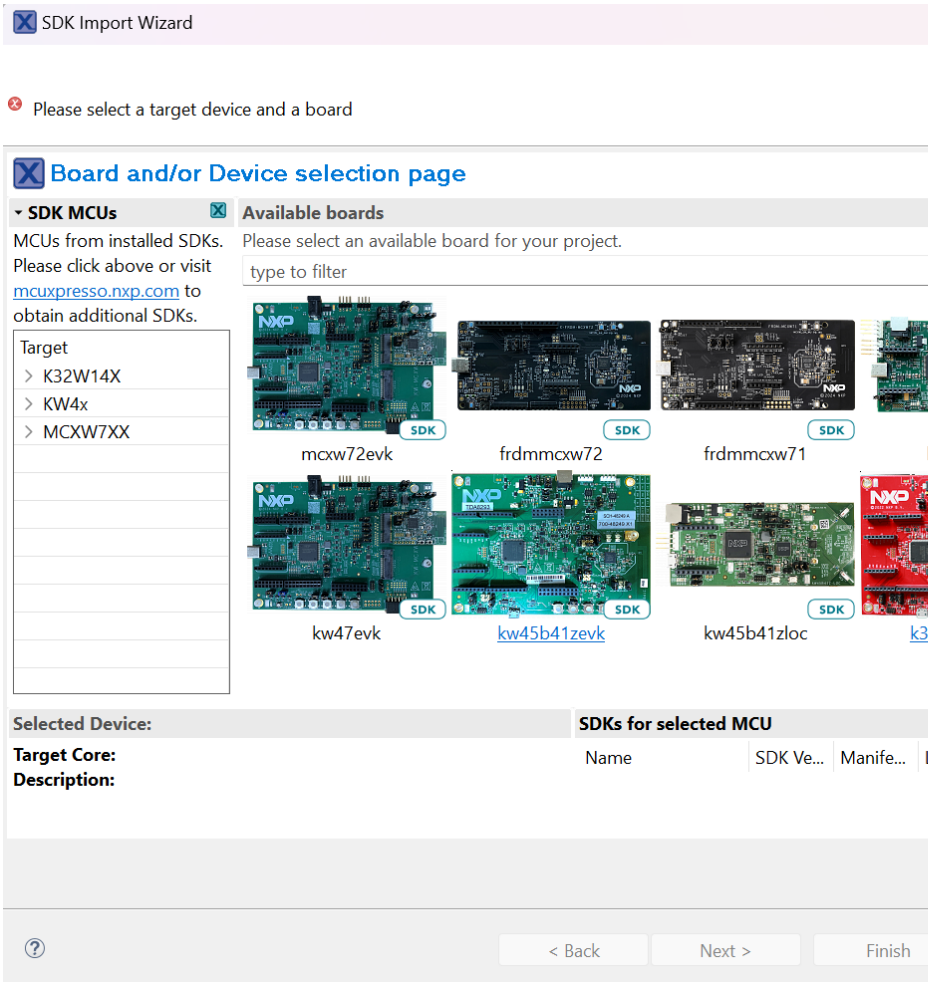
Installed SDKs in MCUXpresso IDE workspace

3. After the SDK is loaded successfully, select the "**Import the SDK examples(s)...**" to add examples to your workspace.

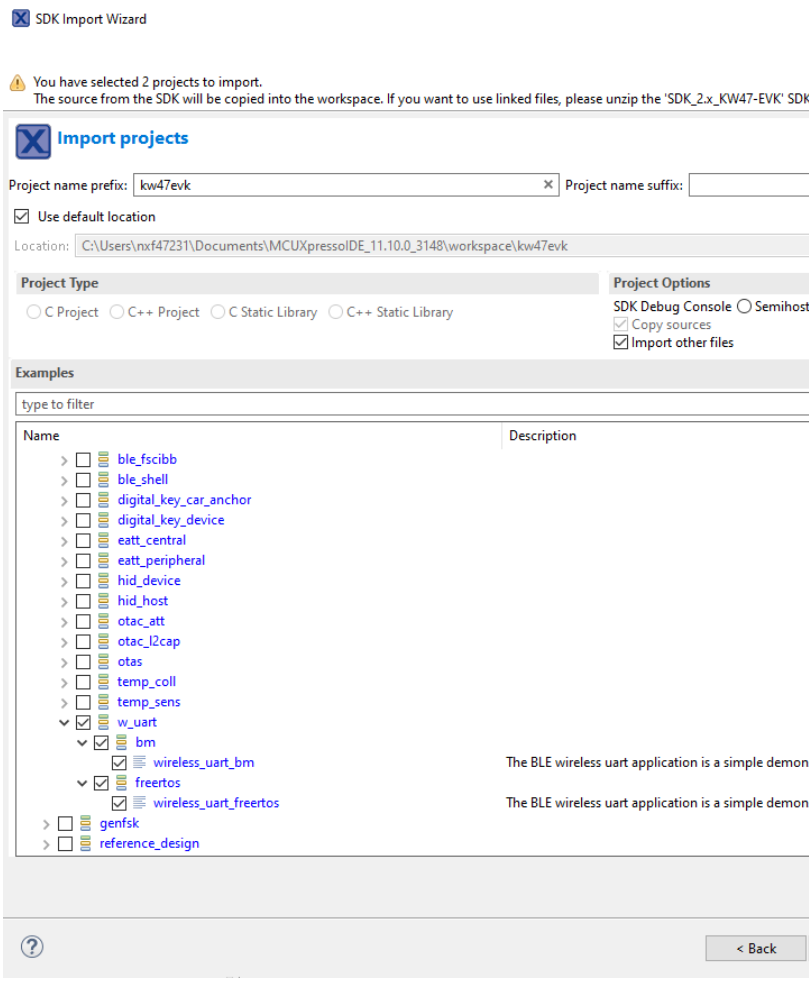


Importing SDK example(s)

4. To select the desired example(s), select the *kw45b41zevk / kw45b41zloc / k32w148evk / kw47evk / frdmmcxw71 / frdmmcxw72 / mcxw72evk / kw47loc* board and then click the "**Next**" button:

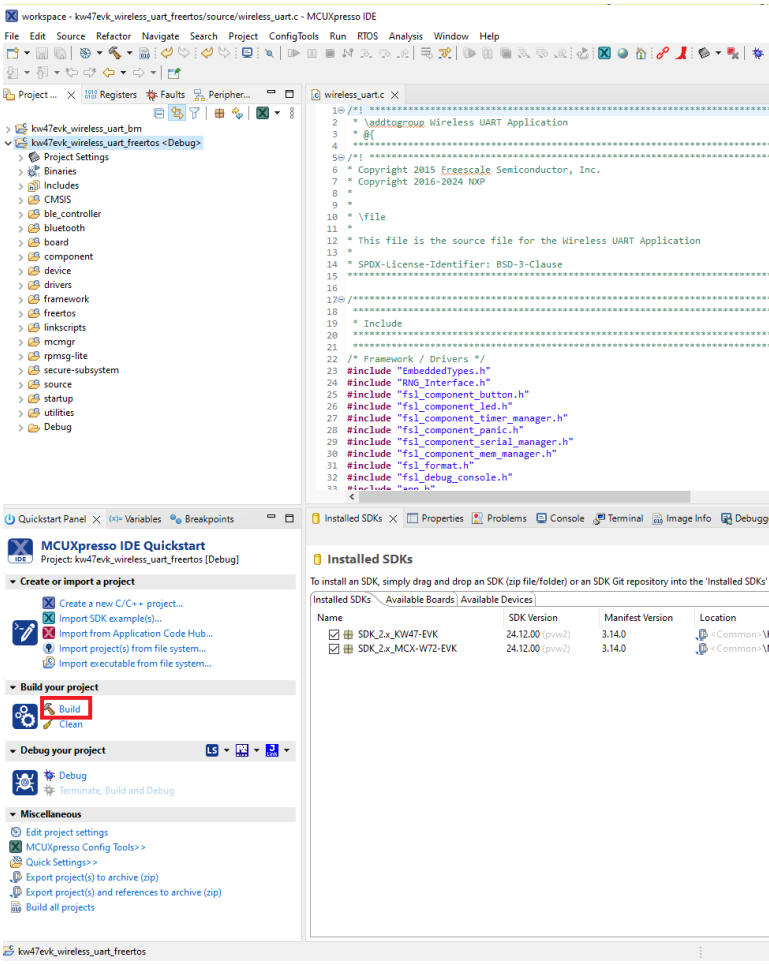


Selecting the KW47-EVK board



Select Wireless UART FreeRTOS project

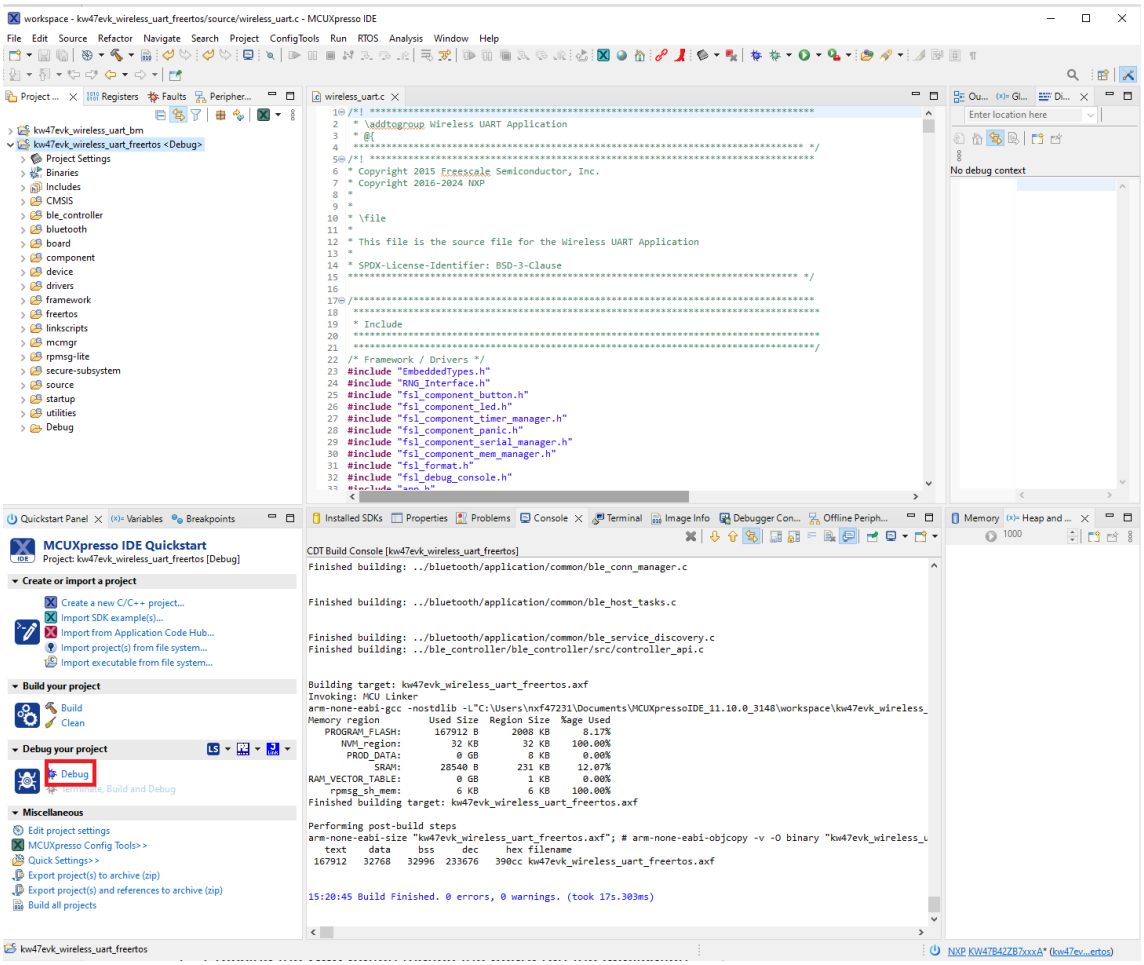
5. Build the wireless_uart_freertos project.



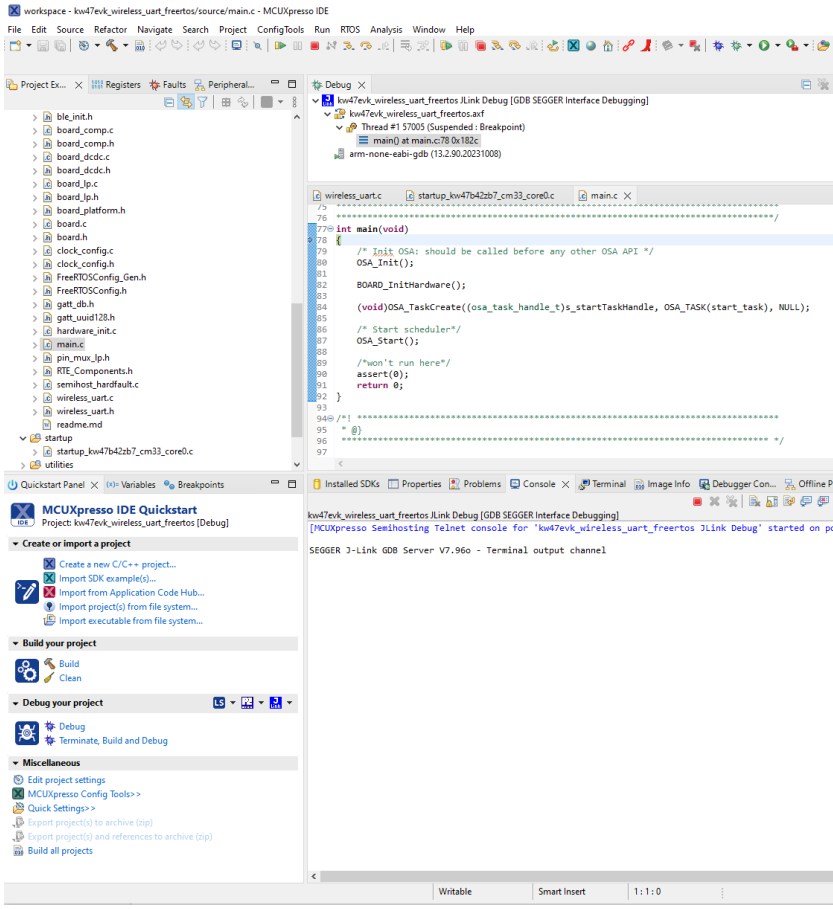
Build the Wireless UART FreeRTOS project

6. Click the “**Debug**” button to download the executable onto the board. Make sure you select the appropriate device to flash.

Download and debug the Wireless UART FreeRTOS project



7. Pressing the **Run** button makes the board run the application.

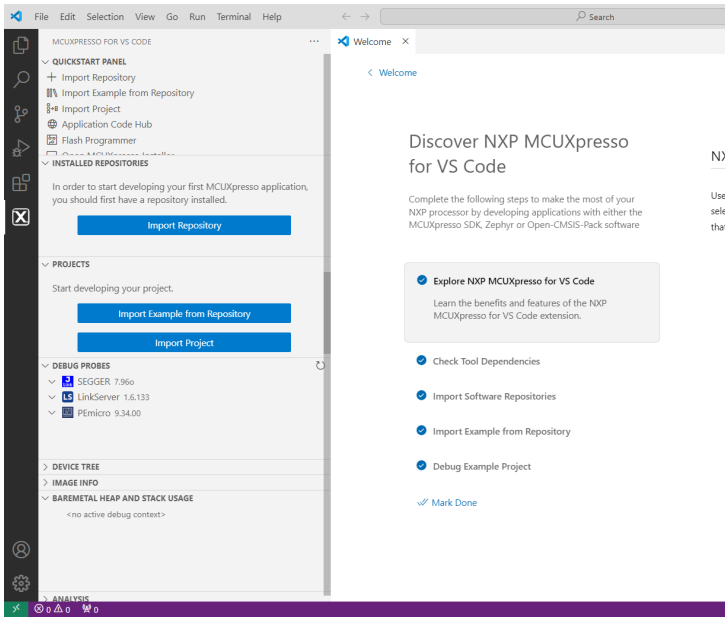


Running the code on MCUXpresso IDE

Parent topic:*Building the binaries*

Building and flashing the BLE software demo applications using Visual Studio Code To build and flash the BLE software demo applications using Visual Studio Code, follow the steps listed below:

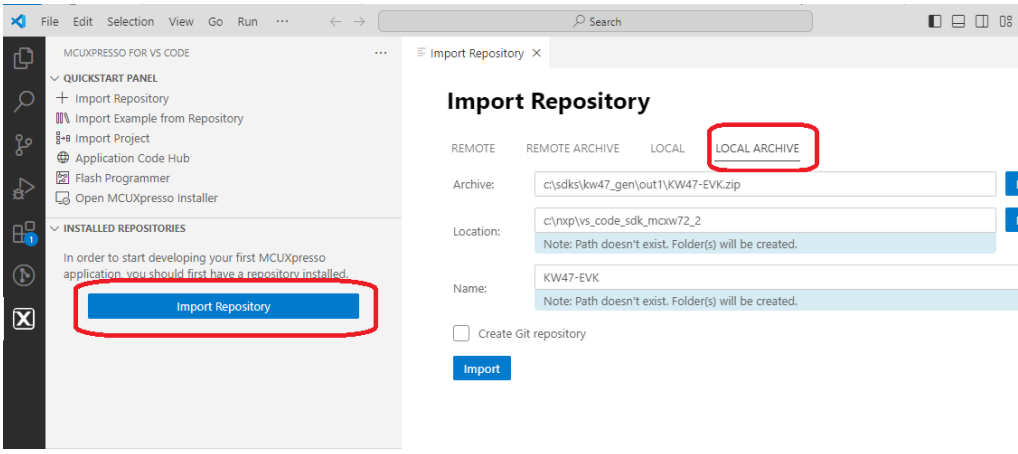
1. Open Visual Studio Code and open the MCUXpresso for Visual Studio Code extension as shown in the figure below.



MCUXpresso for Visual Studio Code extension

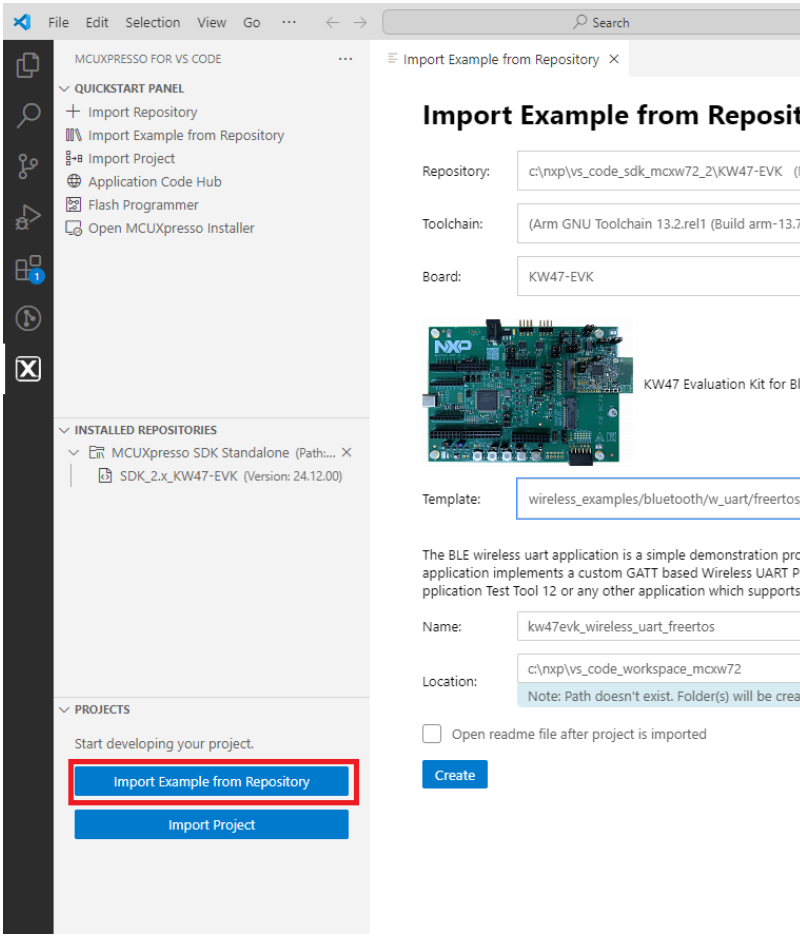
2. Import the SDK package: click “**Import Repository**”. Then, choose the “**Local**” option (if the SDK is archived use “**Local archive**”), browse to the path of the SDK you want, and click “**Import**”.

Steps to import the SDK

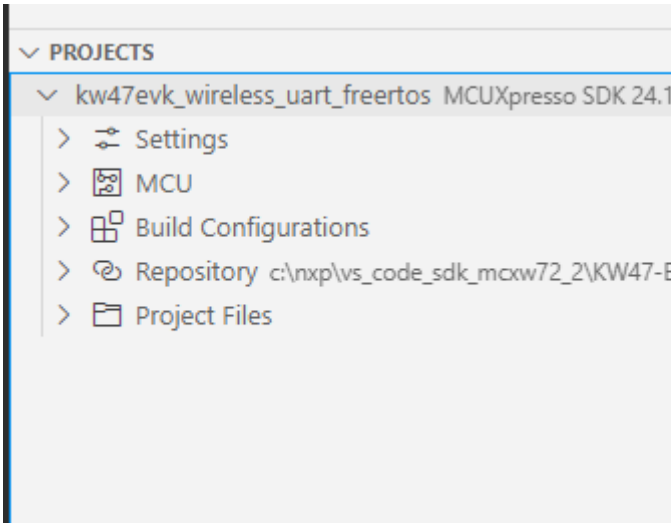


3. After the SDK is loaded successfully, select the “**Import Example from Repository**” to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click “**Create**”.

Steps to import the example application

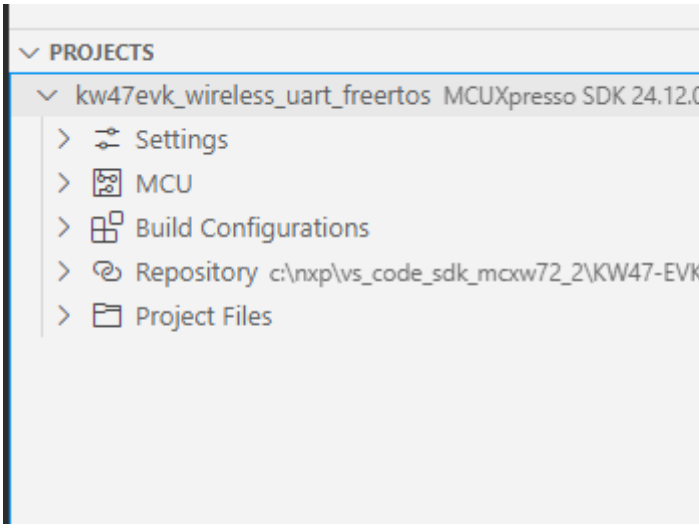


4. The application now appears in the “**Projects**” tab on the left. Build the application by pressing the “**Build selected**” button.



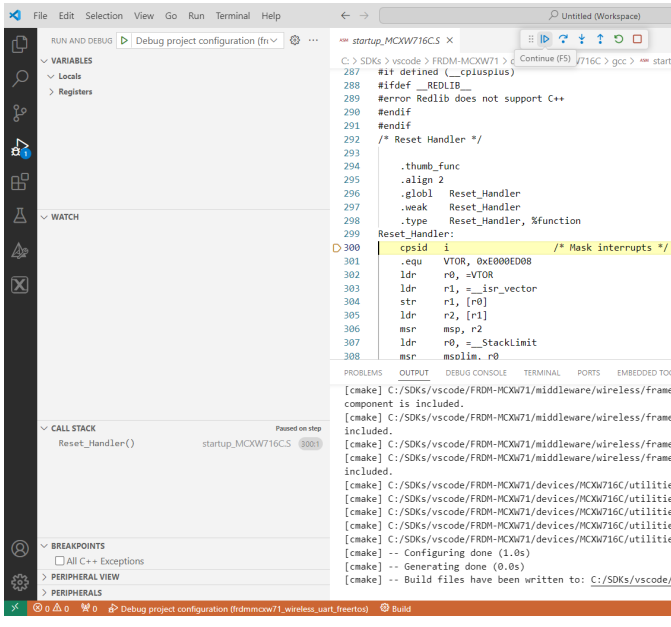
Building the Wireless UART FreeRTOS application

5. After the build is completed successfully, debug the application by clicking the "**Debug**" button.



Debug the Wireless UART FreeRTOS application

6. Press the run ("**Continue**") button twice to run the application on the board.

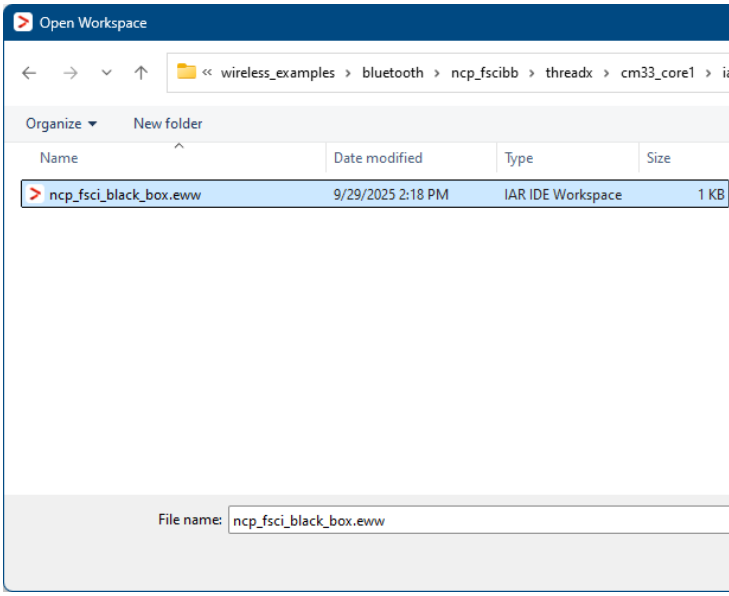


Running the Wireless UART FreeRTOS application

Parent topic:*Building the binaries*

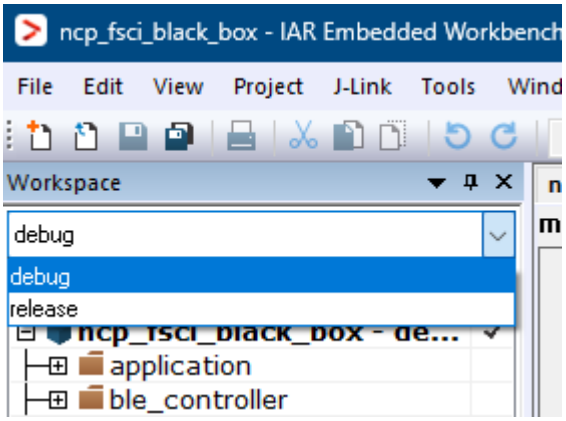
Building and flashing the BLE Extended NBU software demo applications using IAR Embedded Workbench Use the following steps in order to build and flash the BLE software demo applications using the IAR Embedded Workbench:

1. First unpack the contents of the archive to a folder on the local disk. Then, navigate to the resulting location starting from the SDK root directory.
2. Open the IAR workspace file (*.eww file format) highlighted file in the figure below.



NCP FSCI Blackbox IAR demo project location

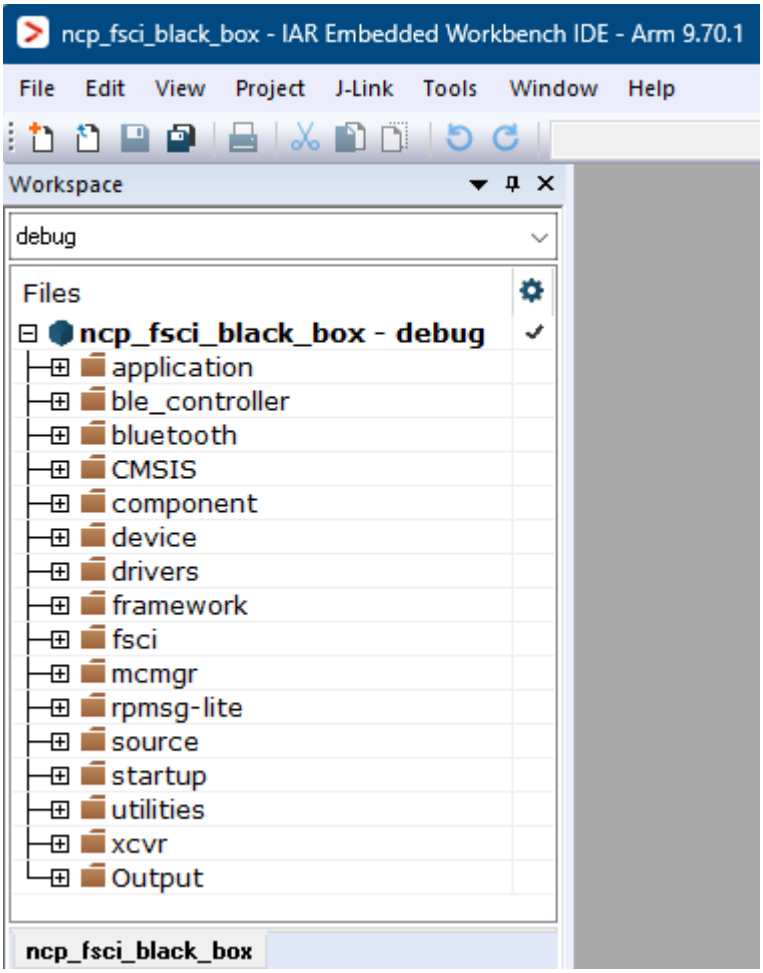
3. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.



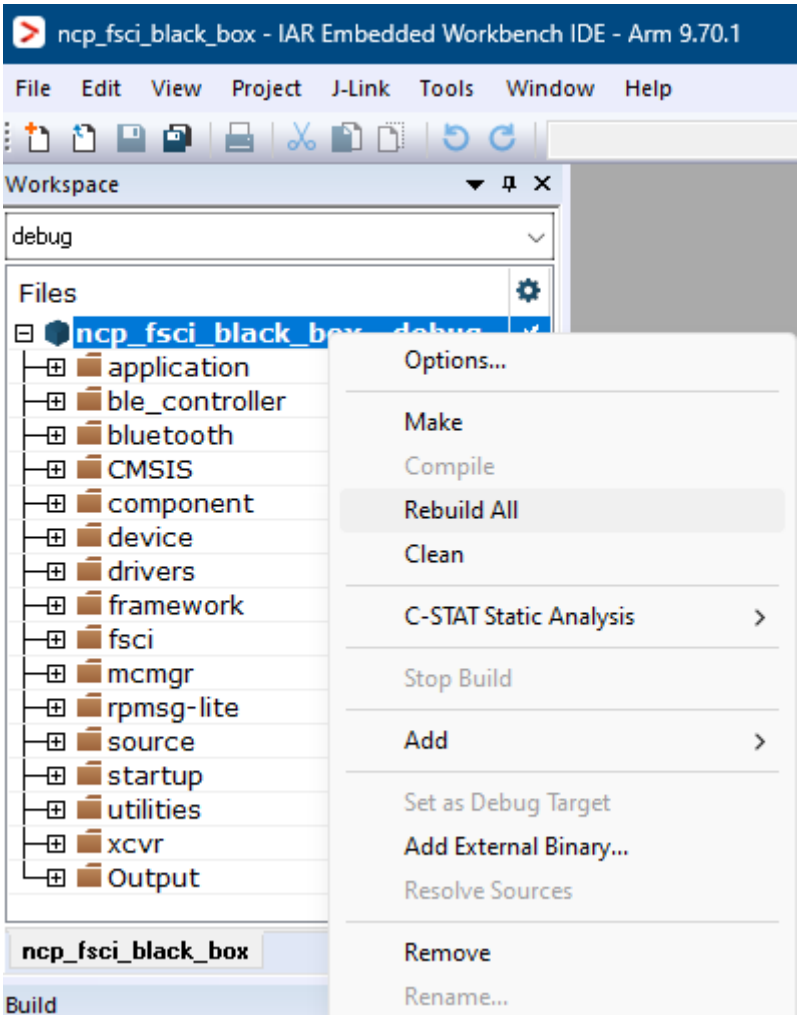
Select the desired configuration (Debug or Release)

The figure below shows the NCP FSCI Blackbox - IAR workspace.

NCP FSCI Blackbox - IAR workspace



4. Build the NCP FSCI Blackbox project using the options shown in the figure.

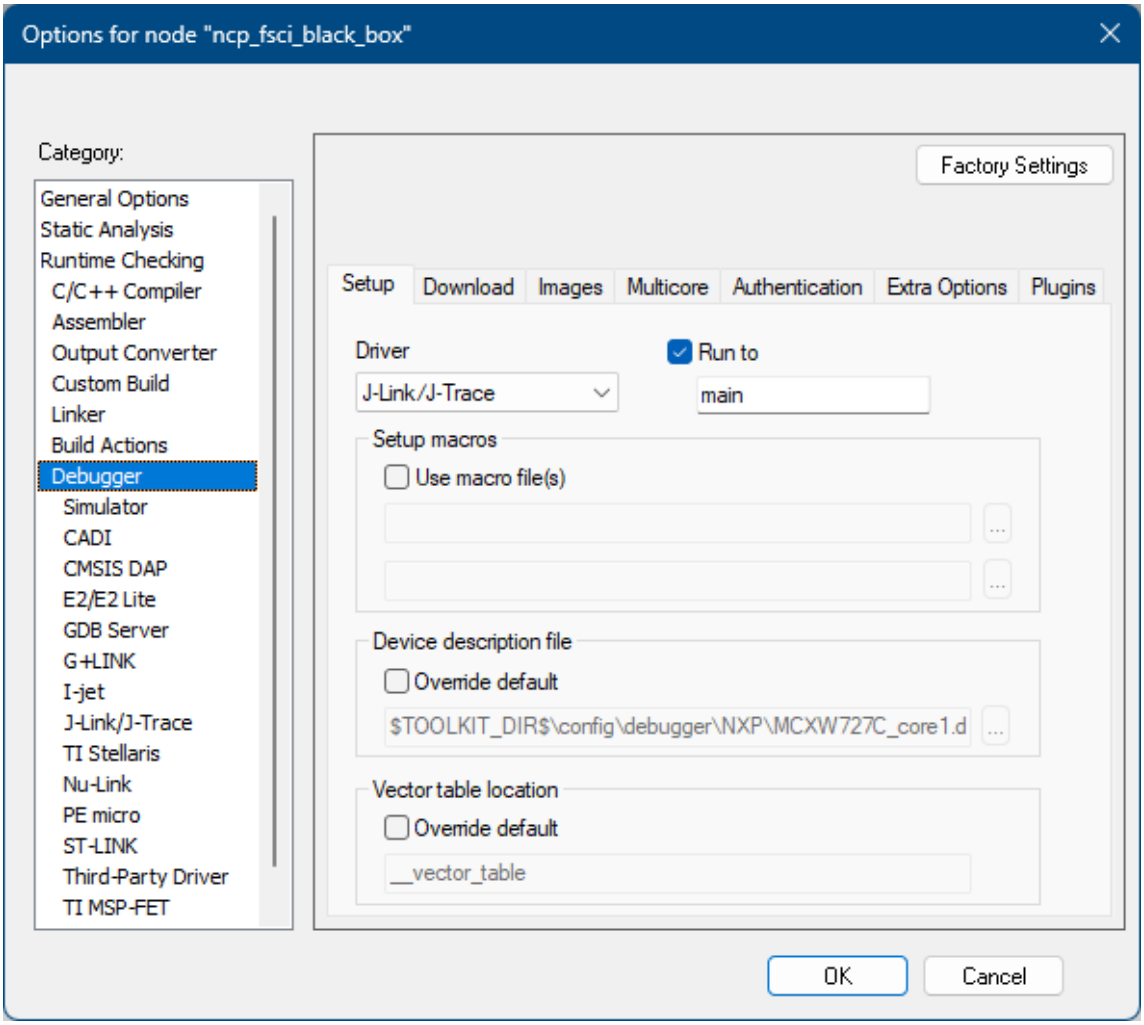


Build NCP FSCI Blackbox application

5. Make the appropriate debugger settings in the project options window, as seen in the next figure.

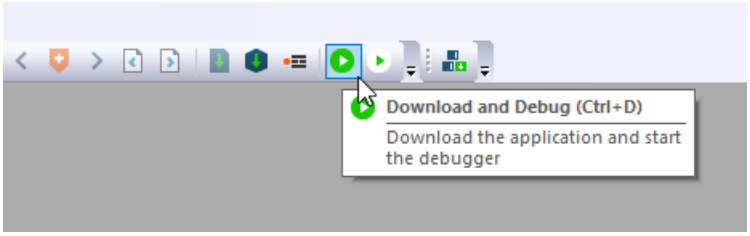
   Go to: **Project** > **Options (Alt+F7)** > **Debugger** > **Setup (tab)** > **Driver** > **J-Link/J-Trace**

   Debugger Settings for the NCP FSCI Blackbox project

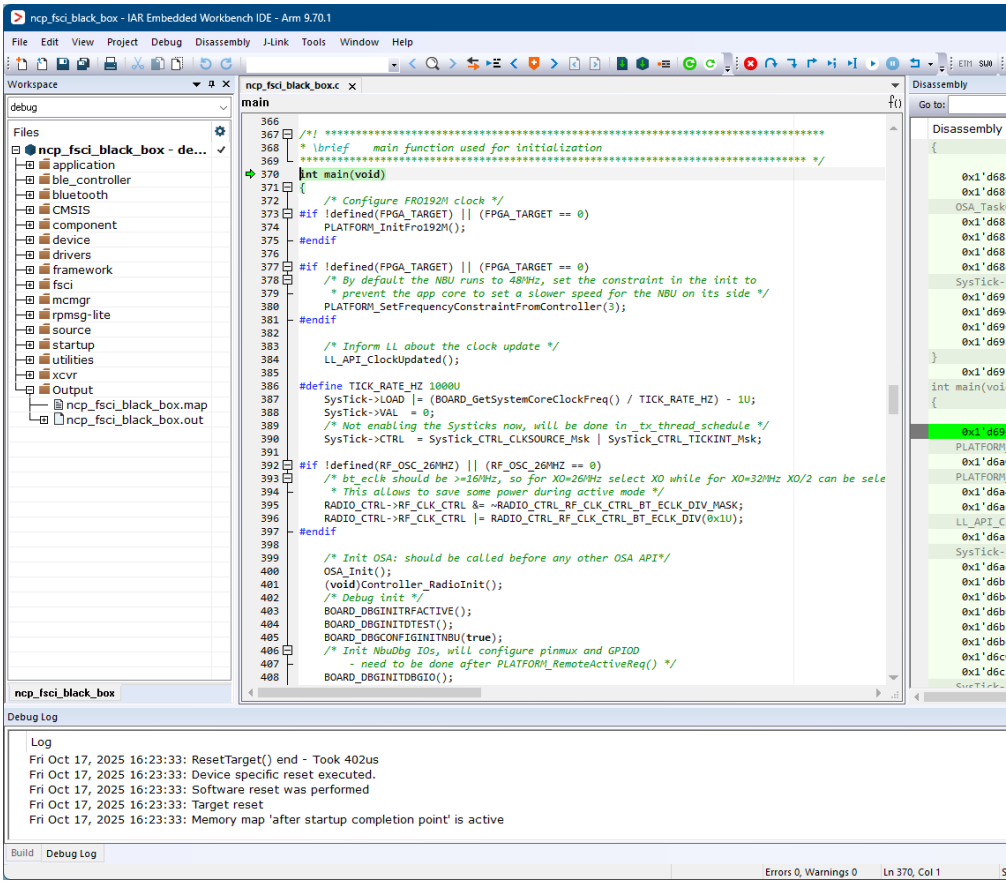


6. Click the "**Download and Debug**" button (or **CTRL+D**) to flash the executable onto the board.

Download and Debug the NCP FSCI Blackbox application

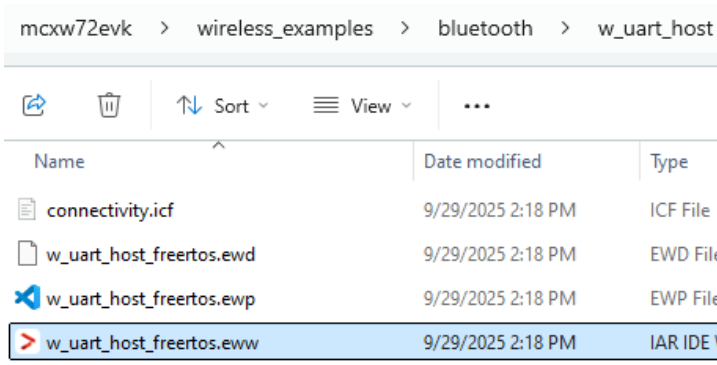


7. Press **Go (F5)**. At this moment, the board starts running the application.



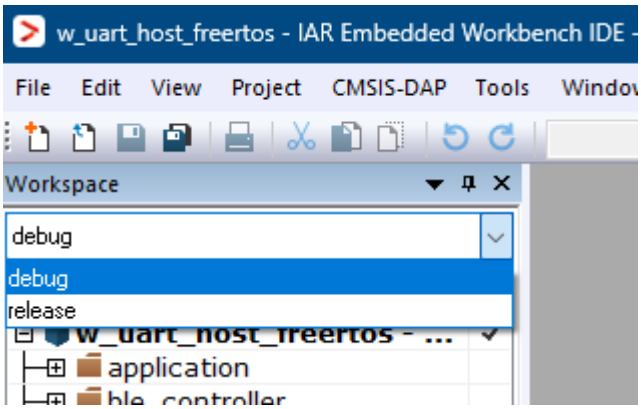
Running the code on IAR

8. Open the Core0 IAR workspace file (*.eww file format) highlighted file in the figure below.



Wireless UART Host IAR demo project location

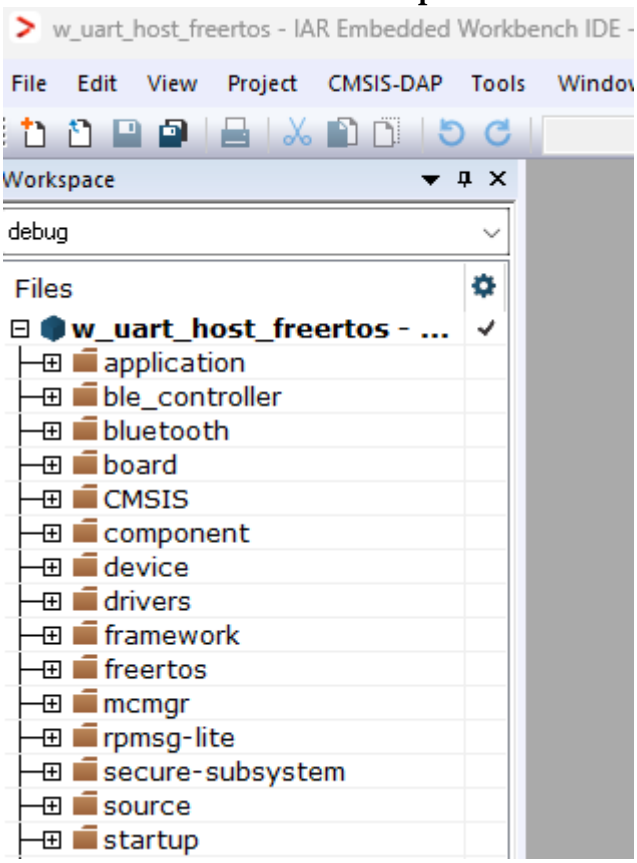
9. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.



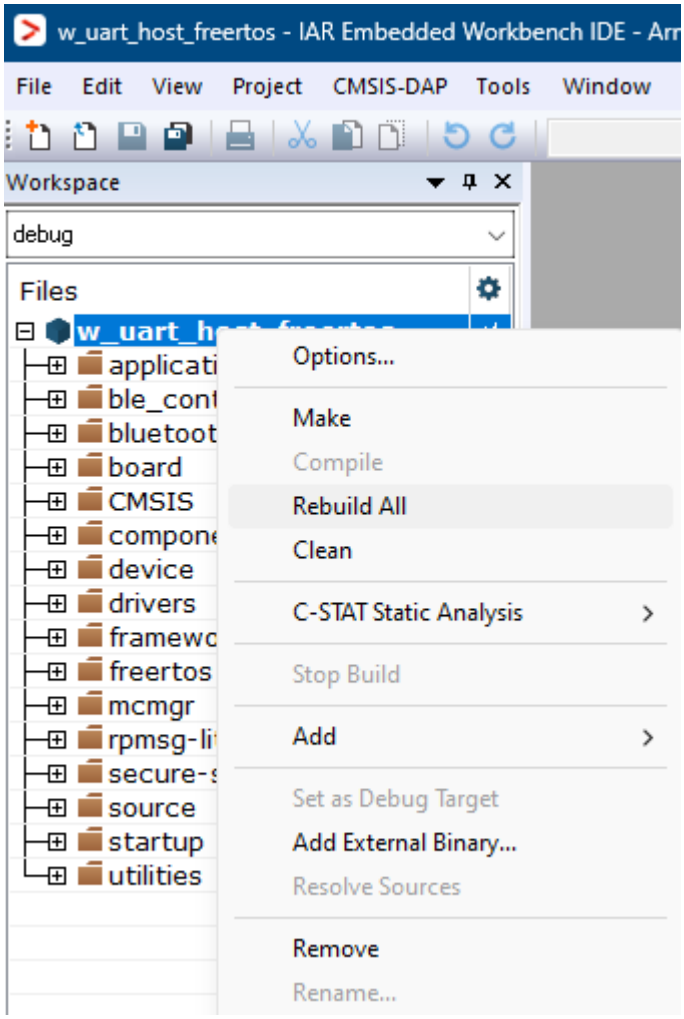
Select the desired configuration (Debug or Release)

The figure below shows the Wireless Uart Host - IAR workspace.

Wireless Uart Host - IAR workspace



10. Build the Wireless Uart Host project using the options shown in the figure.

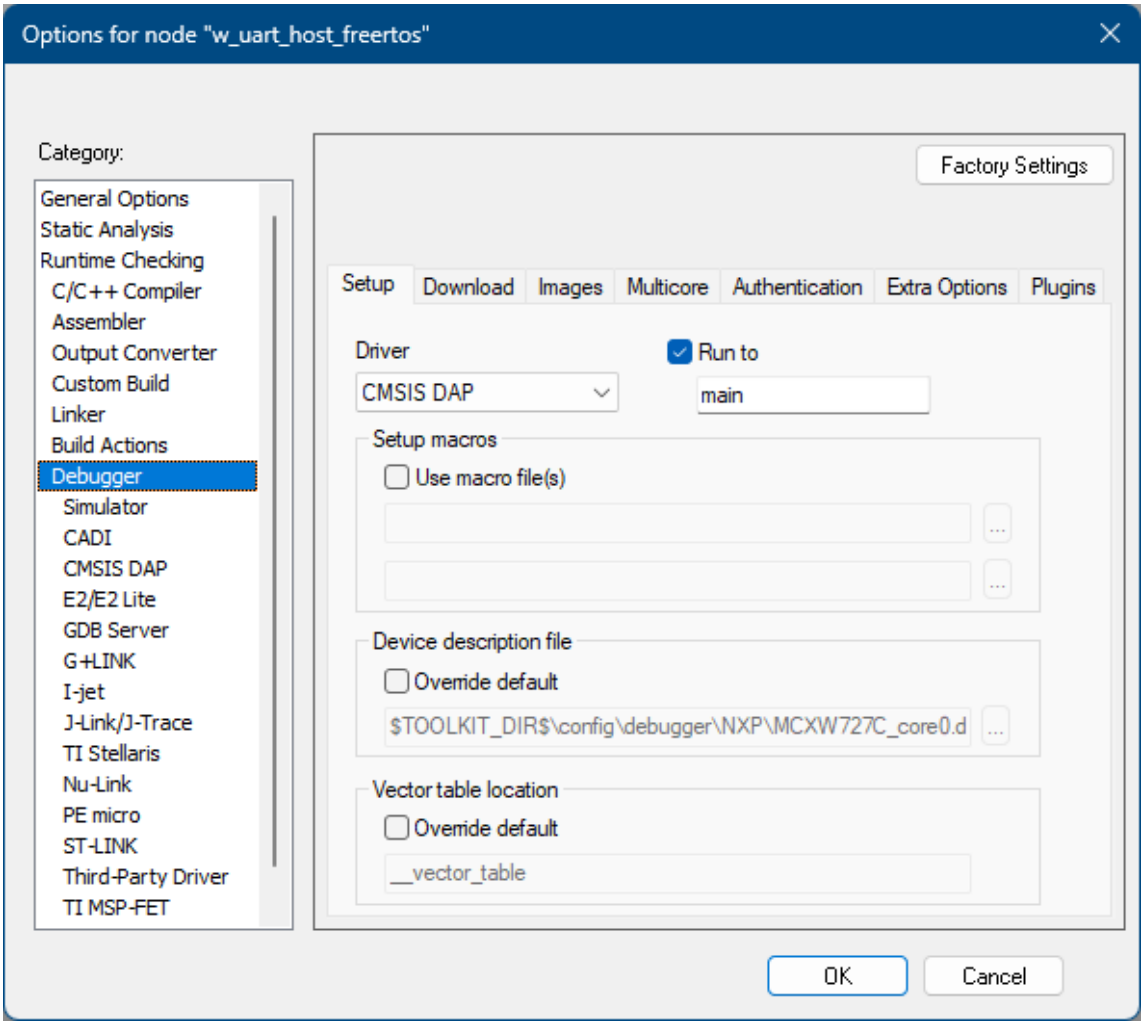


Build Wireless Uart Host application

11. Make the appropriate debugger settings in the project options window, as seen in the next figure.

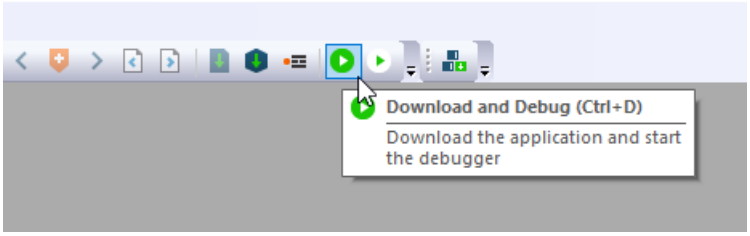
    Go to: **Project > Options (Alt+F7) > Debugger > Setup (tab) > Driver > J-Link/J-Trace**

    Debugger Settings for the Wireless Uart Host project

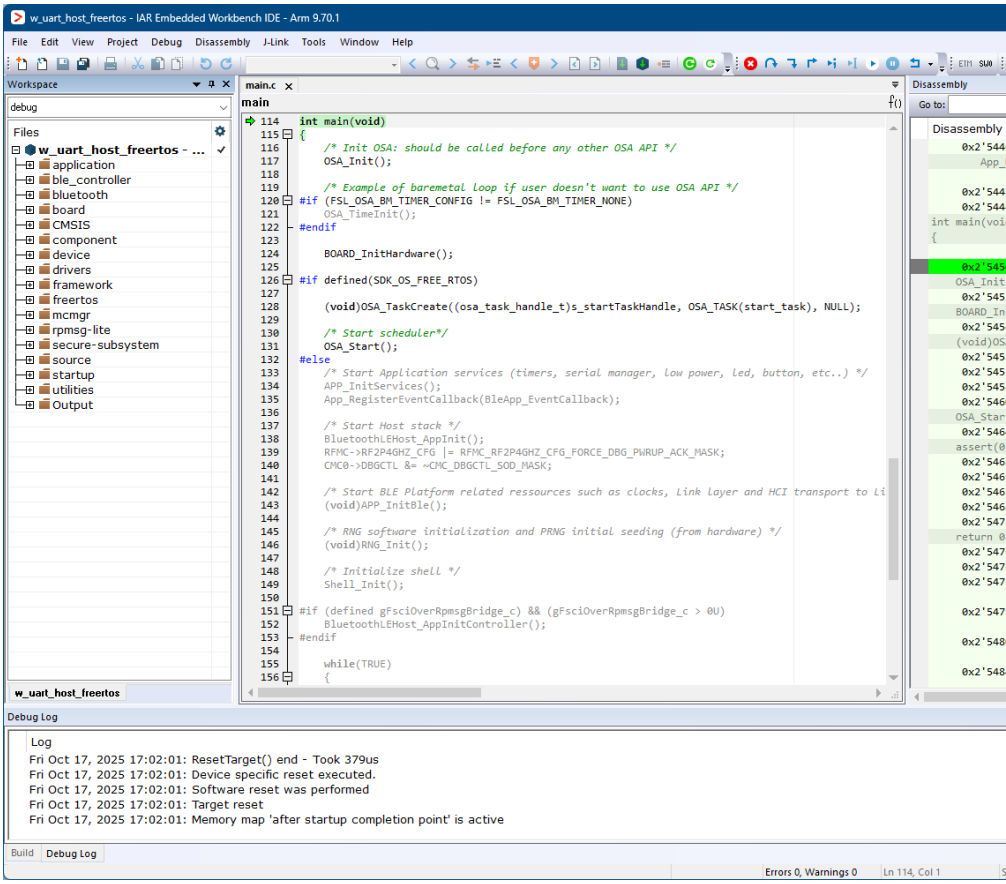


12. Click the "**Download and Debug**" button (or **CTRL+D**) to flash the executable onto the board.

    Download and Debug the Wireless Uart Host application



13. Press **Go (F5)**. At this moment, the board starts running the application.

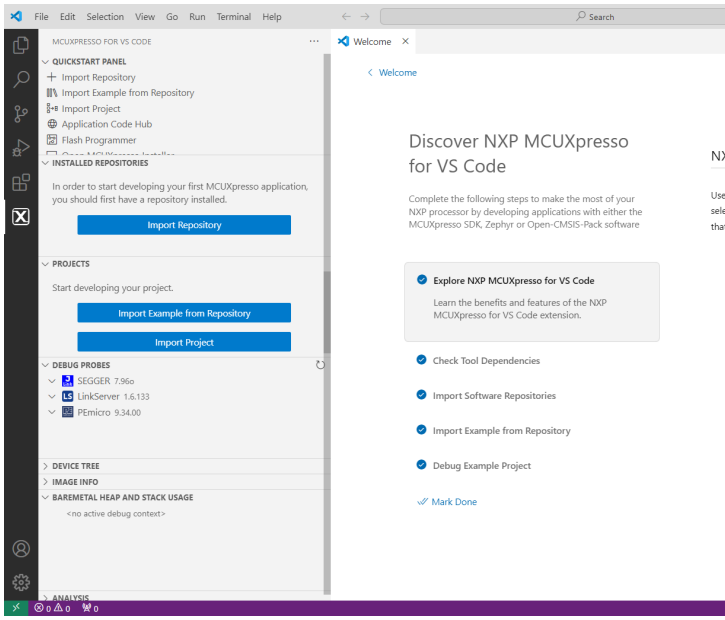


Running the code on IAR

Parent topic:*Building the binaries*

Building and flashing the BLE Extended NBU software demo applications using Visual Studio Code To build and flash the BLE software demo applications using Visual Studio Code, follow the steps listed below:

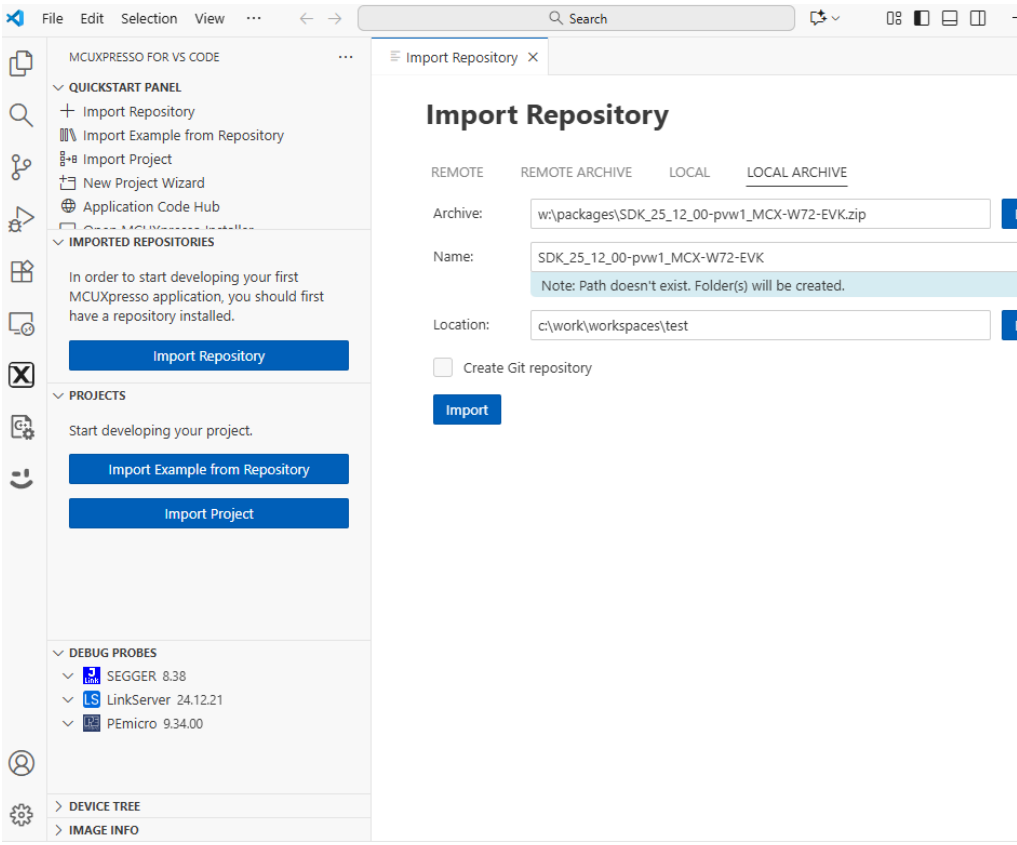
1. Open Visual Studio Code and open the MCUXpresso for Visual Studio Code extension as shown in the figure below.



MCUXpresso for Visual Studio Code extension

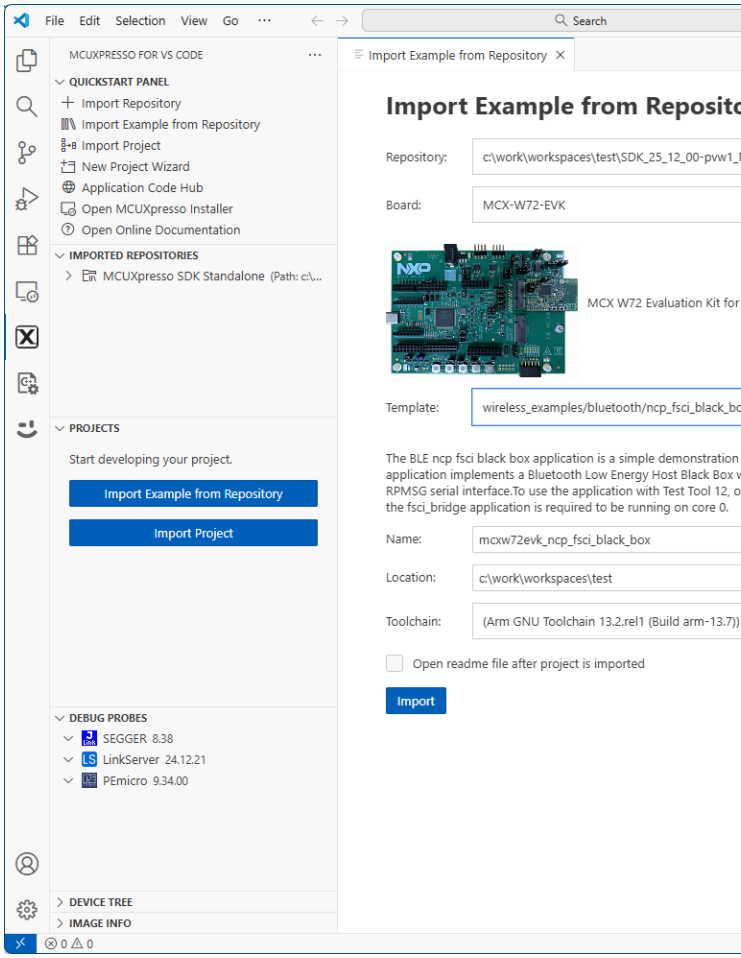
2. Import the SDK package: click “**Import Repository**”. Then, choose the “**Local**” option (if

the SDK is archived use "**Local archive**"), browse to the path of the SDK you want, and click "**Import**".



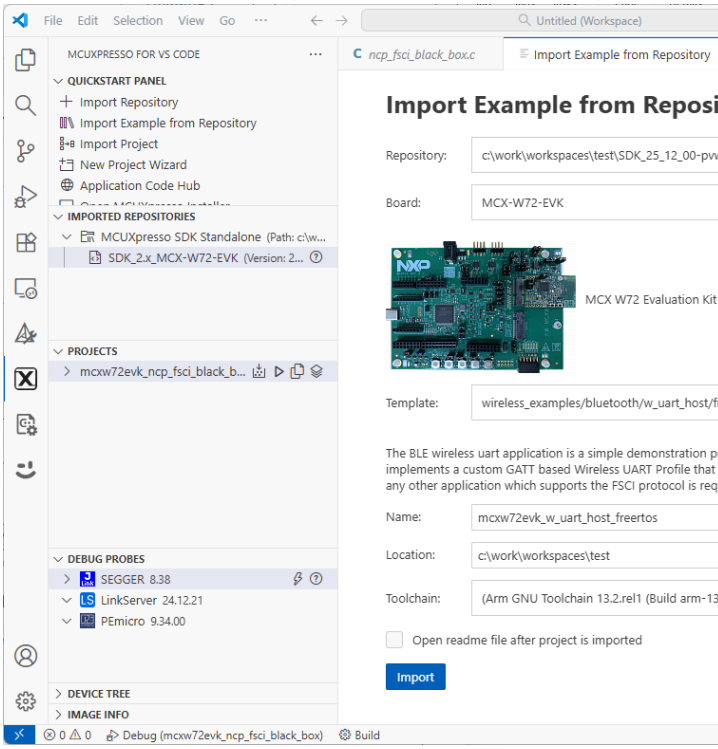
Steps to import the SDK

3. Import NCP(Core1) application: After the SDK is loaded successfully, select the "**Import Example from Repository**" to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click "**Import**".



Steps to import the NCP example application

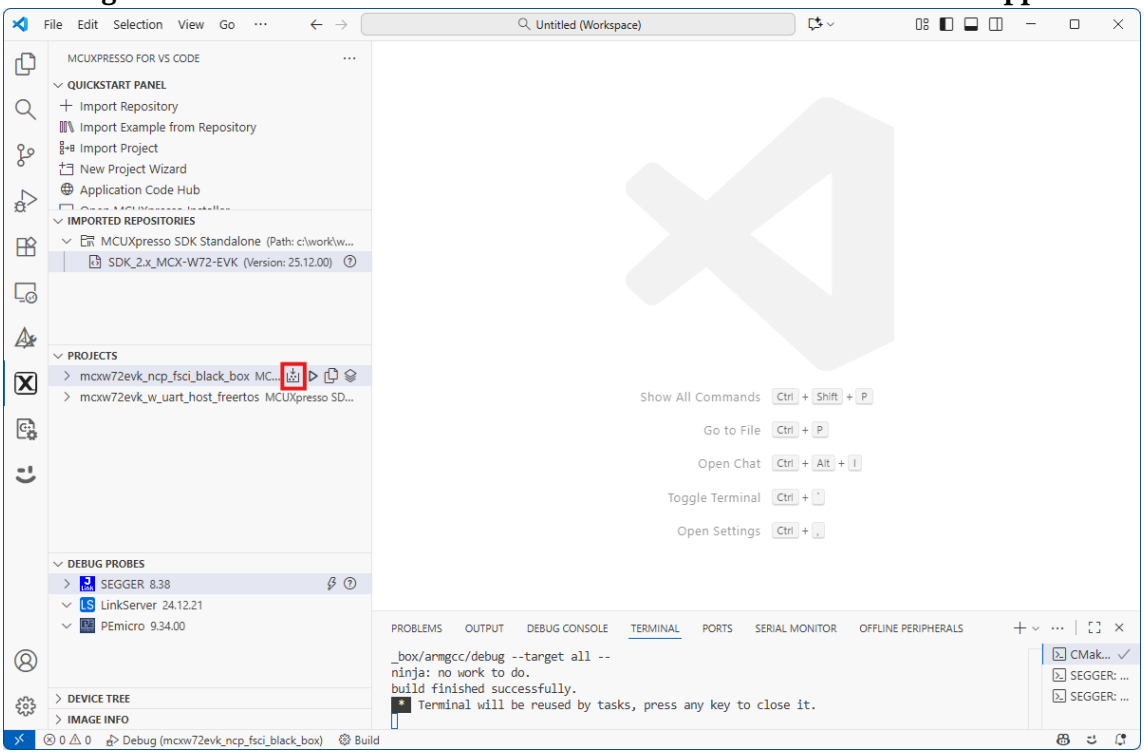
4. Import Core0 application: After the SDK is loaded successfully, select the "**Import Example from Repository**" to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click "**Import**".



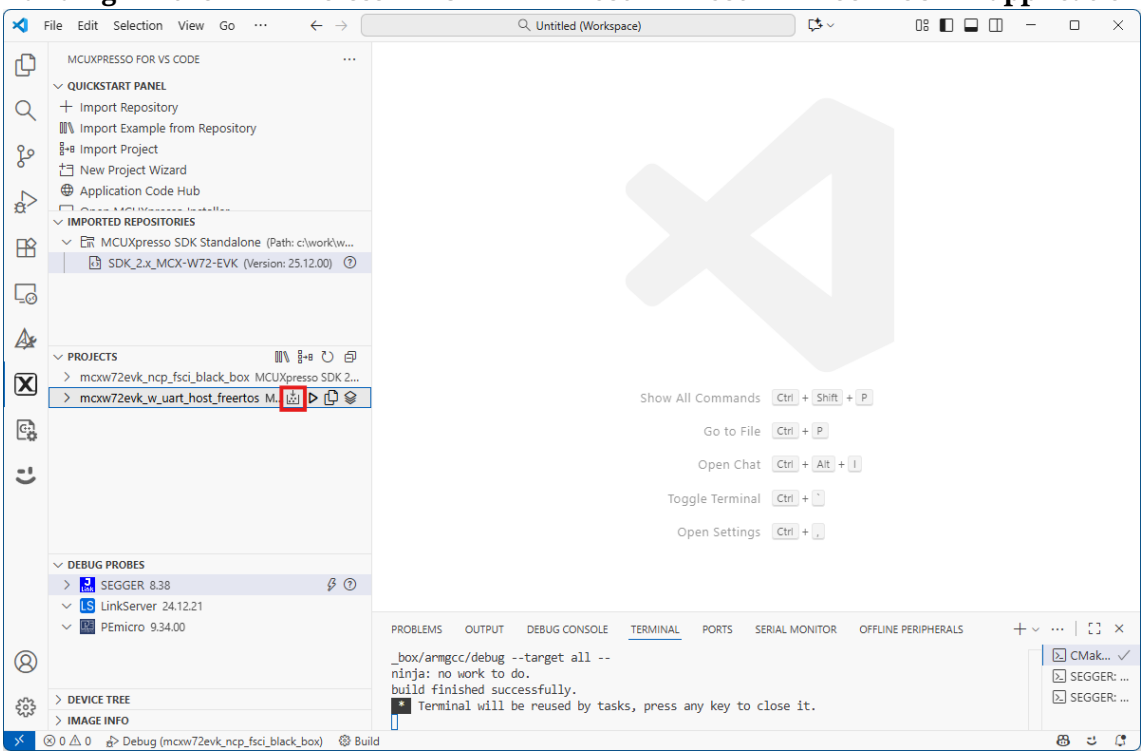
Steps to import the Core0 example application

5. Both Extended NBU applications now appears in the "**Projects**" tab on the left. Build both applications by pressing the "**Build selected**" button on each.

Building the NCP FSCI Black Box ThreadX application

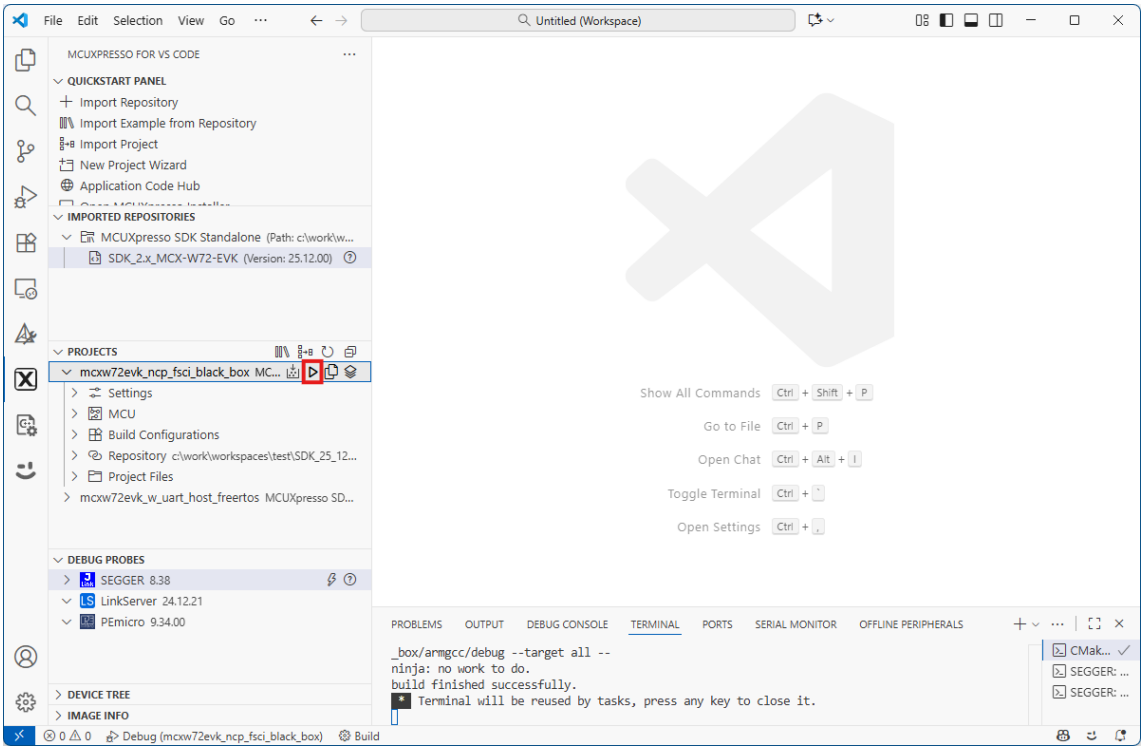


Building the Wireless UART Host Host FreeRTOS application



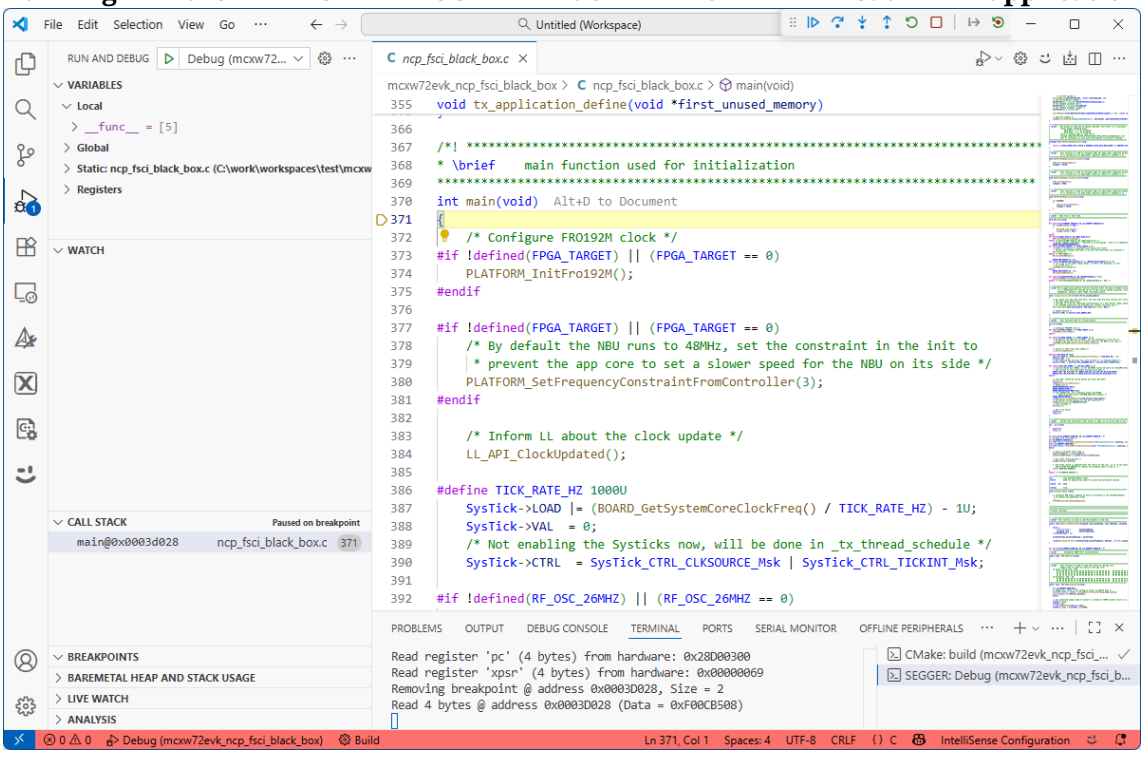
6. After the build is completed successfully, first debug the NBU(Core1) application by clicking the "**Debug**" button.

Debug the NCP FSCI Black Box ThreadX application



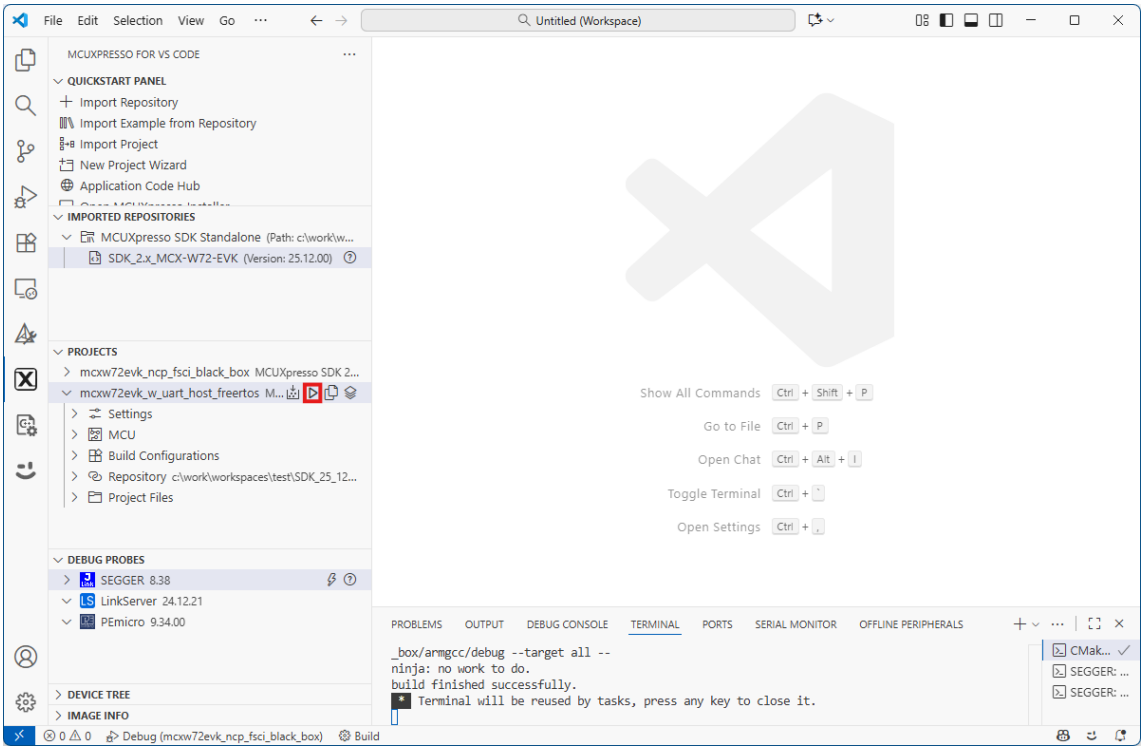
7. Press the run (“**Continue**”) button to run the application on the board.

Running the NCP FSCI Black Box ThreadX application



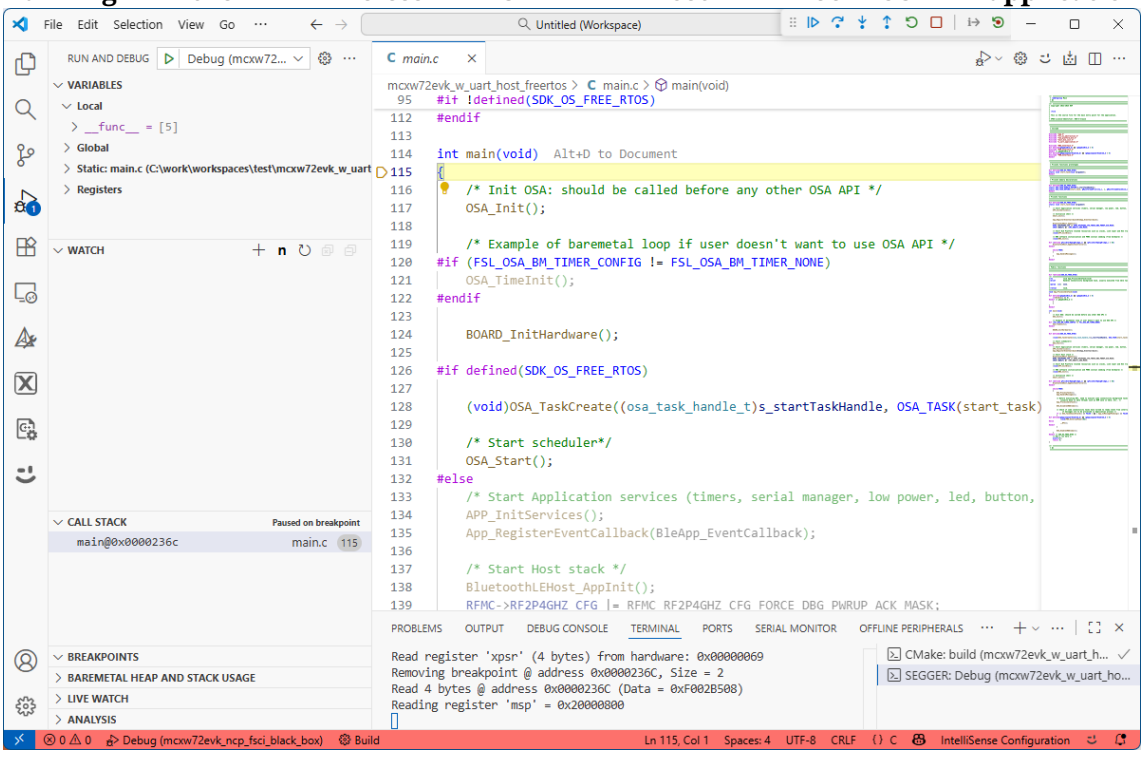
8. After the build is completed successfully and the NBU application was successfully written, debug the Core0 application by clicking the “**Debug**” button.

Debug the Wireless UART Host FreeRTOS application



9. Press the run (“**Continue**”) button to run the application on the board.

Running the Wireless UART Host FreeRTOS application



Parent topic:*Building the binaries*

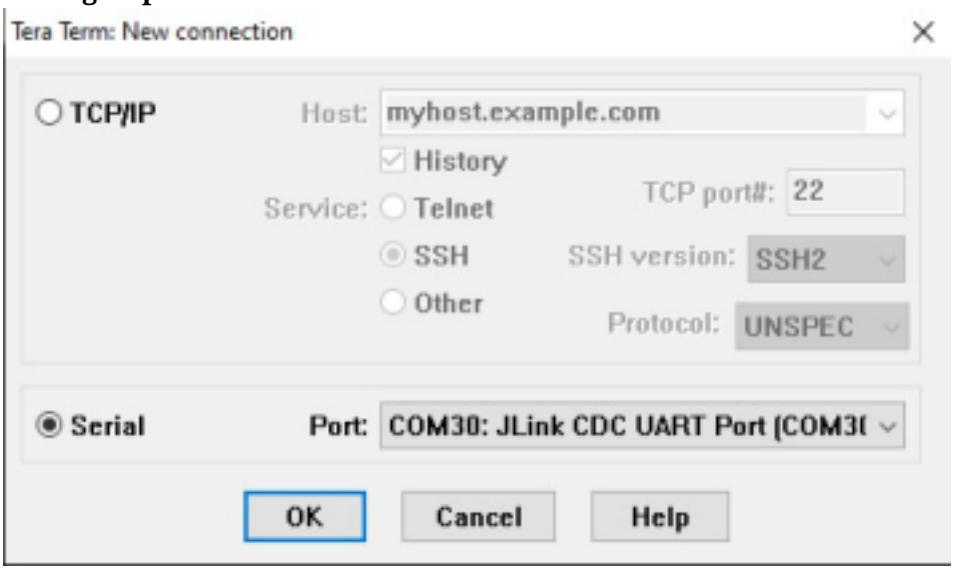
Running the Wireless UART application using NXP IoT Toolbox mobile application To run the Wireless UART application using NXP IoT Toolbox mobile application, follow the steps as listed below:

Note: Before working on these steps, ensure to install the latest version of the NXP IoT Toolbox

mobile application from the application store.

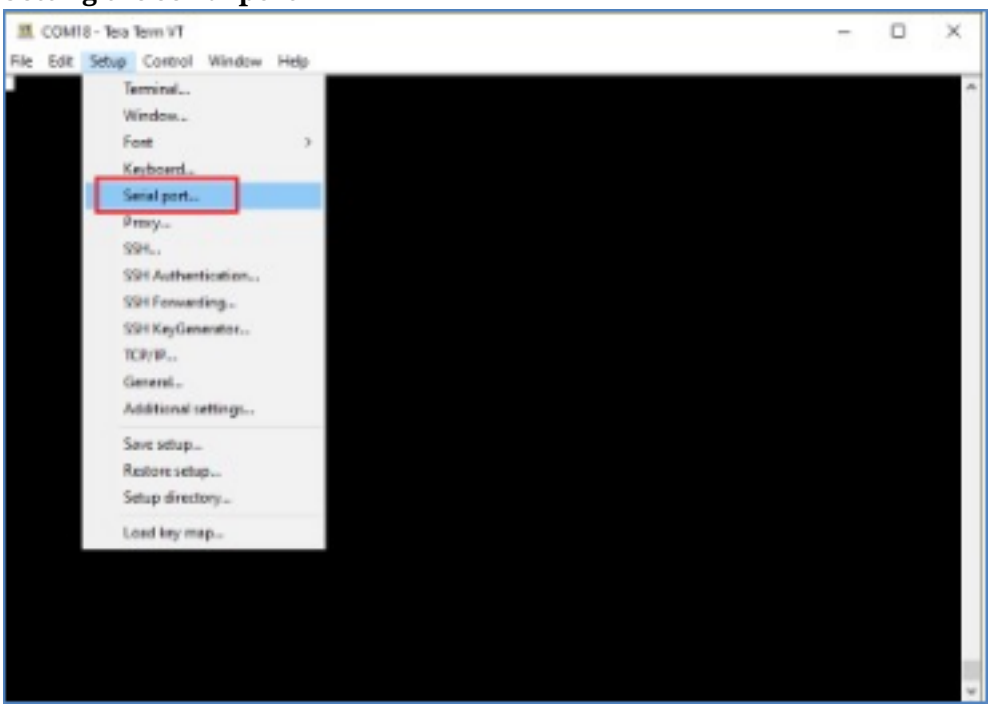
1. Flash the board with the Wireless UART application as previously described.
2. Open Tera Term (or any other Serial Communication software) and choose **Serial**. Then choose the port of your board and click **OK**.

   Setting up a new Tera Term connection for Wireless UART application



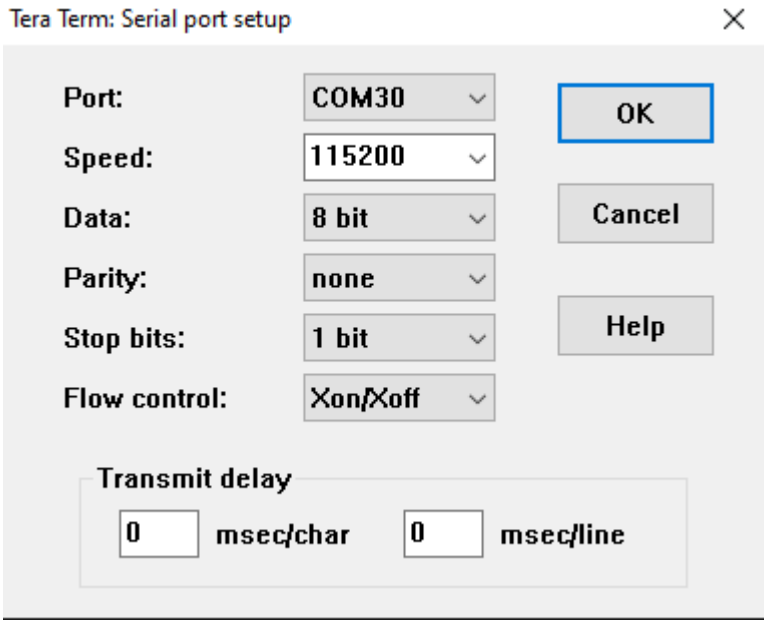
3. Stop running the code on the board. Go to **Setup** >**Serial Port**.

   Setting the serial port



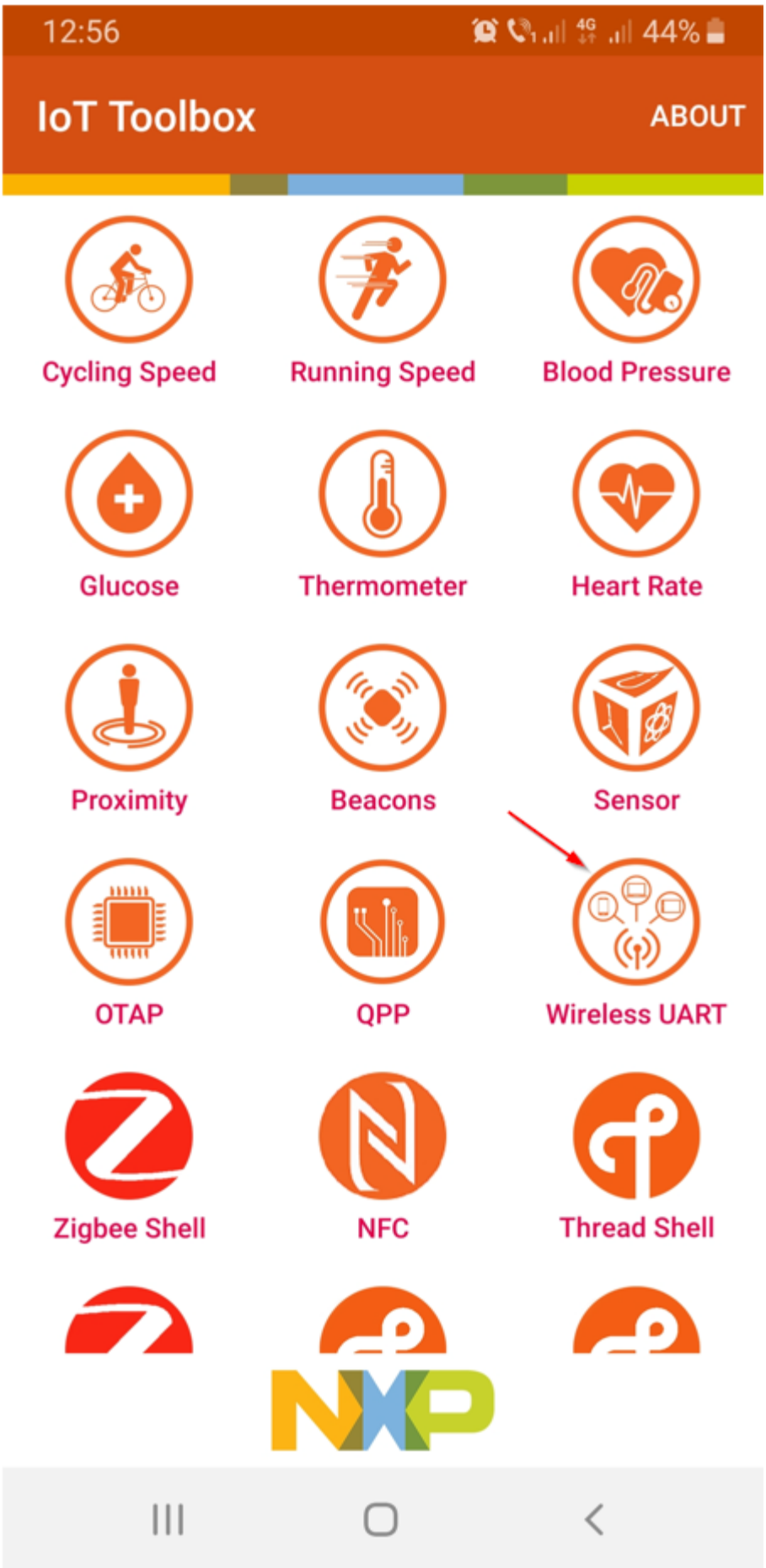
4. Choose the required Speed (baud rate) and Flow Control values. Here, the baud rate should be 115200 and the Flow control mode should be set to **Xon/Xoff**.

   Setting the baud rate



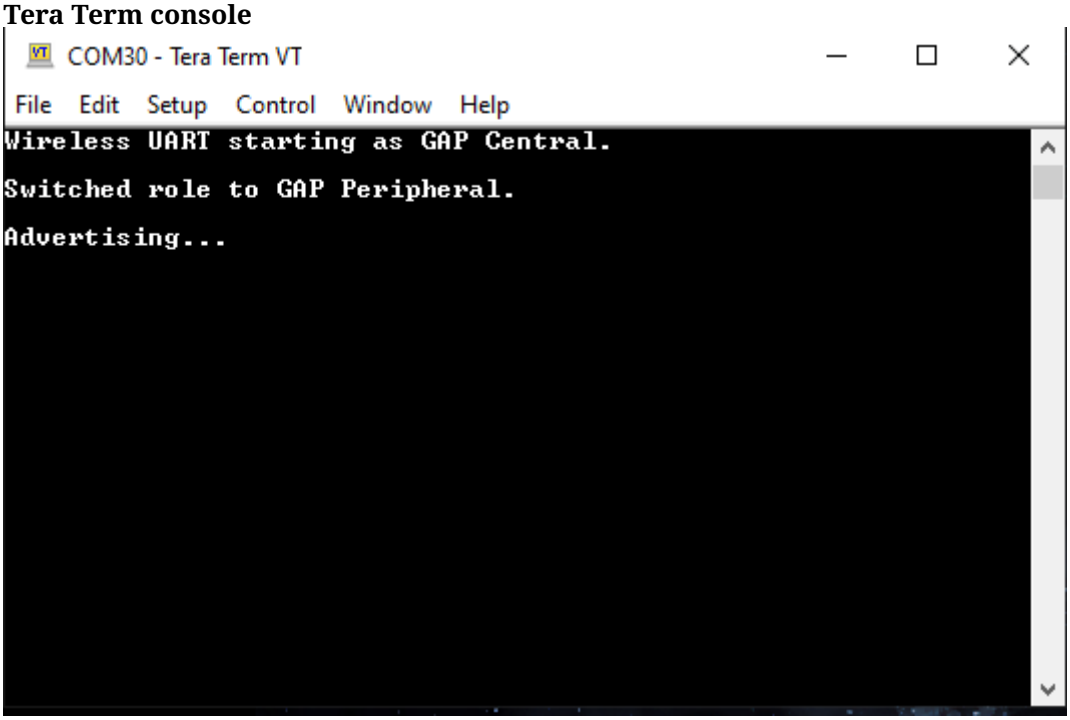
5. Ensure that the Bluetooth device of your phone is enabled.
6. Open the *IoT Toolbox* application and select the ***Wireless UART*** icon.

   NXP IoT Toolbox application

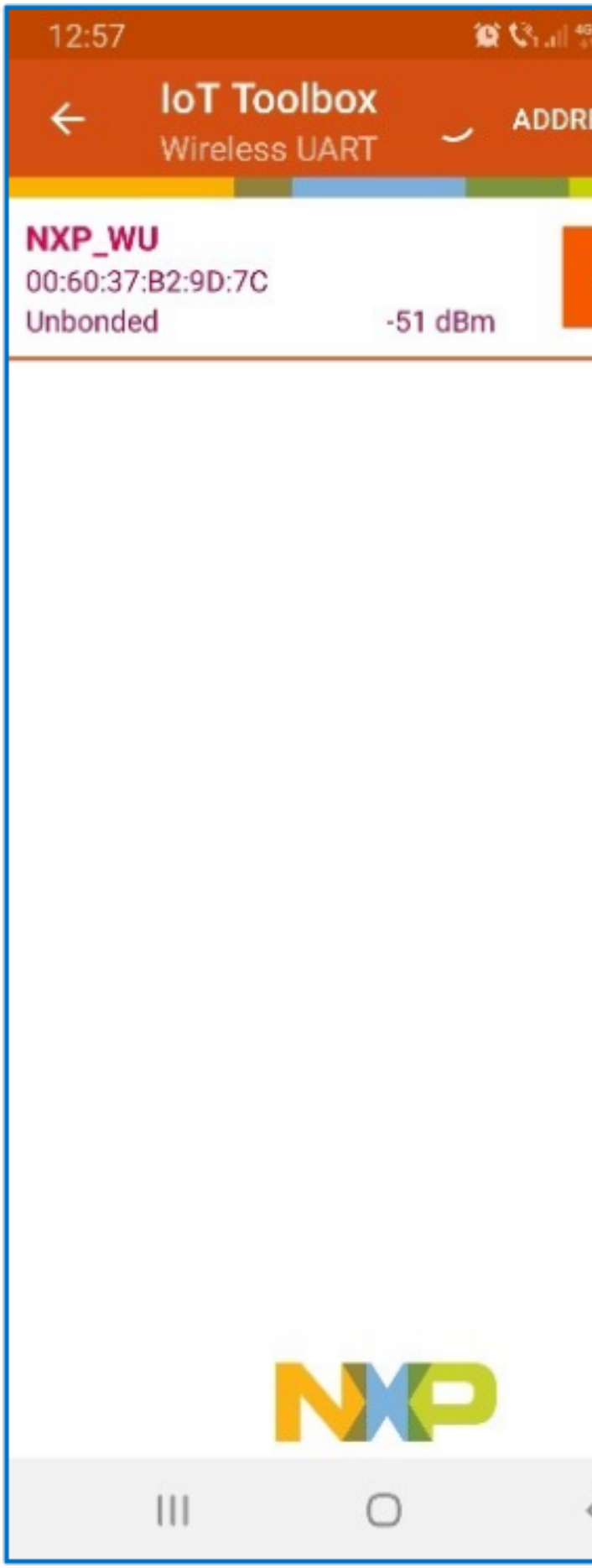

12:56
44%
IoT Toolbox
ABOUT
Cycling Speed
Running Speed
Blood Pressure
Glucose
Thermometer
Heart Rate
Proximity
Beacons
Sensor
OTAP
QPP
Wireless UART
Zigbee Shell
NFC
Thread Shell

7. Now run the application on the board. On this step, the RGB LED blinks white quickly. Press the button **SW3** to switch to Peripheral (Central) mode, then press **SW2** to start Advertising (scanning). The Tera Term terminal shows the message shown in the figure below.

Tera Term console



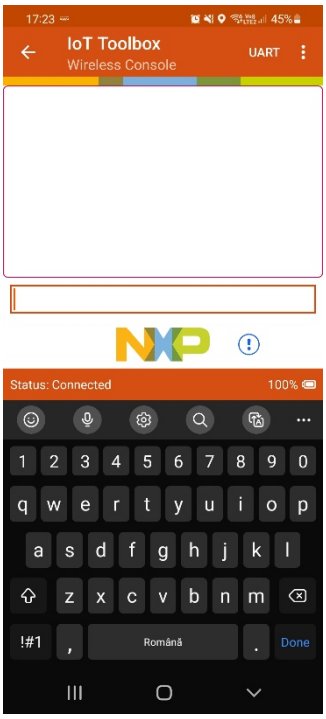
8. The device should become visible for the Wireless UART mobile application.



NXP Wireless UART scanned by IoT toolbox app

9. Select the device that appears in the **Wireless UART** tab to connect to it. After connecting, the mobile application shows the console, as shown below. Users can choose between the Wireless Console and Wireless UART tabs.

   The device should become visible for the Wireless UART mobile application.

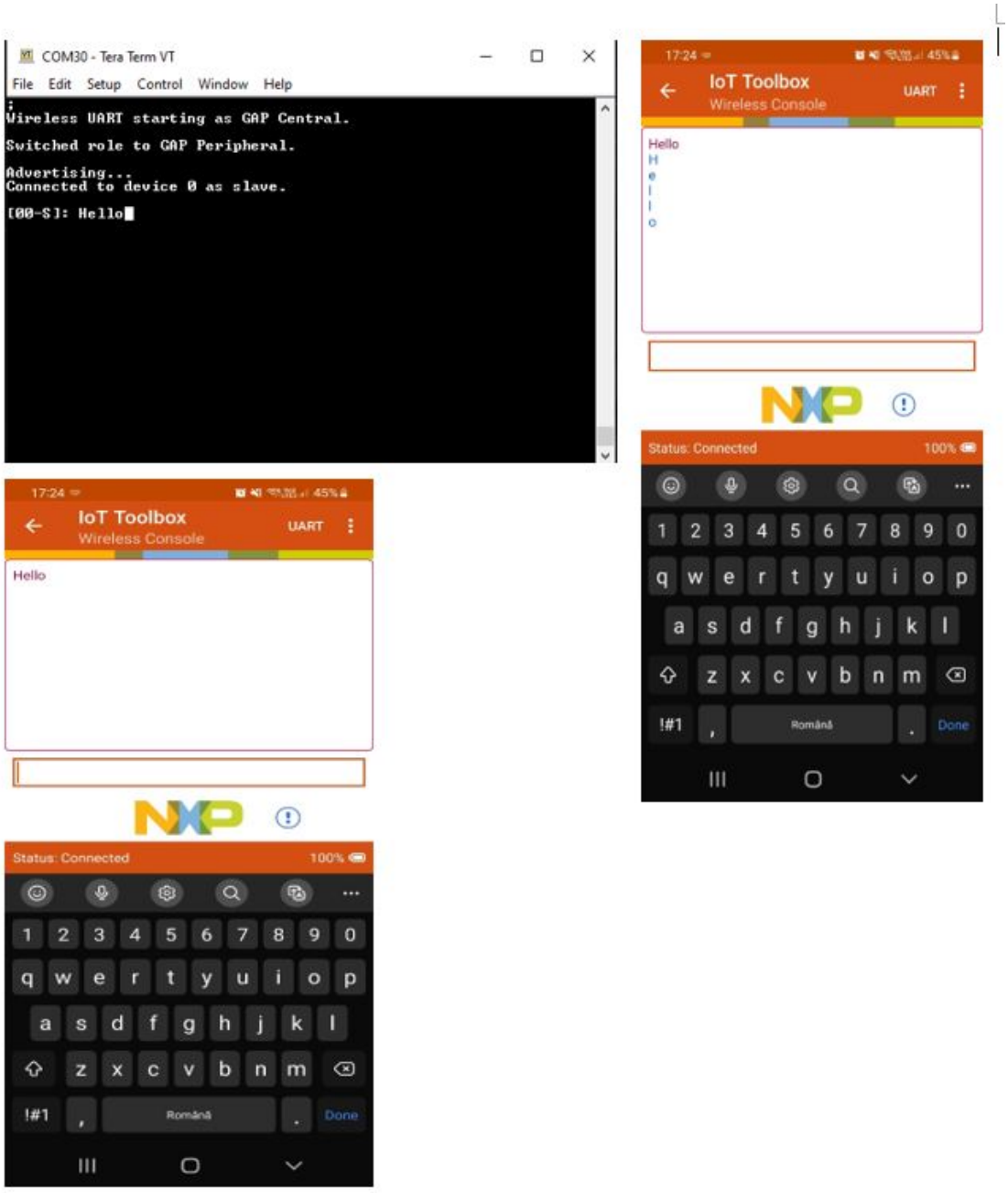


Wireless UART working in Connected mode

10. On the IoT Toolbox Wireless application console, the message is introduced in one line and sent to the peer. If Wireless UART mode is used, communication is done character by character. On Tera Term, the message received is displayed. One character at a time can be sent from Tera Term to the peer. When the mobile application receives the character, it displays it.

    The next figure shows the board acting as a peripheral, connected to a phone using IoT Toolbox in Wireless Console and Wireless UART modes.

    Exchanging messages between the phone and Wireless UART Console



References For more information, refer to the NXP website or contact your local Field Application Engineer (FAE).

Acronyms and abbreviations The following acronyms are used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
EVK	Evaluation Kit
IDE	Integrated Design Environment
ISP	In-system Programming
IoT	Internet of Things
RTOS	Real-time Operating System
SDK	Software Development Kit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy Demo Applications User Guide

Introduction This document describes the Bluetooth Low Energy host stack enablement for NXP development platforms. The document is organized as follows:

- *Bluetooth Low Energy applications* lists the demo applications that can be located in the Software Development Kit (SDK).
- *Hardware configurations* describes the hardware and toolchain requirements.
- *Building and running a Bluetooth LE example application* describes the general requirements for using and testing a Bluetooth Application on a compatible device.
- *Bluetooth LE stack and demo applications* describes the steps and instructions for using the demo applications on your device. It also presents the profiles and services implemented and how to interact with them.
- *References* lists the additional documents that can be referred for more information.
- *Acronyms* lists the acronyms used in this document.

Bluetooth Low Energy applications The Software Development Package provides a Bluetooth Low Energy v5.3-compliant host stack implementation with a set of GATT-based profiles and services implemented on top. To demonstrate the device functionality, the following demo applications are implemented.

1. ANCS Client
2. Beacon Application
3. Bluetooth LE FSCI Black Box
4. EATT Central
5. EATT Peripheral
6. HCI Black Box
7. HID Host
8. HID Device (Mouse)
9. Low-power Temperature Sensor and Collector
10. Low-power Extended Advertising Central and Peripheral
11. OTAP Clients ATT and L2CAP and OTAP Server
12. Wireless UART demo application
13. Bluetooth LE Shell application
14. Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK
15. Wireless UART Host
16. FSCI Bridge
17. NCP FSCI Black Box

Note: Refer to the application notes that are located in the 'documentation' folder.

Hardware configurations

Hardware requirements The NXP Bluetooth LE demo applications run on a selection of the following platforms:

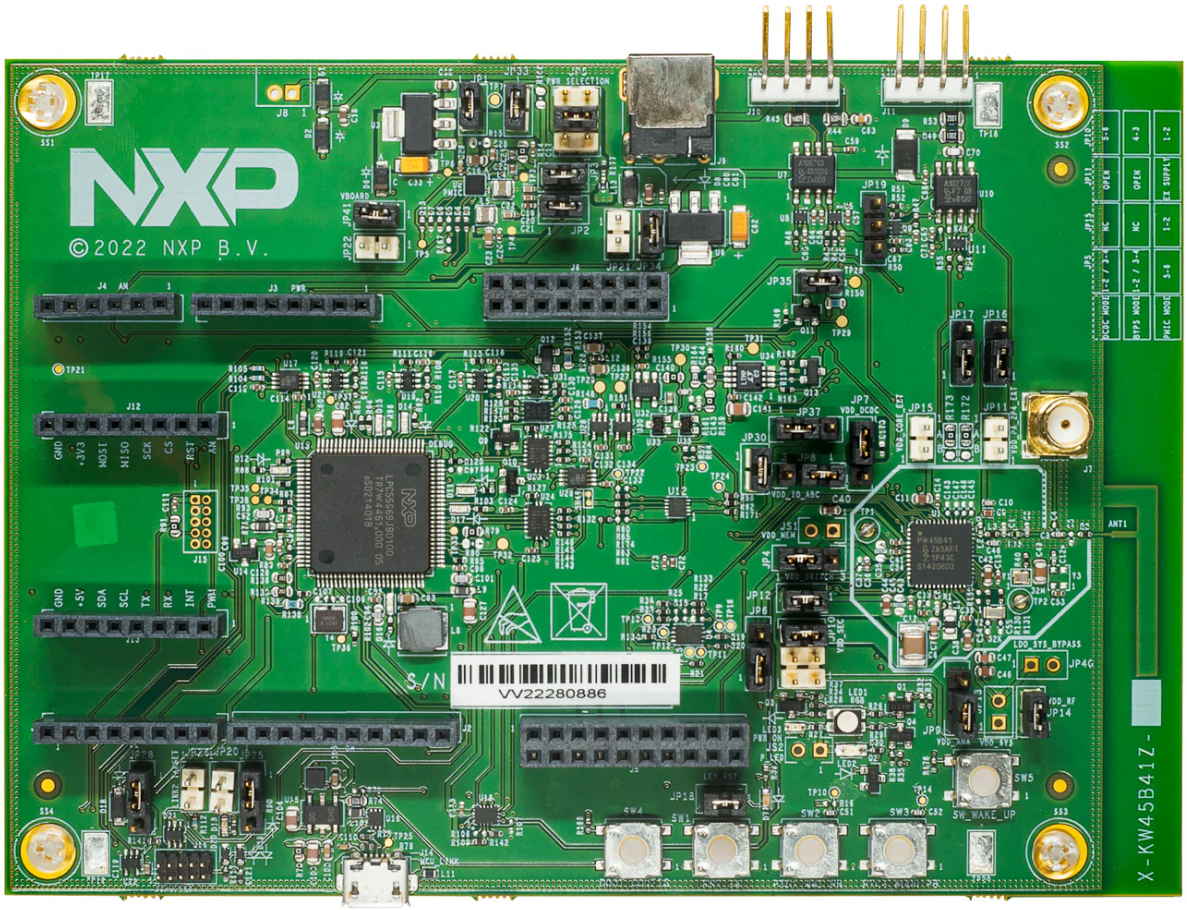
- KW45B41Z-EVK
- KW45B41Z-LOC
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- KW47-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- MCXW72-LOC
- FRDM-MCXW23
- MCXW23-EVK

Parent topic:*Hardware configurations*

Toolchain requirements The Bluetooth Low Energy Stack demo applications were compiled and tested with IAR Embedded Workbench for Arm and MCUXpresso. Users must use one of these tools.

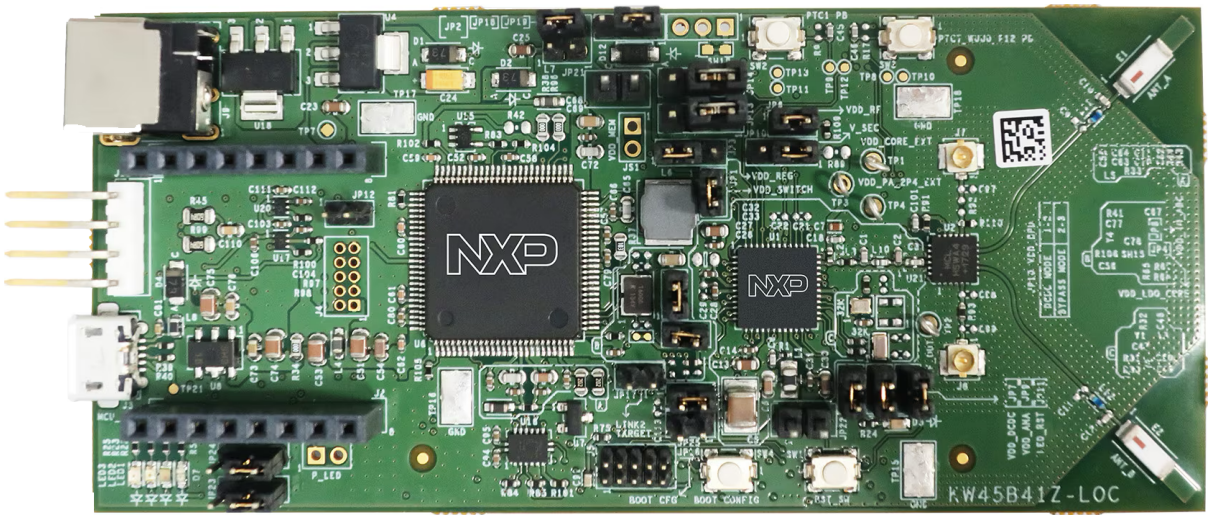
Parent topic:*Hardware configurations*

KW45B41Z-EVK platform The figure below displays the top view of the KW45B41Z-EVK board.



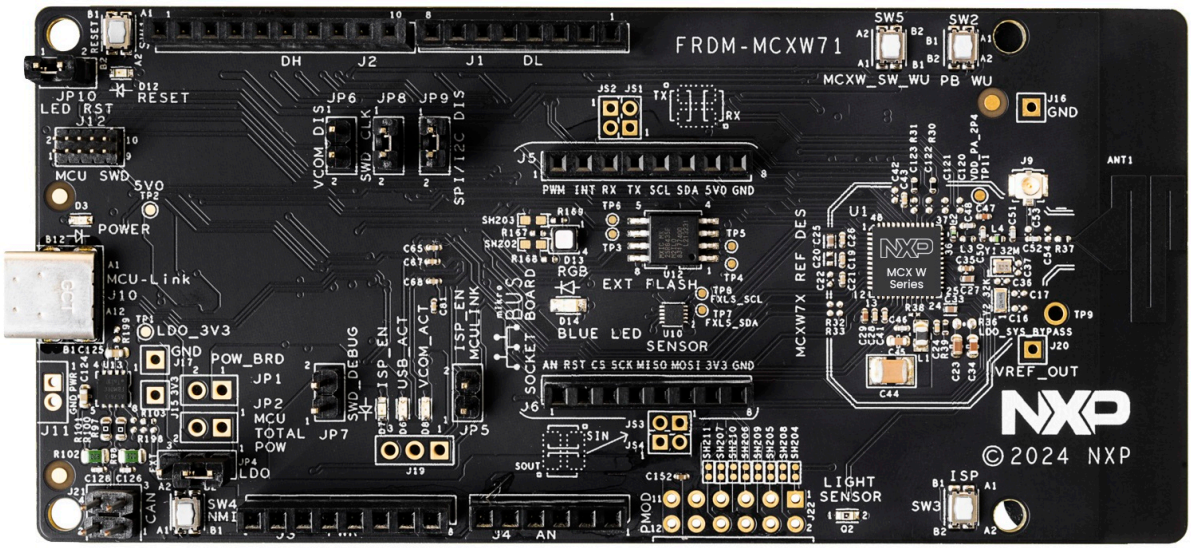
Parent topic:*Hardware configurations*

KW45B41Z-LOC platform The figure below displays the top view of the KW45B41Z-LOC board.



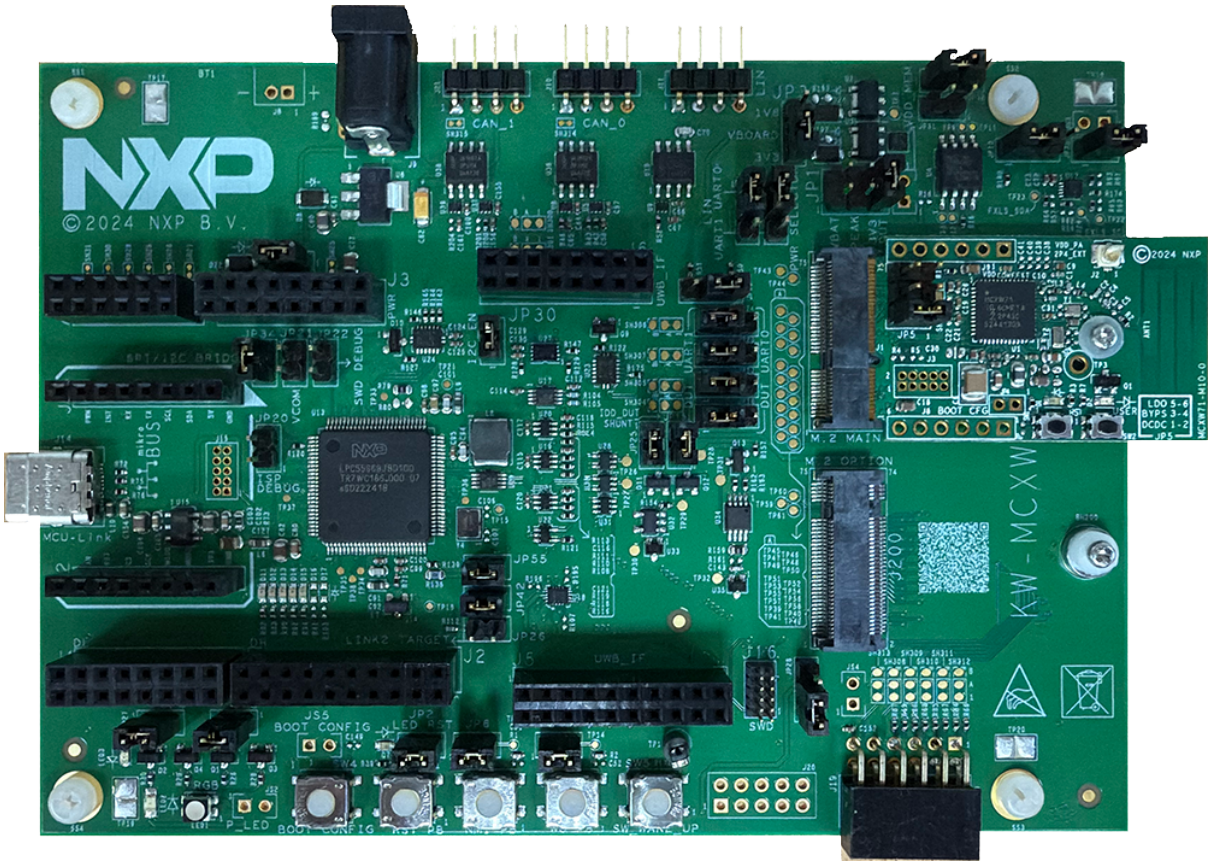
Parent topic:*Hardware configurations*

FRDM-MCXW71 platform The figure below displays the top view of the FRDM-MCXW71 board.



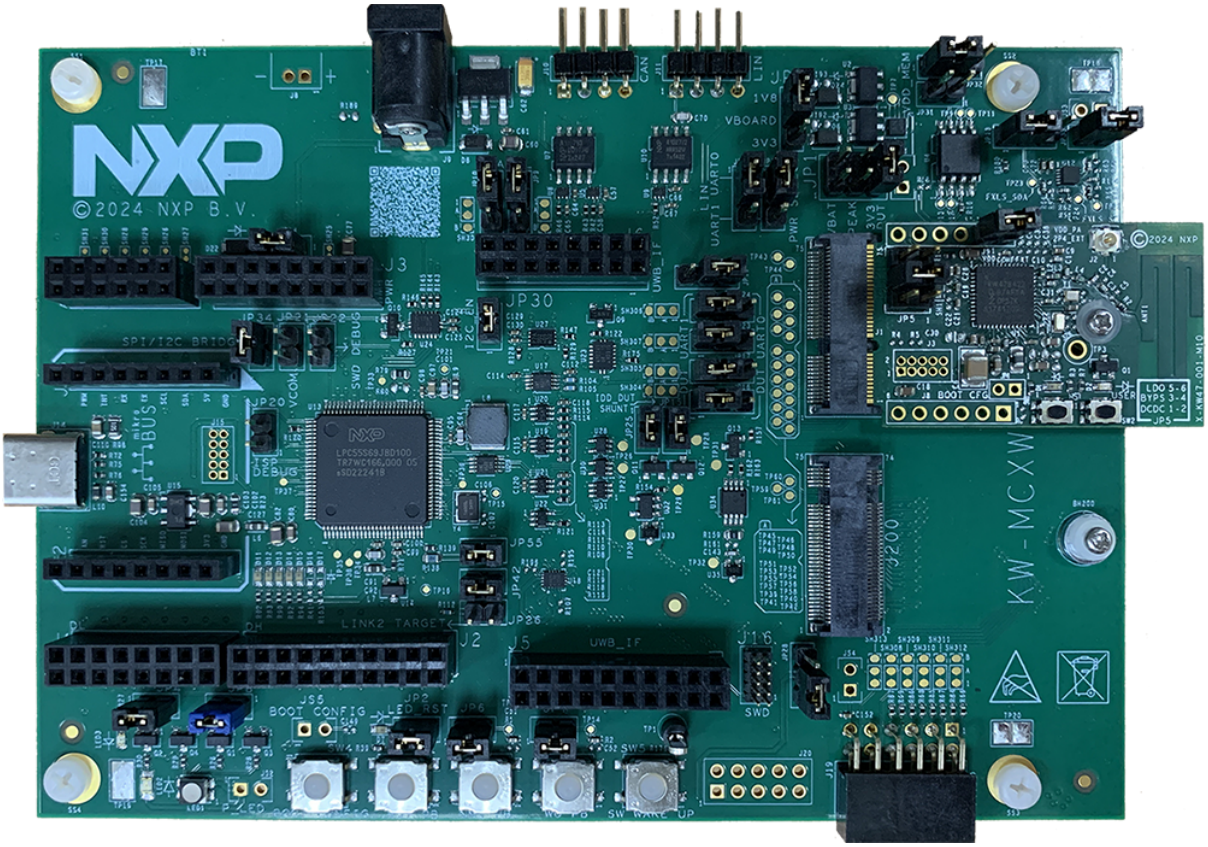
Parent topic:*Hardware configurations*

MCX-W71-EVK platform The figure below displays the top view of the MCX-W71-EVK board.



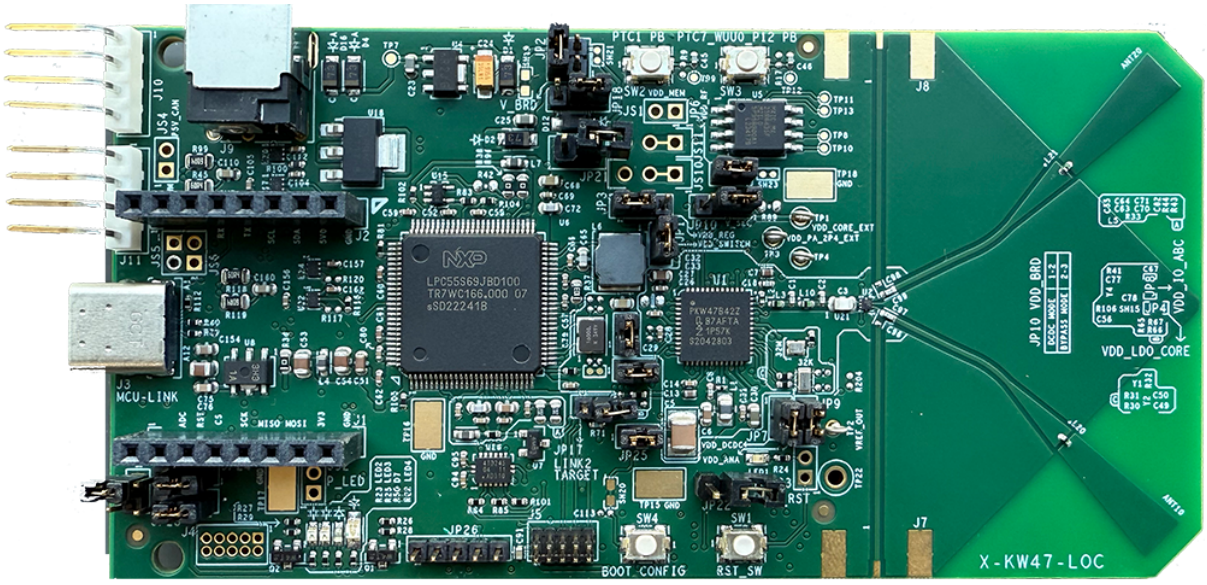
Parent topic:*Hardware configurations*

KW47-EVK platform The figure below displays the top view of the KW47-EVK board.



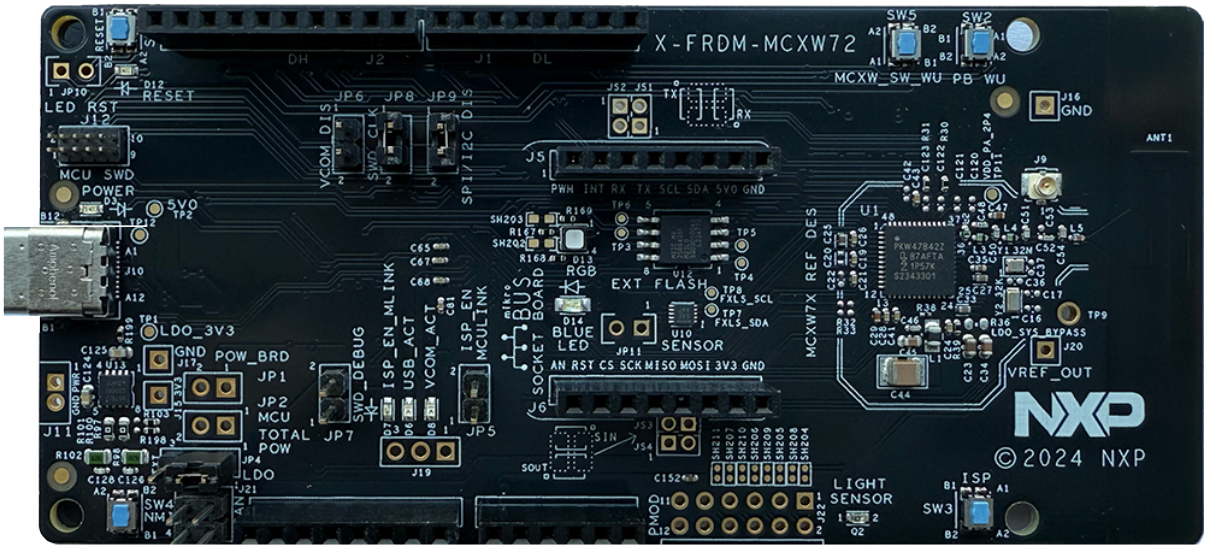
Parent topic:*Hardware configurations*

KW47-LOC platform The figure below displays the top view of the KW47-LOC board.



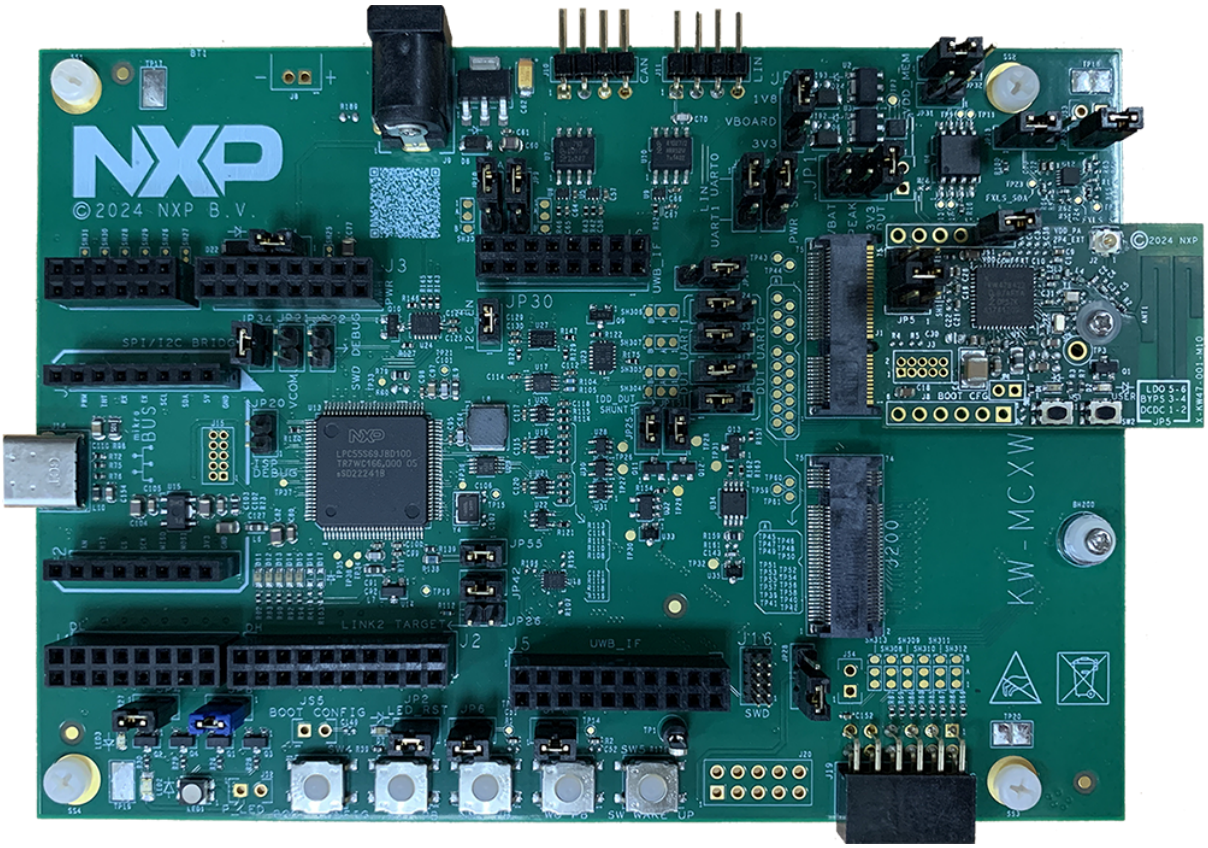
Parent topic:*Hardware configurations*

FRDM-MCXW72 platform The figure below displays the top view of the FRDM-MCXW72 board.



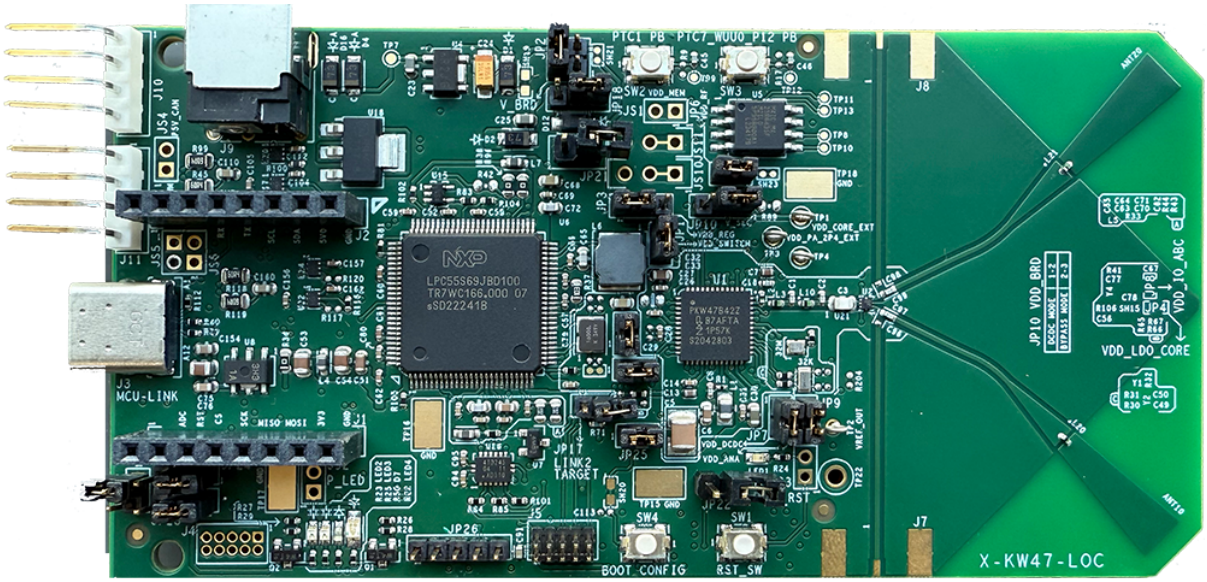
Parent topic:*Hardware configurations*

MCX-W72-EVK platform The figure below displays the top view of the MCX-W72-EVK board.



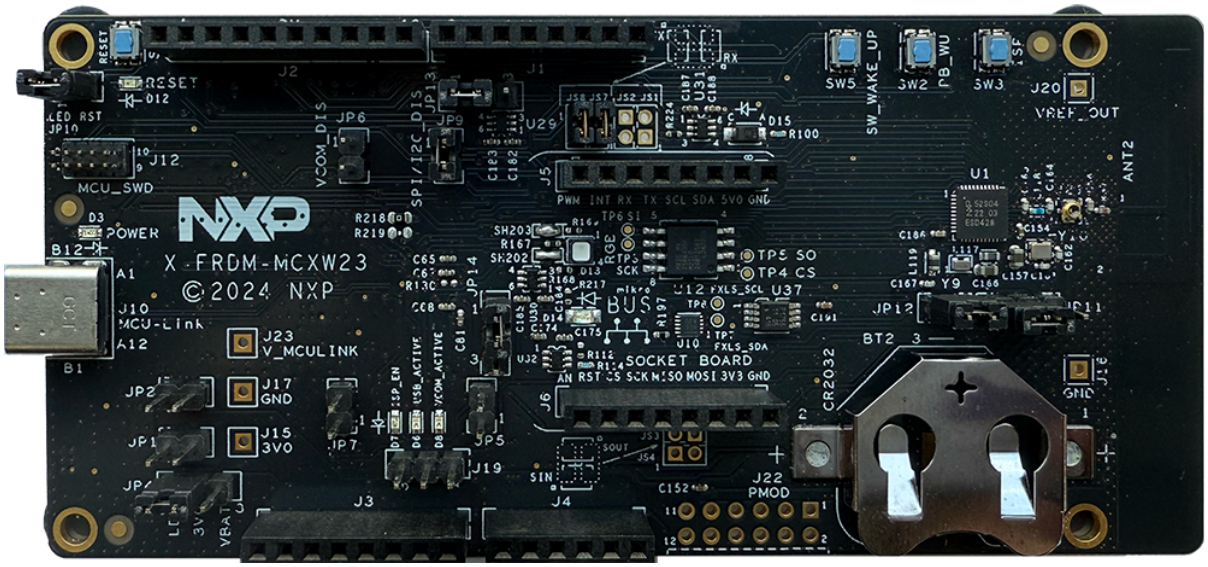
Parent topic:*Hardware configurations*

MCXW72-LOC platform The figure below displays the top view of the MCXW72-LOC board.



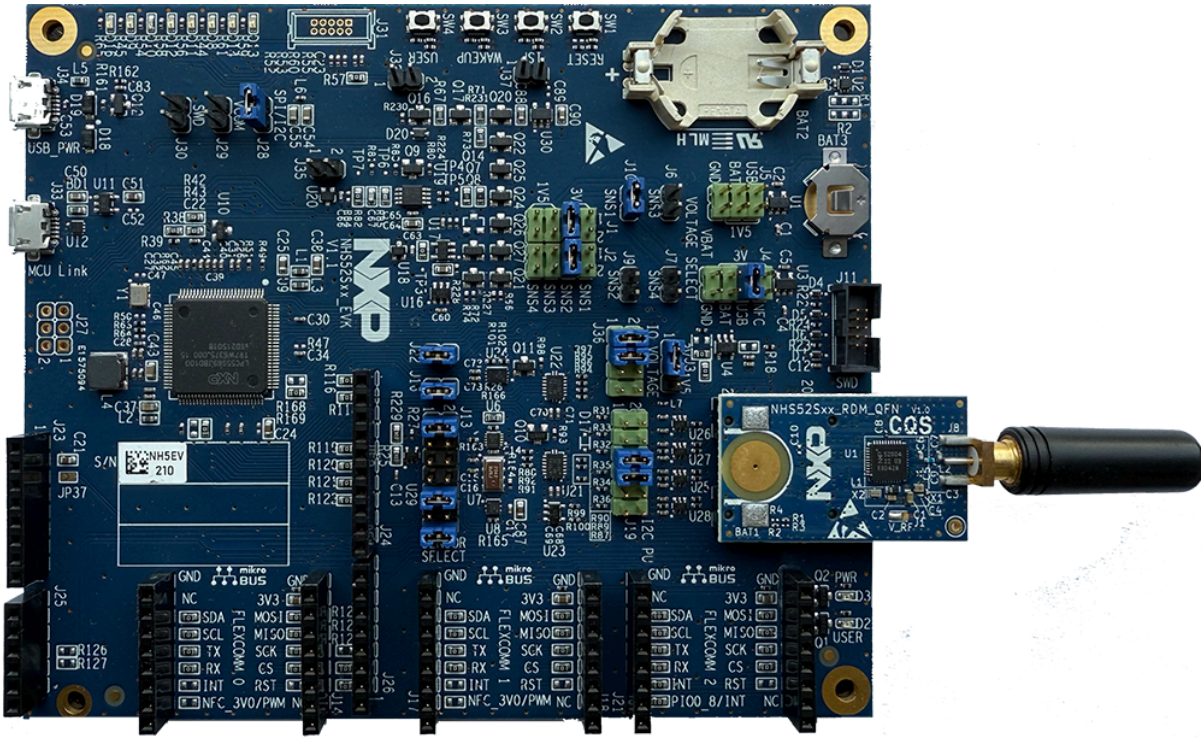
Parent topic:*Hardware configurations*

FRDM-MCXW23 platform The figure below displays the top view of the FRDM-MCXW23 board.



Parent topic:*Hardware configurations*

MCXW23-EVK platform The figure below displays the top view of the MCXW23-EVK board.



Parent topic:*Hardware configurations*

Building and running a Bluetooth LE example application This section presents the general requirements for using and testing a demo application. To open, build, and run any example application on a specific board, refer to the Bluetooth Low Energy Quick Start Guide document for the corresponding board.

User interface The demo applications that implement the Battery Service expose the current battery level, as measured on the board, through the Battery Level characteristic. The value represents a percentage between 0 and 100. The value can be read from the device from a connected GATT client.

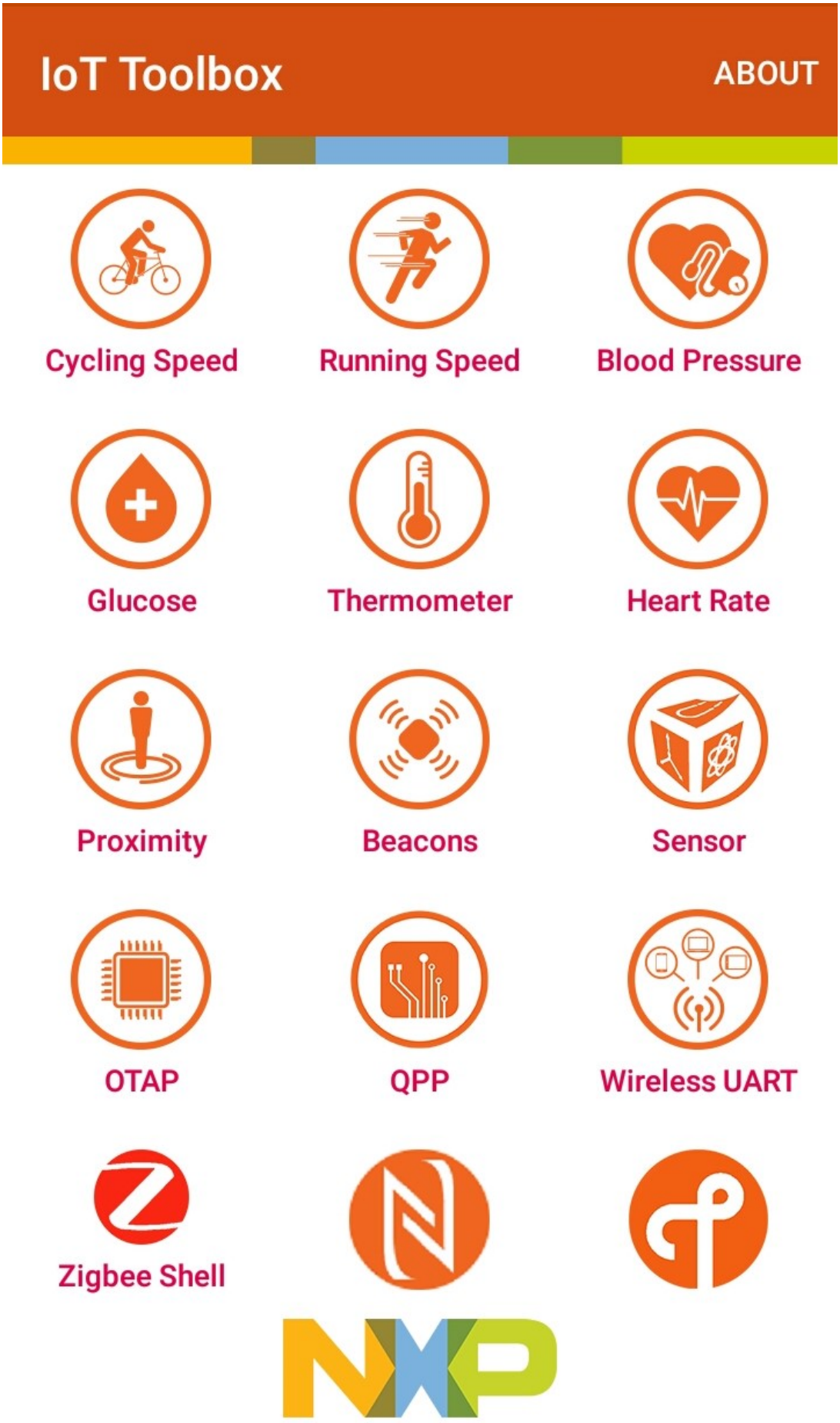
The demo applications that implement the Device Information Service display various information regarding the current software, hardware, and firmware revisions. These values are used as an example and application developers can modify them when developing their product. The values can be read from the device through a connected GATT client.

Parent topic:*Building and running a Bluetooth LE example application*

Security The examples that enable pairing always generate a default passkey of 999999 that must be entered on the Central device, which is usually a smartphone or tablet.

Parent topic:*Building and running a Bluetooth LE example application*

Testing devices To demonstrate the profile functionality, most of the scenarios require one of the supported platforms and a Bluetooth Low Energy capable central device. The device is usually a smartphone or a tablet that runs a compatible Bluetooth LE application. The figure below shows the **IoT Toolbox** UI.


IoT Toolbox
ABOUT
Cycling Speed
Running Speed
Blood Pressure
Glucose
Thermometer
Heart Rate
Proximity
Beacons
Sensor
OTAP
QPP
Wireless UART
Zigbee Shell
NXP

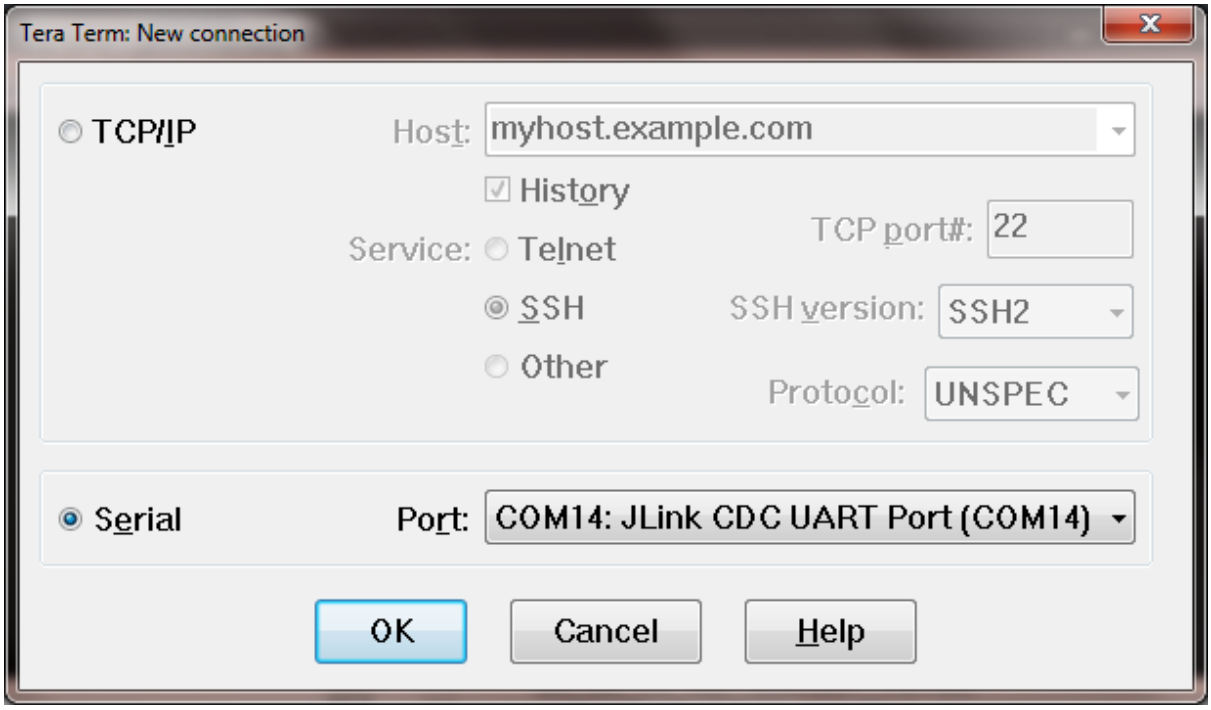
IOT toolbox

The recommended application is the IoT Toolbox, which can be installed on Apple iOS or Android OS handheld devices that support Bluetooth Low Energy. The application can be found on Apple Playstore or on Google Play.

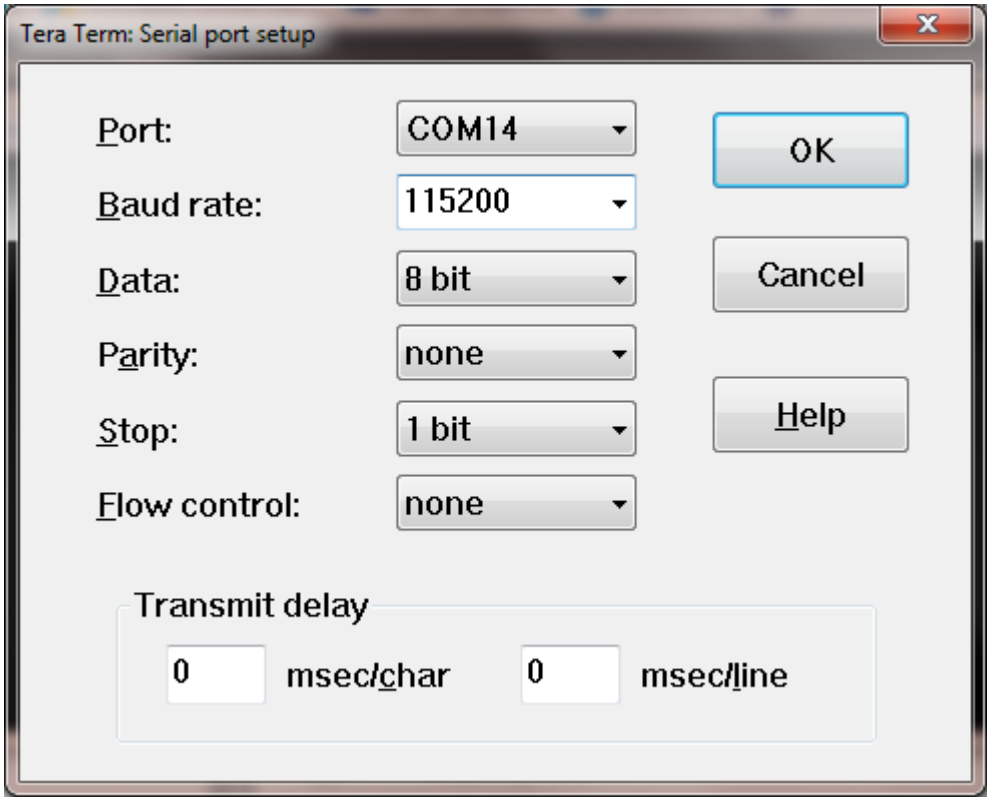
Other demos can be run by using two platforms, one for the peripheral and one for the central role. A few examples of such demos are listed below:

- Low-Power Temperature Sensor and Collector
- Wireless UART
- OTAP Client and Server
- HID Host and Device
- Extended Advertising Central and Peripheral
- EATT Central and Peripheral

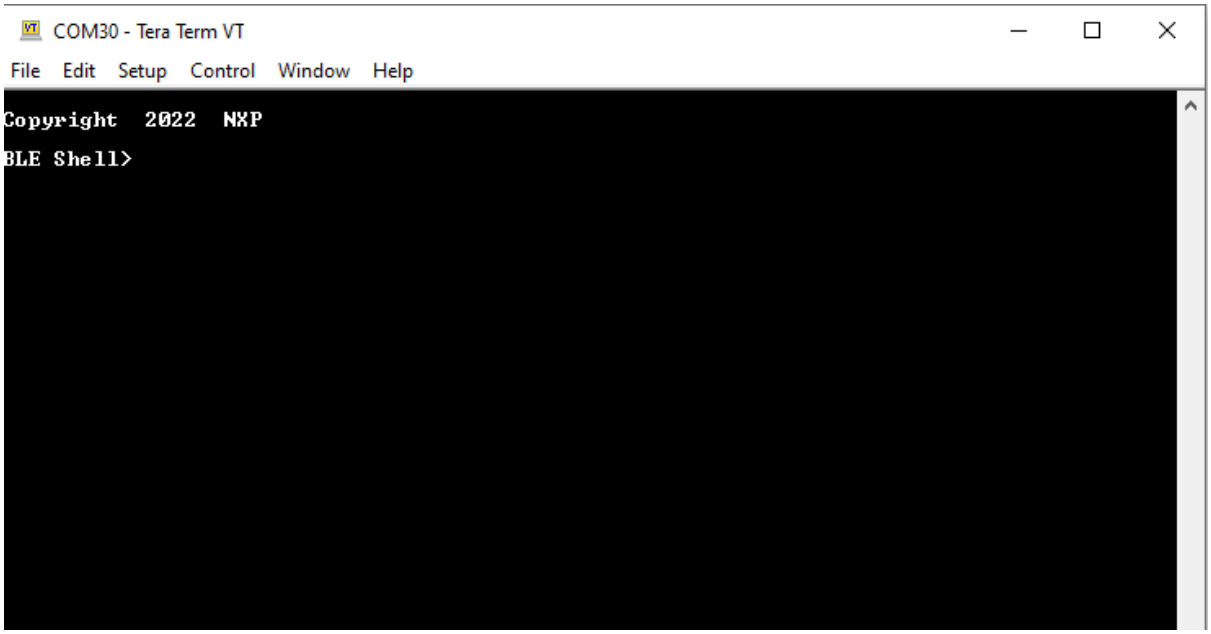
To provide feedback and more interaction, some examples use a shell console via the virtual COM port. To access the device, open a serial port terminal and as shown in the figure below. For this example, Tera Term VT and a KW45B41Z-EVK or a FRDM-MCXW71 board can be used. The communication parameters are 115200 and 8N1.



Connect it to the platform with parameters as shown in the figure below .



The start screen is displayed after the board is reset as shown in the figure below .



Parent topic:*Building and running a Bluetooth LE example application*

Time client devices Some applications implement the Current Time Service. To enable this feature, define the gAppUseTimeService_d parameter in the app_preinclude.h file as 1. If the Time Client is enabled, the device must synchronize with a Time Server to update its internal date/time to the current date/time. If you connect the device to the phone, the Time Client synchronizes with the phone (pairing and bonding must be active).

Parent topic:*Building and running a Bluetooth LE example application*

Bluetooth LE stack and demo applications

ANCS/AMS client (ancs_c) This section describes the implemented profiles, services, user interactions, and testing methods for the ANCS and AMS Client application.

Implemented profile and services The ANCS/AMS Client application implements both an ANCS and an AMS Client for the custom ANCS Service and AMS Service available on iOS mobile devices.

Check the documentation available on the iOS website for details about the ANCS or AMS services, their characteristics, and supported features.

The demo application acts as a GAP Peripheral that advertises a service solicitation for the custom ANCS Service, followed by a solicitation to the AMS Service. It also acts as a GATT Client once connected to a device that offers the ANCS/AMS Service. The application offers some services such as the role of GATT Server.

Once connected to a mobile device offering the ANCS/AMS Service, the application displays information about ANCS Notifications received from that device. This information is followed by the AMS track information (Artist, Album, Title, Duration in seconds). The application also displays the possible remote commands that the device state allows (such as Play, Pause, VolumeUp, VolumeDown). The notifications are received via ATT Notifications, for which the ANCS Client must register on the peer ANCS Server. The same must be done for the AMS server. It initially configures the information that it wants to be notified about. The application also retrieves and displays additional information about the received ANCS notifications. For this purpose, it writes commands to specified characteristics on the ANCS/AMS Server and receives responses via ATT Notifications from other characteristics. All information is displayed to the user using a shell available over a serial communications interface.

Accessing the ANCS Service and AMS Service requires Bluetooth LE security to be enabled.

Parent topic:ANCS/AMS client (ancs_c)

Supported platforms The ANCS/AMS Client application is supported on the following platform:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

Parent topic:ANCS/AMS client (ancs_c)

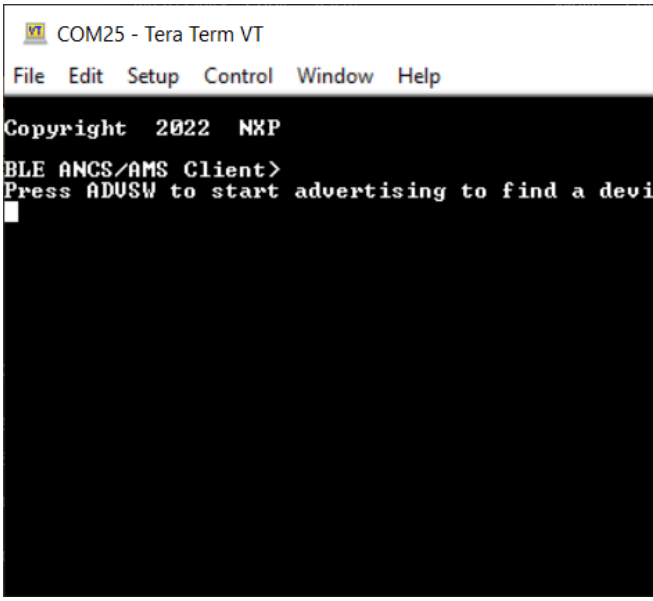
User interface After flashing the board, the device is in idle mode (all LEDs flashing). To start advertising, press the **ADVSW** button. When in GAP Discoverable Mode, **CONNLED** is flashing. When the ANCS/AMS Server (Gap Central) connects to the ANCS/AMS Client (GAP Peripheral), **CONNLED** turns solid. To disconnect, hold the **ADVSW** for 2-3 seconds. The ANCS/AMS Client then re-enters the advertising state.

For displaying operating information and ANCS Notifications (AMS information and commands), the demo application uses a shell exposed via a serial communication interface.

Parent topic:ANCS/AMS client (ancs_c)

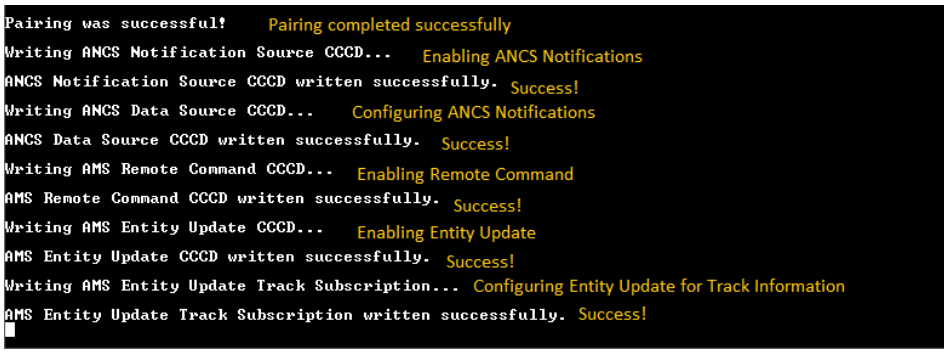
Usage The ANCS/AMS Client demo application is designed to work with a peer mobile device that exposes the ANCS and AMS service. Also, a serial terminal application is required for displaying ANCS Notifications information and AMS commands and information.

1. Open a serial terminal application on the PC and connect it to the serial port corresponding to the board on which the ANCS/AMS Client runs. See the details in Testing devices, "User Interface". A start screen is displayed immediately after the board is reset. All LEDs must be flashing as shown in the figure below.



Start screen for ANCS/AMS Client demo application

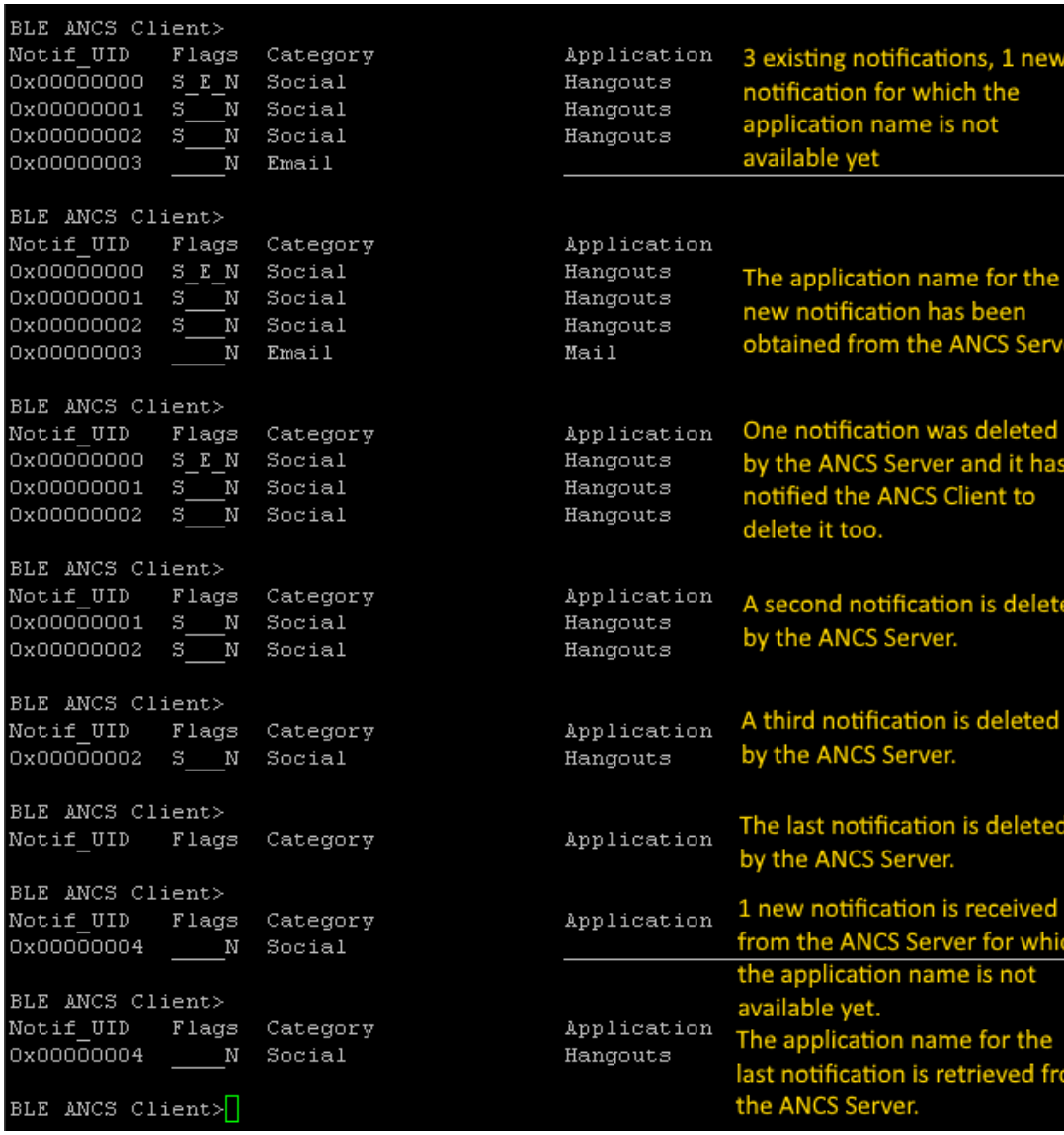
2. Press the **ADVSW** button to start advertising. This instruction is also displayed in the serial terminal as shown in the figure above.
3. The peer device starts scanning for Bluetooth LE devices and connects to the ANCS/AMS Client device that is advertising.
4. Once connected to a peer, the application looks for the ANCS Service and AMS Service and their characteristics. If they are found, the ANCS Client tries to register for receiving notifications and AMS for the tracking data and commands. See Figure.
5. If any security-related ATT errors are encountered, then the application automatically performs Pairing and Bonding and retries the failed ATT operations. Depending on the negotiated pairing method, user interaction might be needed to complete the Pairing. Follow the onscreen instructions provided by both the ANCS/AMS Client and the mobile device. If the ANCS/AMS Client generates a passkey, then the default 999999 passkey is used. See the figure below that shows a 'Pairing is successful' message.



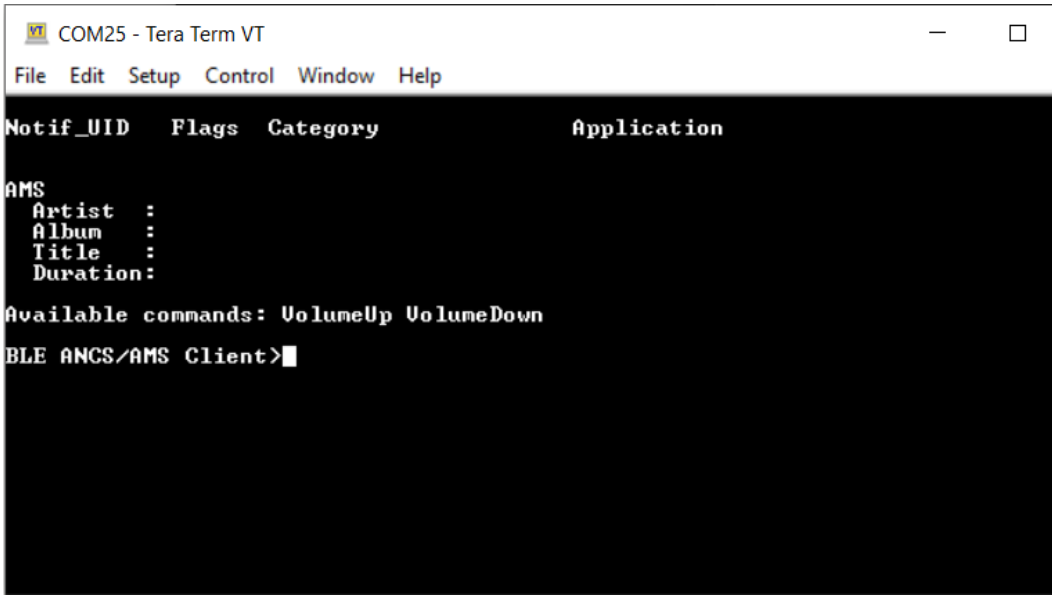
Pairing is successful message

6. After bidirectional communication is established via GATT, the ANCS/AMS Client starts displaying ANCS Notifications information as shown in the figure below.

ANCS Notifications

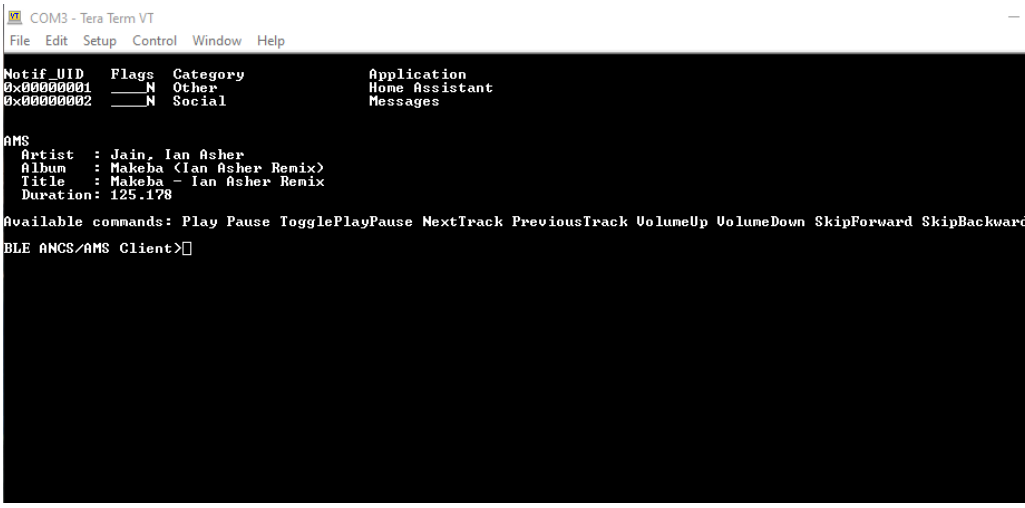


- The combination of the two services (ANCS and AMS) is shown in the two figures below.
- If no media is playing and no player is active, the serial interface looks as shown in the figure below.



AMS no active player

- When media is playing, a player is active and the state must look similar to the figure below.



AMS with player active

Parent topic:ANCS/AMS client (ancs_c)

Parent topic:*Bluetooth LE stack and demo applications*

Beacon This section presents the user interactions and testing methods for the Beacon application.

Advertising data The beacons are non-connectable advertising packets that are sent on the three advertising channels. The latter contains the following fields.

- Company Identifier (2 bytes): 0x0025 (NXP ID as defined by the Bluetooth SIG).
- Beacon Identifier (1 byte): 0xBC (Allows identifying an NXP Beacon alongside with Company Identifier).
- UUID (16 bytes): Beacon sensor unique identifier.
- A (2 bytes): Beacon application data.
- B (2 bytes): Beacon application data.
- C (2 bytes): Beacon application data.

- RSSI at 1m (1 byte): Allows distance-based applications.

By default, the UUID value is a random value based on the unique identifier of the board.

Parent topic: Beacon

Supported platforms The following platforms support the Beacon application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

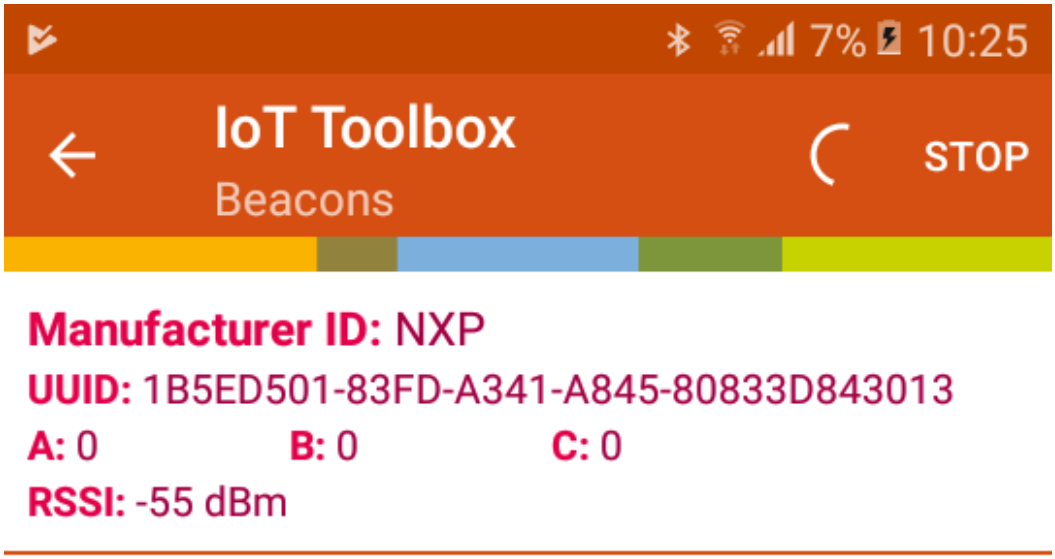
Parent topic: Beacon

User interface After flashing the beacon, the sensor is put in deep sleep (all LEDs are off). To flash the board in case the beacon is put in Deep-sleep mode, press the **ADVSW** or **RESET** button. After this step, any attached debugger loses its connection. The default configuration of the application enables low power, which disables LED support. The user can manually change the configuration and enable LED support, otherwise all subsequent LED behavior references are ignored.

By default, the application uses extended advertising. However, it can be configured to use legacy advertising by setting the $\mathrm{gBeaconAE_c}$ define to 0. In the legacy configuration, the first press of the advertising switch starts the legacy advertising and the second press stops it.

Parent topic: Beacon

Usage The beacon can be tested with any Bluetooth® Smart Ready products available on the market. The IoT Toolbox can also be used to showcase the profile functionality, as shown in the figure **IoT Toolbox Beacon Demo** below.


7% 10:25
IoT Toolbox
Beacons
STOP
Manufacturer ID: NXP
UUID: 1B5ED501-83FD-A341-A845-80833D843013
A: 0
B: 0
C: 0
RSSI: -55 dBm

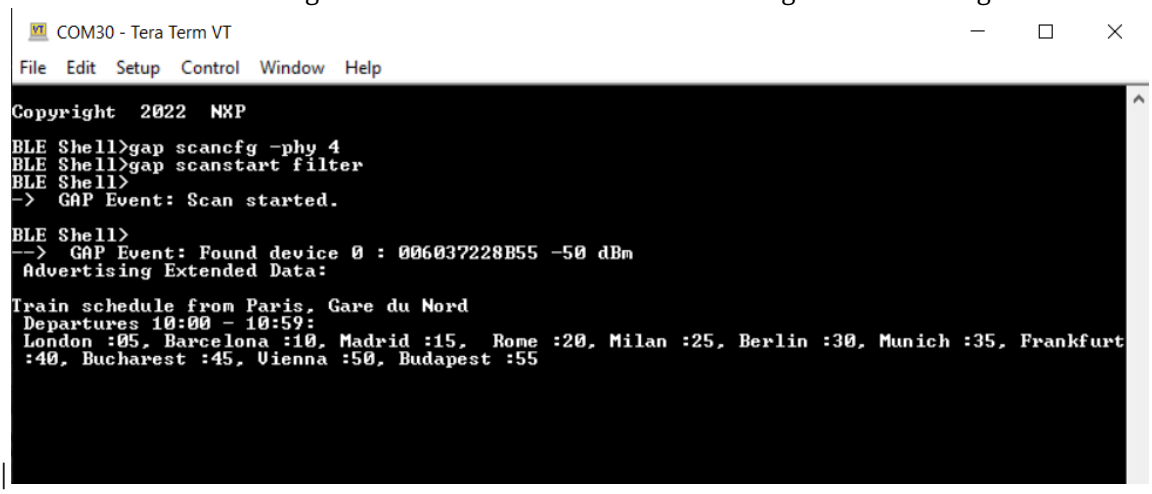
Parent topic: Beacon

Beacon usage with extended advertising To use the Beacon application with the advertising extensions capabilities, the gBeaconAE_c define option must be set to 1. Doing this enables the usage of extended advertising and periodic advertising. The application cycles between these modes are in the following manner:

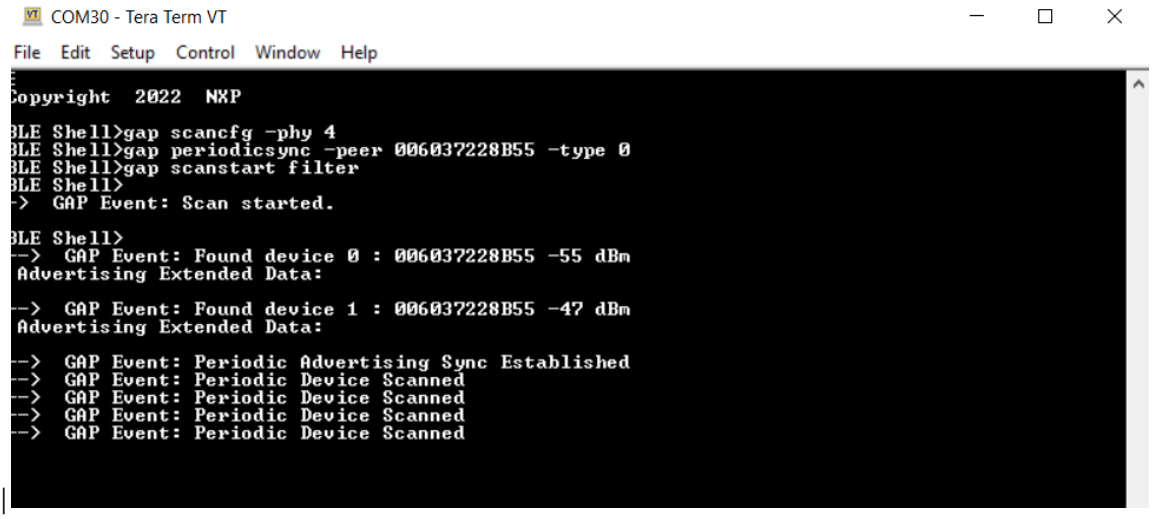
- The first **ADVSW** press starts legacy advertising, **CONNLED** turns solid.
- The second **ADVSW** press stops legacy advertising and starts extended advertising, **CONNLED** turns off, **EXTADVLED** turns solid.
- The third **ADVSW** press stops extended advertising carrying data and then starts extended advertising without data and periodic advertising, **EXTADVLED** starts flashing.
- The fourth **ADVSW** press stops periodic advertising and extended advertising without data and starts legacy advertising and extended advertising, both **CONNLED** and **EXTADVLED** turn solid.
- The fifth ADVSW press stops them all, both **CONNLED** and **EXTADVLED** turn off.

Not all smartphones support extended advertising, hence a different method to view the AE beacon is to use the ble_shell application. In order to do this, perform the following steps:

1. Flash a board with the beacon application, as described above.
2. Flash a board with the ble_shell application, as described in Bluetooth LE Shell and connect to it using a serial port.
3. Press the **ADVSW** button two times on the beacon to start extended advertising on the coded PHY.
4. To view the advertising data, enter the following commands in the shell terminal to set the scanning PHY to coded and start scanning. See the figure below.



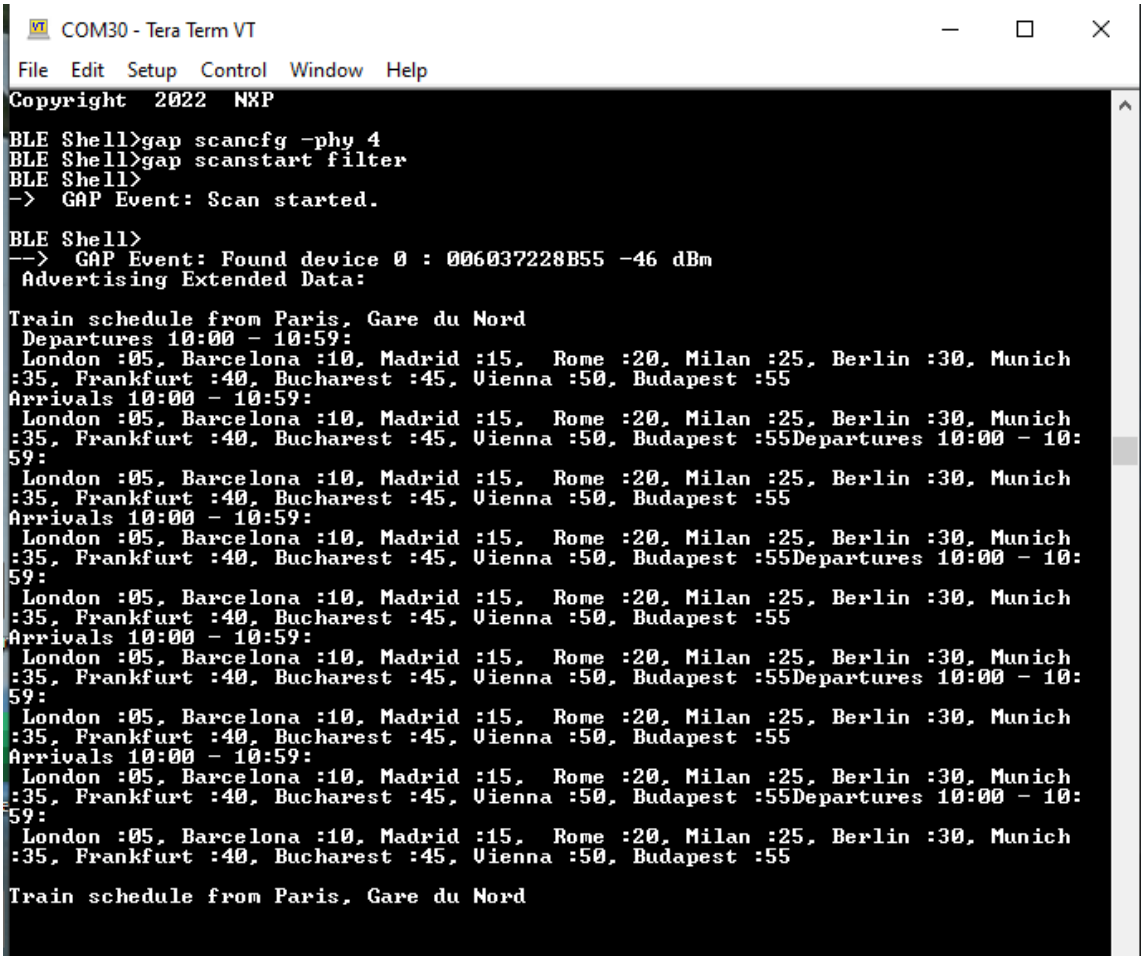
5. To start the periodic advertising, press **ADVSW** button again on the beacon.
6. To sync with the beacon, issue the following commands on the shell terminal as shown in the figure below.



The peer parameter of the periodicsync command is the public address of the beacon.

Extended Advertising with very large data To use very large advertising data for extended advertising, set the gBeaconLargeExtAdvData_c define to 1. The same steps are used to view the data using ble_shell :

1. Flash a board with the beacon application.
2. Flash a board with the ble_shell application, as described in Bluetooth LE Shell and connect to it using a serial port.
3. Press the **ADVSW** button two times on the beacon to start extended advertising on the coded PHY.
4. To view the advertising data, enter the following commands in the shell terminal to set the scanning PHY to coded and start scanning. See the figure below.



Parent topic: Beacon

Parent topic:*Bluetooth LE stack and demo applications*

Bluetooth LE FSCI Black Box This section describes the functionality, user interactions, and testing methods for the Bluetooth LE FSCI Black Box demo application.

Description The Bluetooth LE FSCI Black Box demo application gives access to the Bluetooth LE Host Stack via a serial interface using the FSCI protocol. See the *FSCI (Framework Serial Communication Interface) manual* for the format of the FSCI commands and a full list of supported commands.

The demo can be used with the Test Tool for Connectivity Products. Command Console application can be downloaded from the NXP website or using a custom application that supports the FSCI protocol and commands.

Parent topic: Bluetooth LE FSCI Black Box

Supported platforms The following platforms support Bluetooth LE FSCI Black Box application:

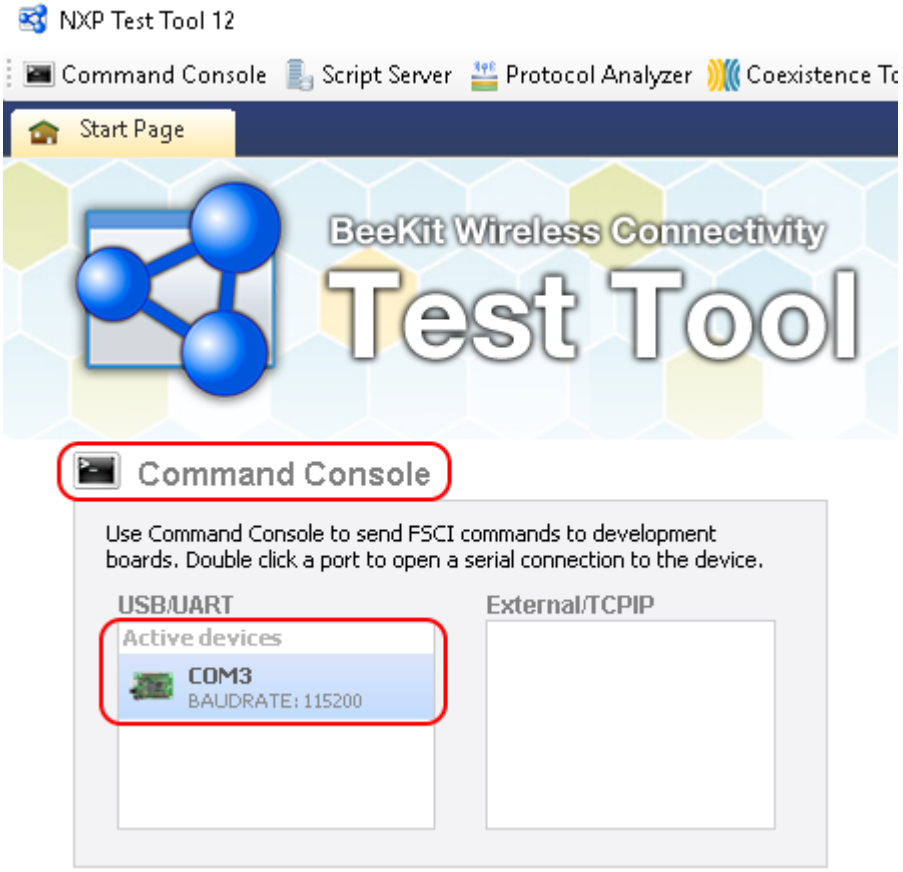
- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK

- FRDM-MCXW72
- MCX-W72-EVK

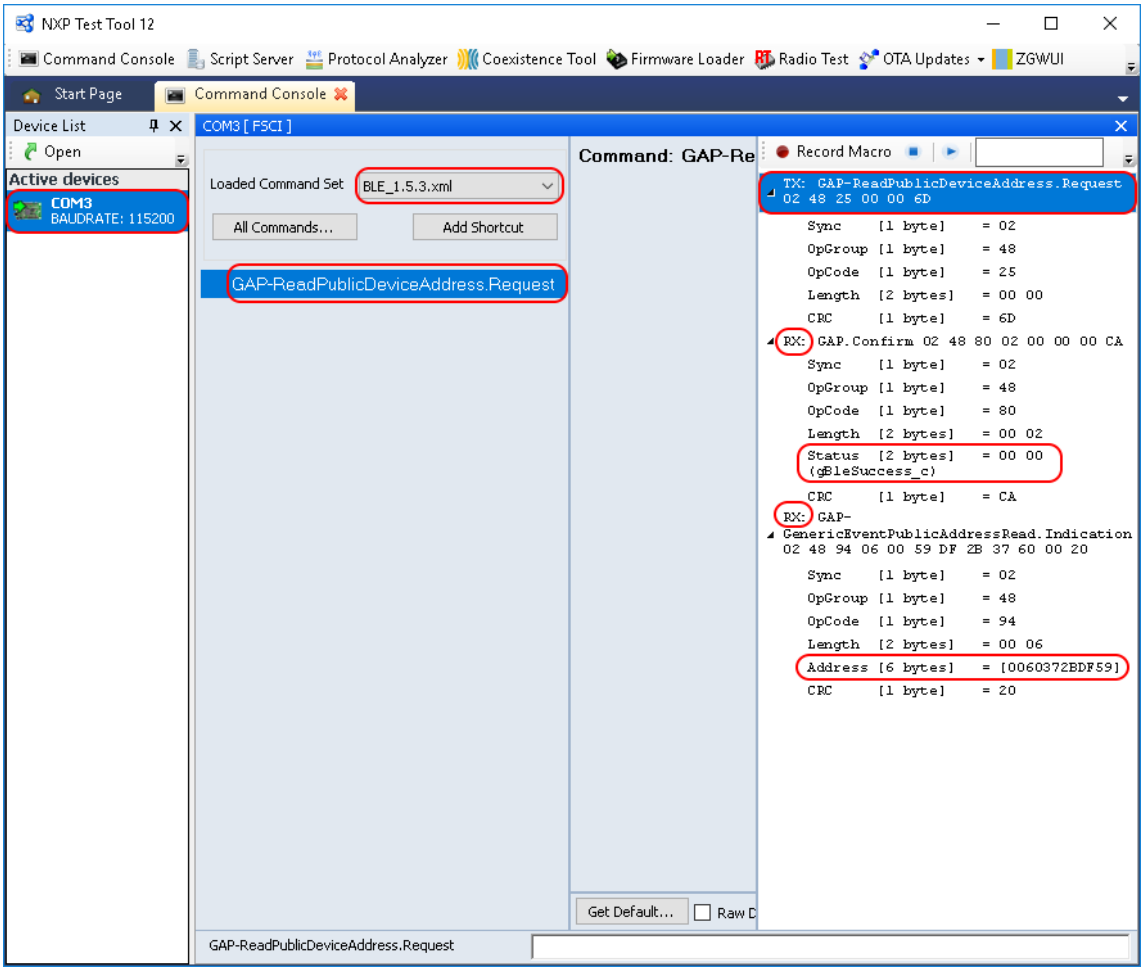
Parent topic: Bluetooth LE FSCI Black Box

Usage with Test Tool for connectivity products The Bluetooth LE FSCI Black Box demo application is designed to be used via serial interface. This can be done using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the demo application onto a supported board.
2. Connect the board to a USB port of the PC. The UASB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the Bluetooth LE FSCI Black Box application runs. See Figure. The serial communication parameters are: baud rate 115200, 8N1, and no flow control.



4. Select the appropriate Test Tool XML file from the drop-down list for the release being used and send commands to the application. An example is shown in Figure.



Parent topic: Bluetooth LE FSCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

EATT Central This section describes the implemented profiles and services, user interactions, and testing methods for the EATT Central application.

Implemented profiles and services The EATT Central application implements a GATT server and the following profiles and services:

- Generic Attribute Profile

The application behaves as a GAP central node. It searches for an EATT peripheral to connect to. After connecting, it performs service discovery, initiates an EATT connection and configures indications on the peripheral for services A and B. The Central reports the received service data and the steps taken during the setup on a terminal connected to an UART port.

Parent topic:EATT Central

Supported platforms The EATT Central application is supported on the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK

- FRDM-MCXW72
- MCX-W72-EVK

Parent topic:EATT Central

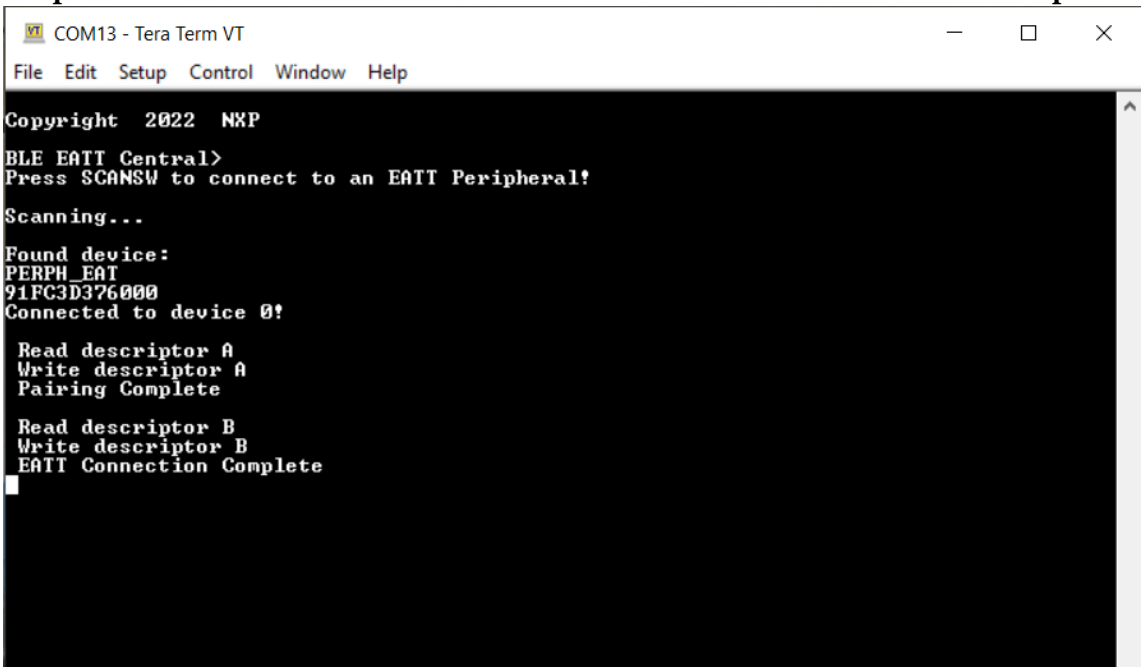
User interface After flashing the board, the central is in Idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. After connecting to the peripheral, the **CONNLED** turns solid. The data information together with the bearer it was received on is sent over UART. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then restarts scanning.

Parent topic:EATT Central

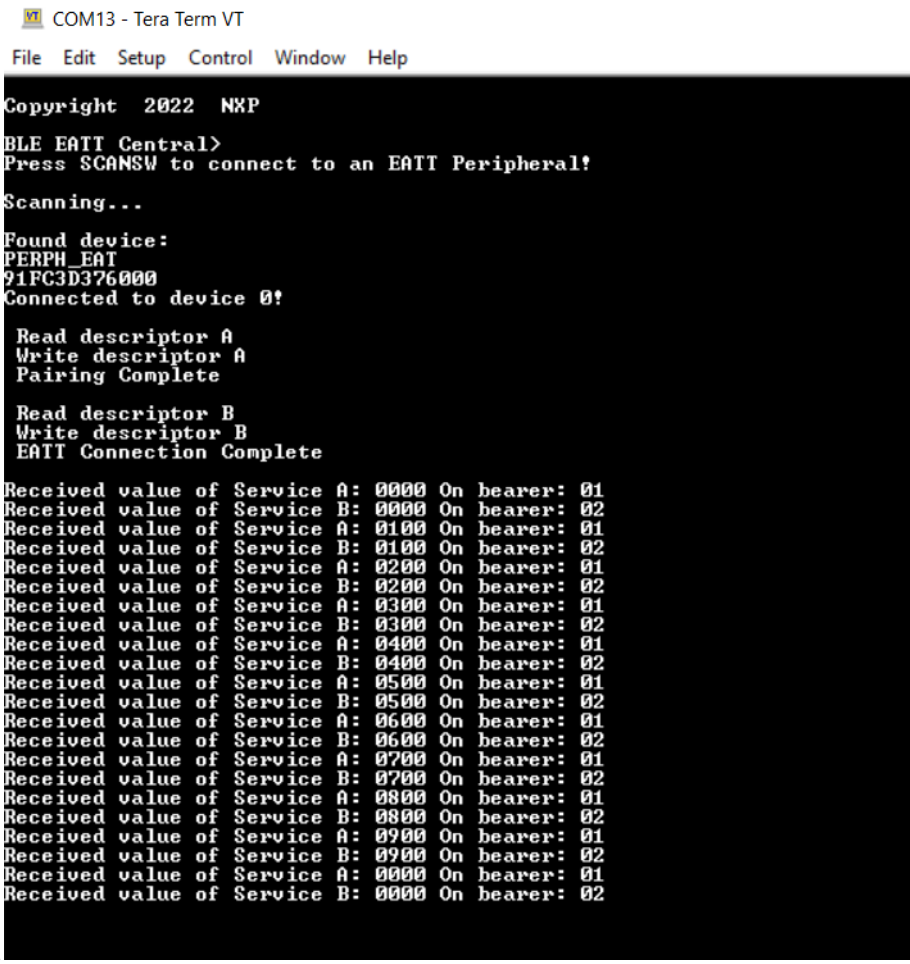
Usage The application can be tested using another board flashed with the EATT Central application as described in the EATT Peripheral.

1. Open a serial port terminal and connect it to board, in the same manner described in Testing devices. The start screen is displayed after the board is reset.
2. To start scanning for devices, press the **SCANSW** button on the EATT Central board. To make it enter discoverable mode, perform the same step on the EATT Peripheral board. The host connects with the board after it sees it advertise the service A and service B UUIDs. After connecting, the central performs service discovery, indicates its EATT support to the server by writing the Client Supported Features characteristic, enables indications for services A and B, and then initiates an EATT connection with the server. See the figure below.

   Output console on the EATT Central connected with an EATT Peripheral



3. After the EATT connection is complete, the console displays the received data and the bearer on which it was sent, as shown in the figure below.



Output console on EATT Central

Parent topic:EATT Central

Parent topic:*Bluetooth LE stack and demo applications*

EATT Peripheral This section describes the implemented profiles and services, user interactions, and testing methods for the EATT Peripheral application.

Implemented profiles and services The EATT Peripheral application implements a GATT server and the following profiles and services:

- Battery Service v1.0
- Device Information Service v1.1
- Generic Attribute Profile

The Generic Attribute Profile includes the Server Supported Features characteristics, which is used to indicate EATT support to the peer, and the Client Supported Features characteristic, which is used by the peer to indicate its own EATT support.

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect. The application implements two custom services, Service A and Service B. After the EATT connection is completed, the peer must enable indications for the two services to periodically receive profile data over EATT.

Parent topic:EATT Peripheral

Supported platforms The EATT Peripheral application is supported on the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

Parent topic:EATT Peripheral

User interface After flashing the board, the peripheral is in idle mode (all LEDs flashing). To start advertising, press the ADVSW button. When in GAP Discoverable Mode, **CONNLED** is flashing. When a central node connects to the peripheral, **CONNLED** turns solid. The node then waits for an EATT connection request and for the client to configure indications for the two services. The service information is sent over enhanced bearers only. The values indicated are cycled between 0 and 10. To disconnect the node, hold the ADVSW button for 2-3 seconds. The node then re-enters GAP Discoverable Mode and starts advertising.

Parent topic:EATT Peripheral

Usage The application can be tested using another board flashed with the EATT Central application as described in the EATT Central Section.

Parent topic:EATT Peripheral

Parent topic:*Bluetooth LE stack and demo applications*

HCI Black Box This section describes the functionality, user interactions, and testing methods for the Bluetooth LE HCI Black Box demo application.

Description The Bluetooth LE HCI Black Box demo application gives access to the Bluetooth LE Controller via a serial interface using the HCI protocol over serial interface. Refer to the Bluetooth Specification for the format of HCI commands and events over serial interface and the full list of supported commands and events.

The demo can be used with the Test Tool for Connectivity Products. The Command Console application can be downloaded from the NXP website. Alternatively you can also download it using a custom application that supports the HCI protocol and commands and events over serial interface.

Note: The HCI protocol encapsulation is dependent on the type of interface it is being used on. See the Bluetooth Specification for the HCI message format on each type of supported interface.

For instructions using the Bluetooth LE HCI Black Box with a serial terminal application or the hcitool in Linux, check the article "FRDM-KW40Z Bluetooth LE Controller Usage with the Linux hcitool".

Parent topic:HCI Black Box

Supported platforms The following platforms support the HCI Black Box application:

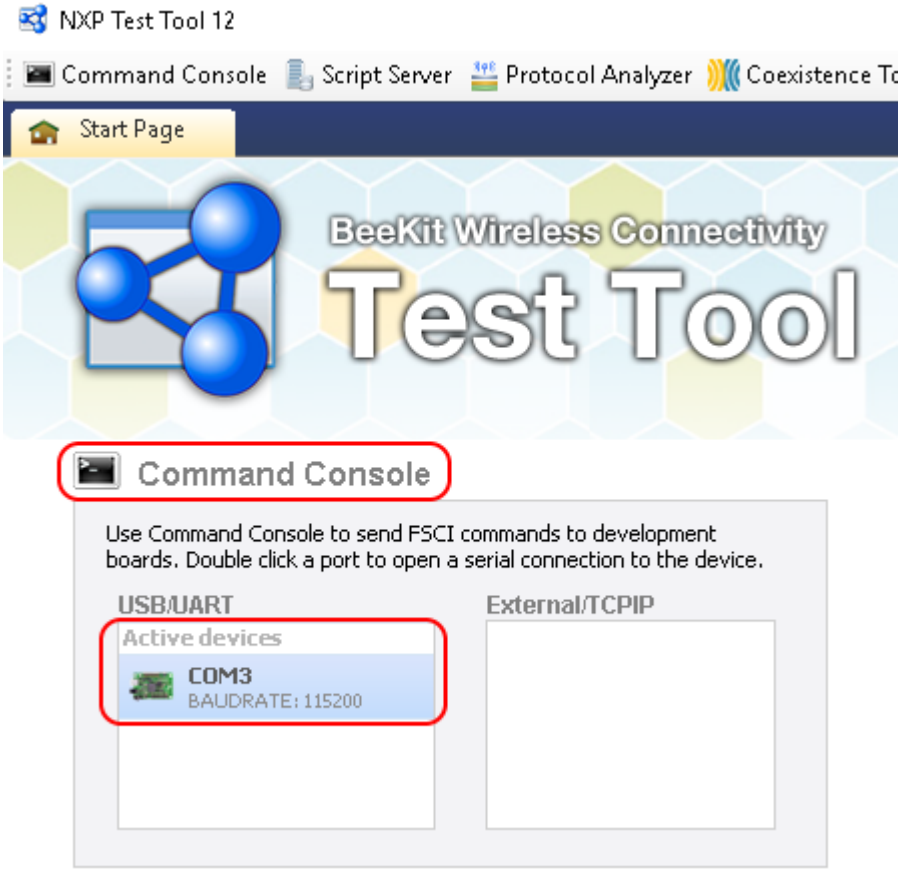
- KW45B41Z-EVK
- FRDM-MCXW71

- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

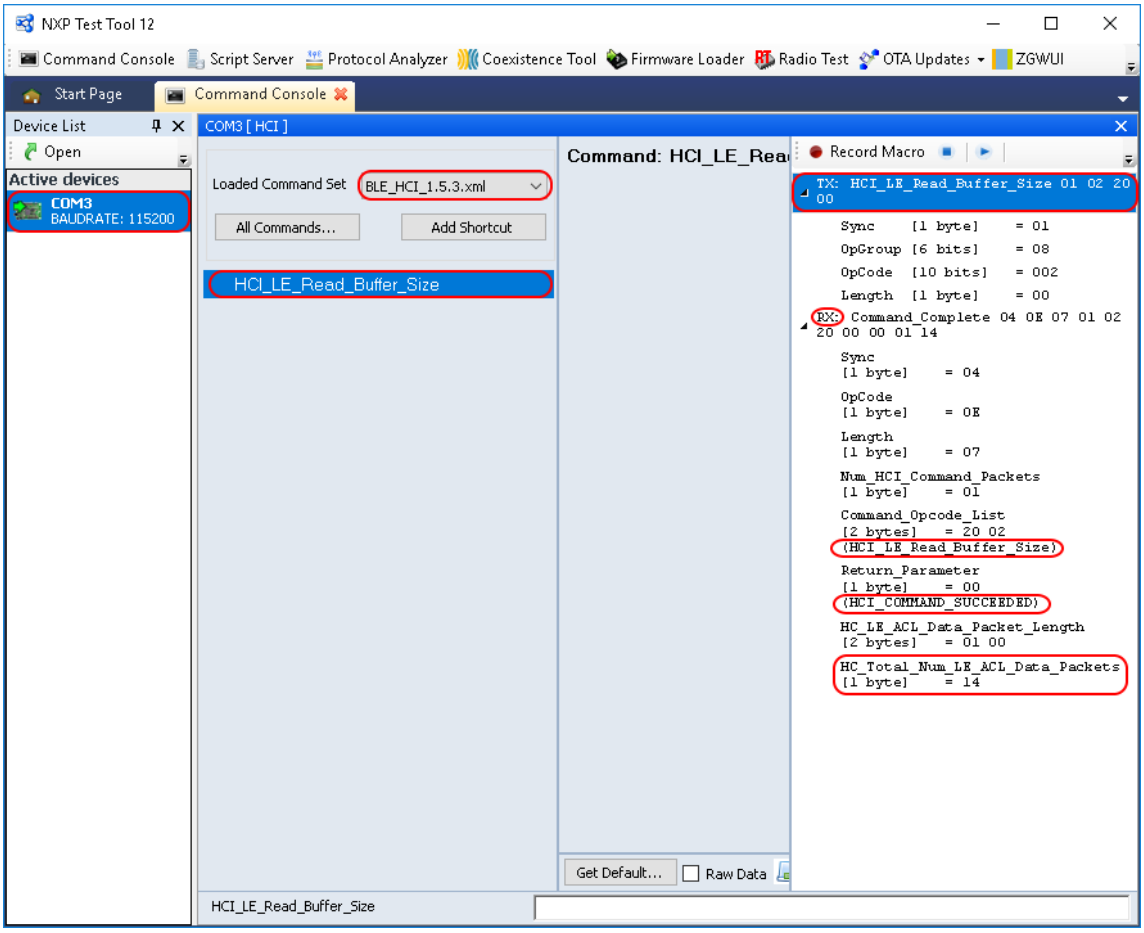
Parent topic:HCI Black Box

Usage with Test Tool for Connectivity products The Bluetooth LE HCI Black Box demo application is designed to be used via serial interface. This can be achieved using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the demo application to a supported board.
2. Connect the board to a USB port of the PC. The UASB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the Bluetooth LE HCI Black Box application runs. The serial communication parameters are: baud rate 115200, 8N1, and no flow control. See the figure **Test tool command console serial port selection** below.



4. Select the appropriate Test Tool HCI XML file from the drop-down list for the release you are using. Send a few commands to the application. An example is shown in the figure **HCI black box command example** below.



Parent topic:HCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

HID Device (Mouse) This section describes implemented profiles and services, user interactions, and testing methods for the HID mouse application.

Implemented profiles and services The HID Device application implements a GATT server and the following profile and services.

- HID over GATT Profile v1.0
- Human Interface Device Service v1.0
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect. Security on the services and bonding is enabled on this device.

When the GATT client configures notification, the application starts sending HID reports every two seconds with the movement of the MOUSE_STEP. The demo moves the cursor in a square pattern between AXIS_MIN and AXIS_MAX. The report contains 3 bytes, one for button status, one for X axis, and one for Y axis. The report descriptor matches the example in chapter E.10 from the USB Device Class Definition for Human Interface Devices (USB HID Specification), Version 1.11.

Parent topic:HID Device (Mouse)

Supported platforms The following platforms support the HID Device application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

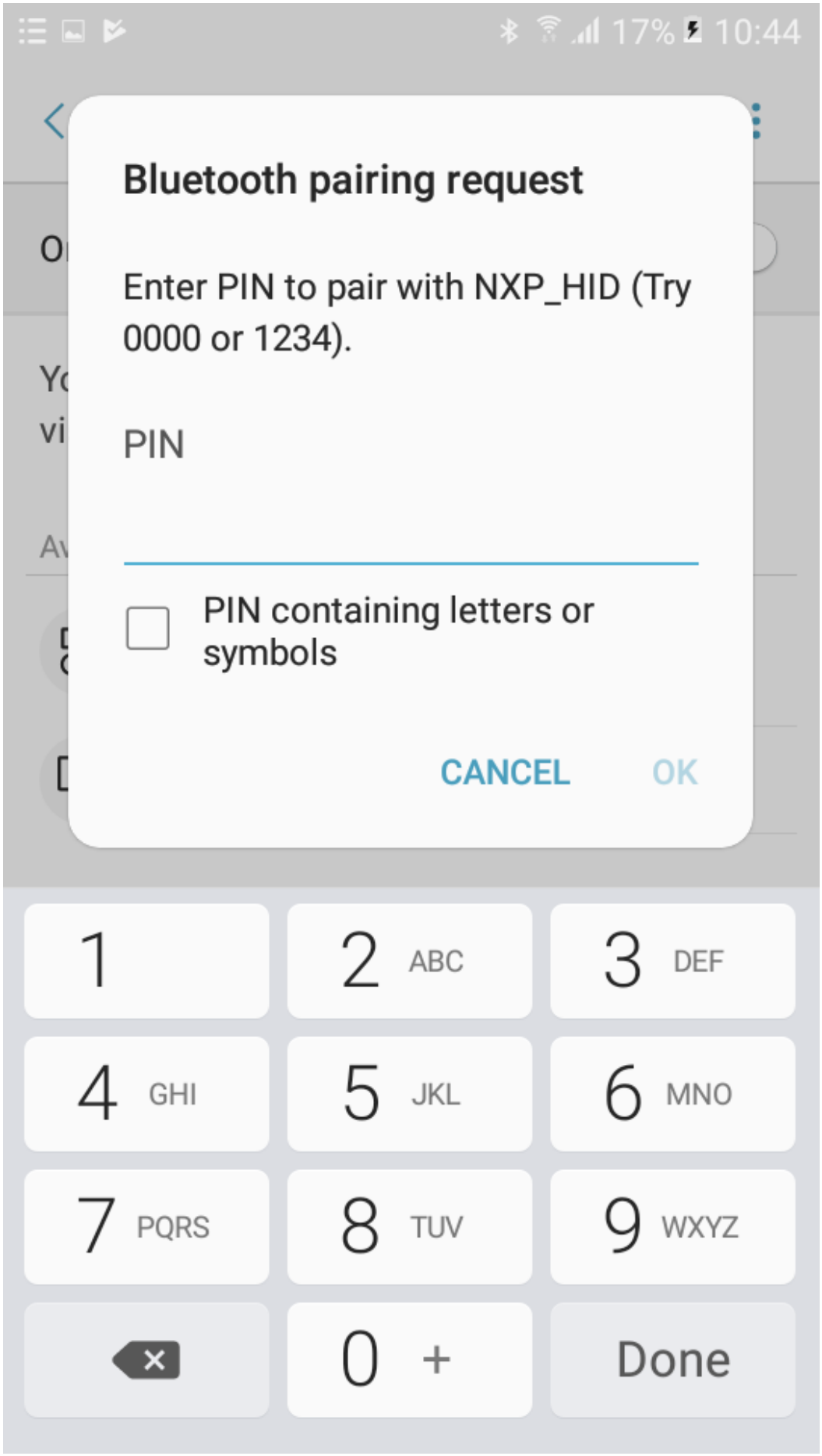
Parent topic:HID Device (Mouse)

User interface After flashing the board, the device enters Idle mode with all LEDs flashing. To start advertising, press the **ADVSW** button. In GAP Discoverable mode, **CONNLED** is flashing. When the central node connects to the peripheral, **CONNLED** turns solid. To disconnect the node, hold the **ADVSW** pressed for 2-3 seconds. The node then re-enters GAP Discoverable Mode.

Parent topic:HID Device (Mouse)

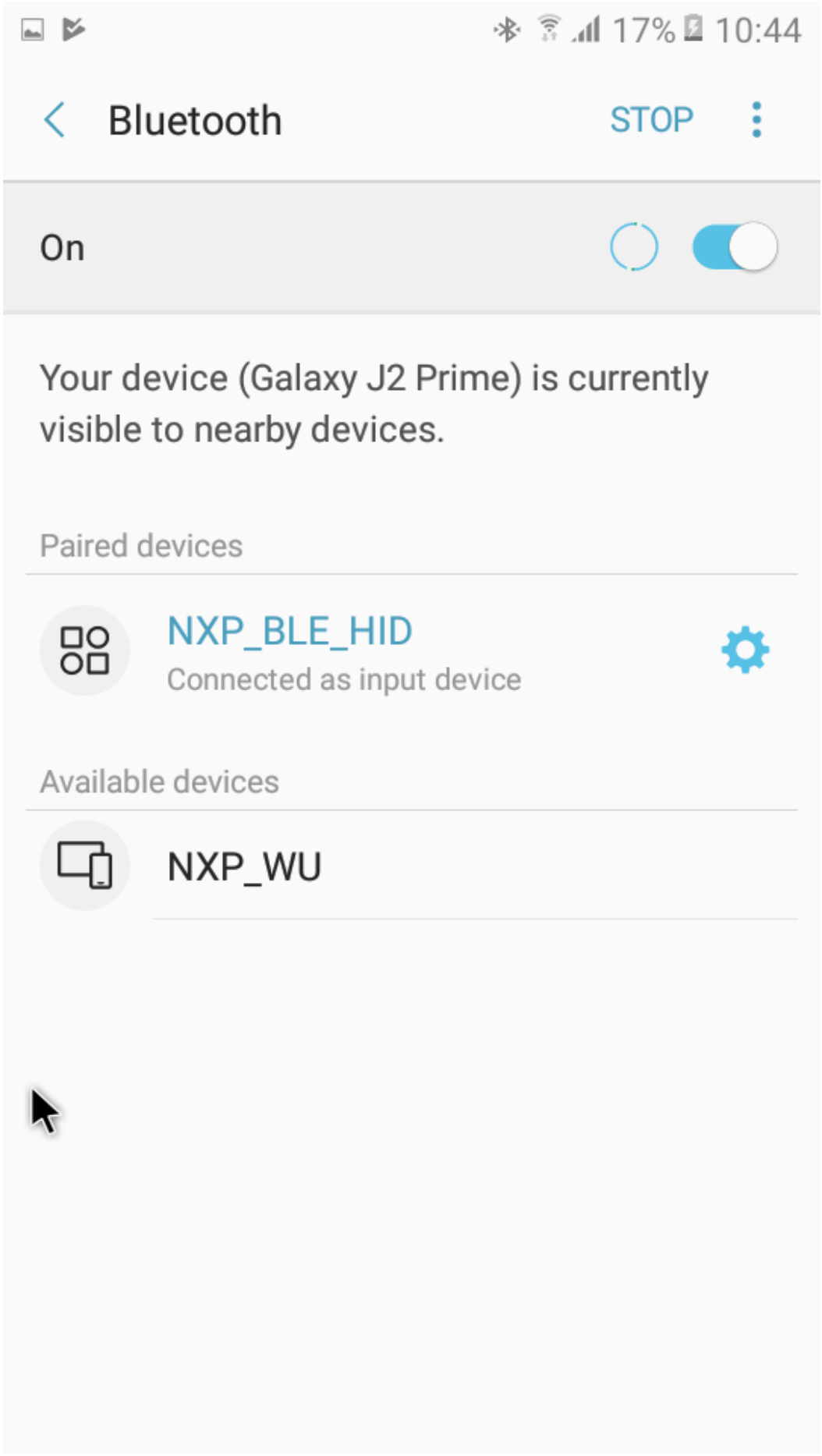
Usage The HID mouse can be connected to any Bluetooth Smart Ready products available on the market that supports HID devices or to another supported platform running the HID Host example (setup steps detailed in the HID Host section).

To make the HID mouse visible, press the **ADVSW** button to start sending advertisements, which causes CONNLED to start flashing. See the figure **Enter PIN prompt on Android platform** below.


17% 10:44
Bluetooth pairing request
Enter PIN to pair with NXP_HID (Try 0000 or 1234).
PIN
PIN containing letters or symbols
CANCEL
OK
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0 +
Done

The sensor name “NXP_HID” shows on the device when its scanning is active. A solid CONNLED indicates a successful connection between the 2 devices. When prompted to enter the pin, type the 999999 passkey.

When configured, the HID mouse starts sending HID report, which is configured as explained above, with notifications every 100 milliseconds. The mouse cursor shows a square pattern movement on the screen as shown in the figure **HID Mouse detected by Android platform** below.


17% 10:44
Bluetooth
STOP
On
Your device (Galaxy J2 Prime) is currently visible to nearby devices.
Paired devices
NXP_BLE_HID
Connected as input device
Available devices
NXP_WU

Parent topic:HID Device (Mouse)

Parent topic:*Bluetooth LE stack and demo applications*

HID Host This section presents the implemented profiles and services, user interactions, and testing methods for the HID Host application.

Implemented profiles and services The HID Host application implements a GATT client or server for the following profile and service.

- HID over GATT Profile v1.0
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters the GAP Limited Discovery Procedure and searches for HID devices to connect to. After connecting with the peripheral, it configures notifications and displays the received HID reports on a terminal connected to the UART port. The application uses pairing with bonding by default. When connected with the HID Device application, it sends the 999999 passcode to the host stack by default.

Parent topic:HID Host

Supported platforms The following platforms support the HID Host application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

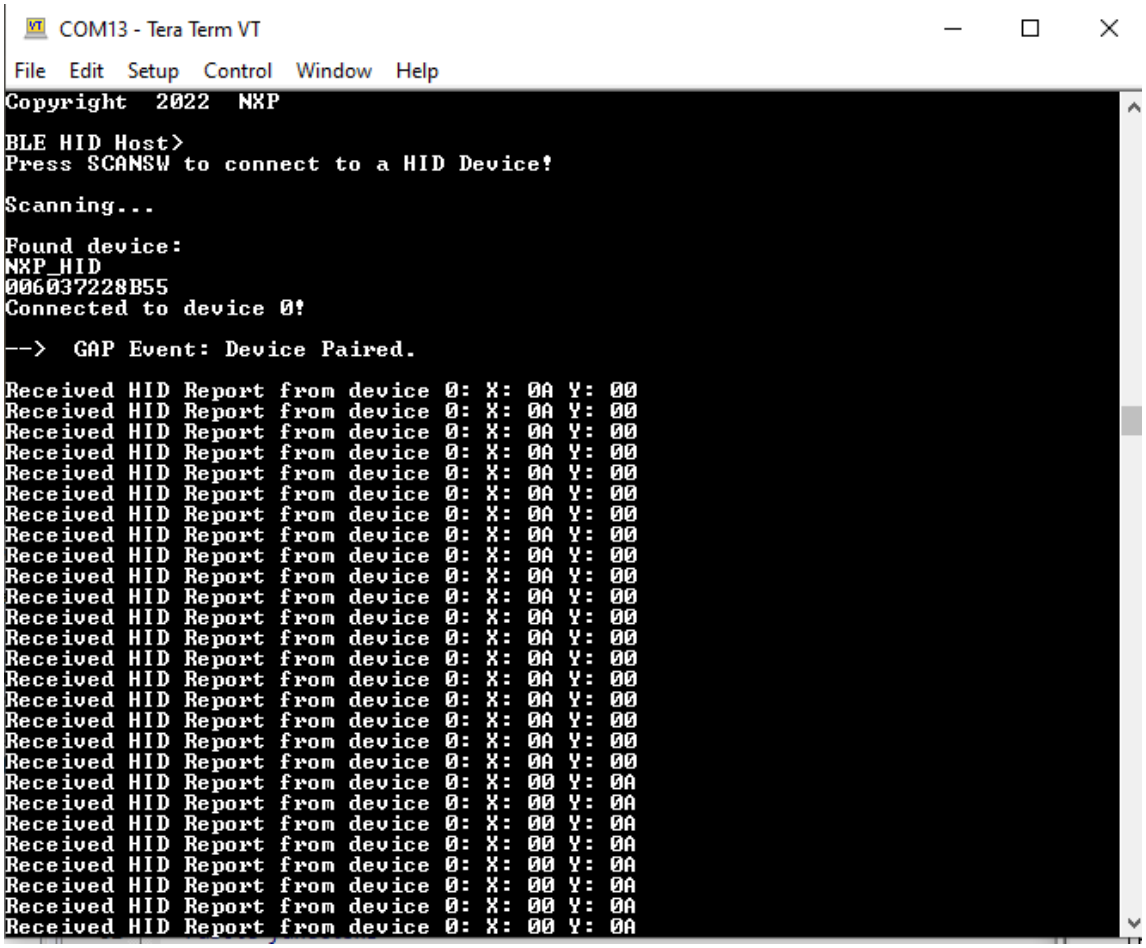
Parent topic:HID Host

User interface After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure, **CONNLED** is flashing. When the central node connects to the peripheral, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic:HID Host

Usage The application is built to work only with the HID Device application presented in HID Device (Mouse). It supports up to 2 peripherals connected at the same time.

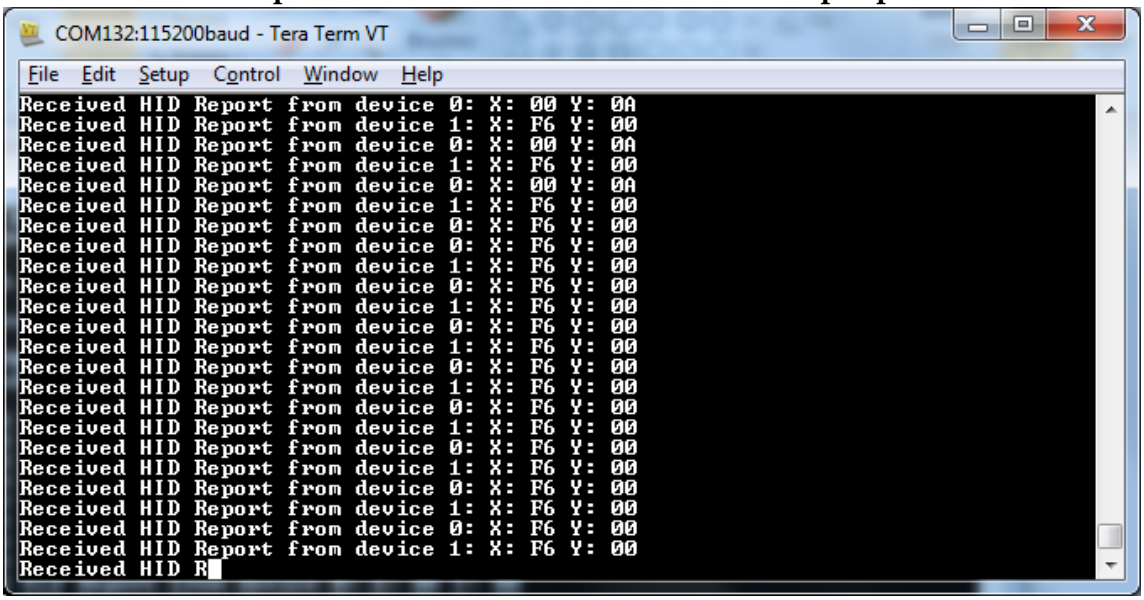
1. Open a serial port terminal and connect it to board, in the same manner described in Testing devices. The start screen is displayed after the board is reset.
2. To start scanning for devices, press the **SCANSW** button on the HID Host board. To make it enter discoverable mode, perform the same step on the HID device board. The host connects with the board after it sees it advertise the HID service, connects to it, and configures report notifications. The device then starts sending HID reports, as shown in Figure.



HID host

3. To connect a second HID device, press again the **SCANSW** button on the HID Host board to start scanning for devices. Do the same on the second HID device board to make it enter discoverable mode. The host connects with the board after it sees it advertise the HID service, connects to it, and configures report notifications. The device then starts sending HID reports. The console displays reports from both devices, as shown in Figure.

Tera Term – Output Console on HID Host with 2 peripherals connected



Parent topic:HID Host

Parent topic:*Bluetooth LE stack and demo applications*

Low-power temperature sensor and collector This section describes the implemented profiles and services, user interactions, and testing methods for the temperature sensor application.

Implemented profiles and services The Temperature Sensor application implements a GATT server, a custom profile and the following services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect and configure notifications for the temperature value.

The Temperature service is a custom service that implements the Temperature characteristic (UUID: 0x2A6E) with a Characteristic Presentation Format descriptor (UUID: 0x2904), both defined by the Bluetooth SIG.

The Temperature Collector application implements a GATT client or server for the following profile and services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for sensor devices to pair with. After pairing with the peripheral, it configures notifications and displays temperature values on a terminal connected to the UART port.

Both application uses pairing with bonding by default. When connected with the Low-Power Temperature Sensor application, the collector sends the 999999 passcode to the host stack by default.

Parent topic: Low-power temperature sensor and collector

Supported platforms The Temperature Sensor and Collector applications are supported by the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

Parent topic: Low-power temperature sensor and collector

User interface After flashing the board, both nodes enter Low-power mode. In case the sensor is put in deep sleep, press **WAKESW** or **RESET**. To flash the board in case the sensor is put in deep sleep, press either **WAKESW** or **RESET** button. By default, the application is configured to be in low power mode, which disables LED support.

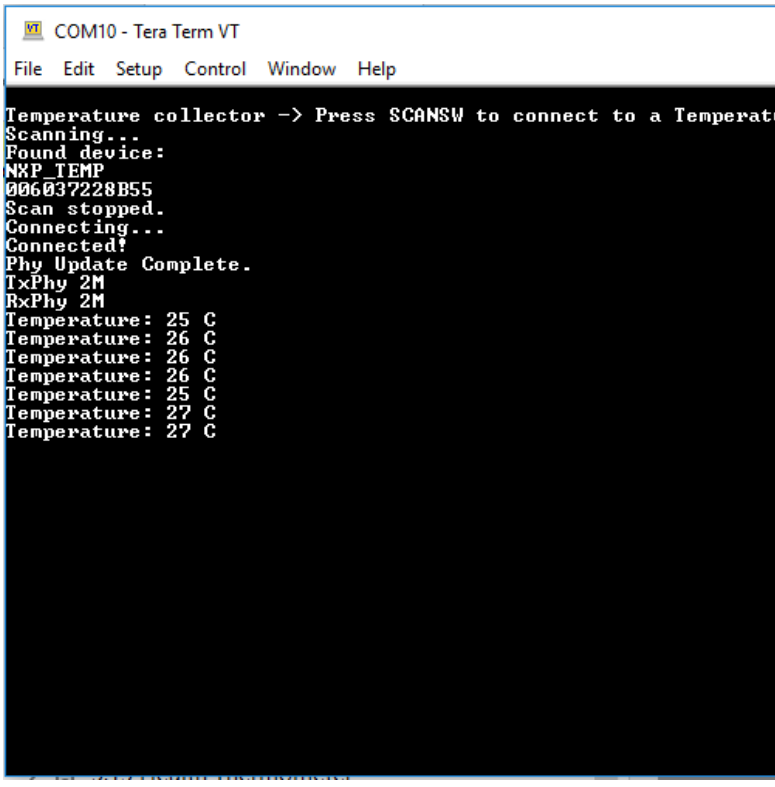
The user can manually change this configuration and enable LED support, else all subsequent LED behavior references are ignored and all LEDs are off. The devices disconnect and enter Deep-sleep only if low power is enabled. When the node is awake and communicating, **CONNLED** is on. To wake up the node, press the **WAKESW** button.

Parent topic: Low-power temperature sensor and collector

Usage The setup requires two supported platforms, one for the temperature sensor and one for the temperature collector.

1. Open a serial port terminal and connect it to the temperature collector board, in the same manner as described in Testing devices. The start screen is displayed after the board is reset. At first the LEDs are off on both devices.
2. To start advertising on the sensor, press the WAKESW button and CONNLED lights up. The sensor enters into the Deep-sleep mode, which means that the MCU wakes up on any packet from the Link layer, in this case the connect request. If no connection is established in an interval of 30 seconds, the sensor stops advertising and enters into the Deep-sleep mode again. CONNLED turns off.
3. To start scanning on the collector, press the WAKESW button and CONNLED lights up. The device wakes up, enters into the Deep-sleep mode, scans, and connects to a compatible sensor device. If no connection is established within 30 seconds, the collector stops scanning and enters Deep-sleep mode again. CONNLED turns off.
4. If the collector connects to a sensor node, it bonds (if no bond was previously made), does service discovery (only the first time it connects with the sensor), and configures notification and waits for notifications from the sensor for 5 seconds. If no data is sent, the node disconnects and re-enters Deep-sleep mode. The sensor exits low power and sends a notification with the value of the temperature read through an ADC from the thermistor, if present, or random generated if not.

   Once the connection is established, the PHY is automatically updated to 2M, if both the sensor and the collector support this feature as shown in the figure below. The PHY update is configurable from the application.



Output Console on Temperature Collector

5. Subsequent key pressing triggers other notifications for the collector. If no key is pressed in an interval of 5 seconds, the sensor node disconnects and re-enters Deep-sleep mode.

Parent topic: Low-power temperature sensor and collector

Parent topic:*Bluetooth LE stack and demo applications*

Low-power extended advertising Peripheral and Central This section describes the implemented profiles and services, user interactions, and testing methods for the adv_ext_peripheral and adv_ext_central applications.

Implemented profile and services The adv_ext_peripheral application implements a GATT server, a custom profile and the following services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect and configure notifications for the temperature value.

The Temperature service is a custom service that implements the Temperature characteristic (UUID: 0x2A6E) with a Characteristic Presentation Format descriptor (UUID: 0x2904), both defined by the Bluetooth SIG.

The adv_ext_central application implements a GATT client or server for the following profile and services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for peripherals devices to pair with. After pairing with the peripheral, it configures notifications and displays temperature values on a terminal connected to the UART port.

Both applications use pairing with bonding by default. When connected with the Low-Power Extended Advertising Peripheral application, the Extended Advertising Central application sends the 999999 passcode to the host stack by default.

To enable Periodic Advertising With Responses (PAWR), set the gAppPAWRSupport_d define TRUE in app_preinclude.h.

In case PAWR is enabled and more than one ext_adv_centrals are to be synced with the PAWR train, a different response slot may be configured for each board using the gUserDefinedResponseSlot_c define in app_preinclude.h to avoid collisions. Otherwise, a random response slot will be chosen by the application after reset.

To enable Encrypted Advertising Data (EAD), set the gAppEADSupport_d define TRUE in app_preinclude.h.

In case EAD is enabled, the advertising structures of gAdManufacturerSpecificData_c data type will be encrypted and transmitted over the air as gAdEncryptedAdvertisingData_c data type structures. On the receiver side, they will be decrypted and printed under the “Decrypted data :” text to mark them as EAD. This applies for extended advertising, periodic advertising, PAWR subevent data or responses.

Parent topic: Low-power extended advertising Peripheral and Central

Supported platforms The following platforms support Extended Advertising Peripheral and Central applications:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK

- FRDM-MCXW72
- MCX-W72-EVK

Deep-sleep mode is used by default.

Parent topic: Low-power extended advertising Peripheral and Central

User interface After flashing the board, both nodes enter Deep-sleep mode. To flash the board again, press **WAKESW**. The application default configuration enables low power that disables LED support. The user disables low power and enables LED support setting the gAppLowpowerEnabled_d define to 0. To wake up the node, press the **WAKESW** button. Both applications provide guidance over the UART.

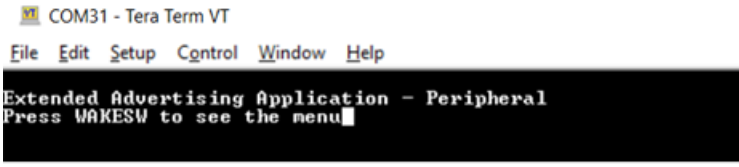
Open a serial port terminal using the following settings:

baud rate 115200, data bits 8, parity none, stop bits 1.

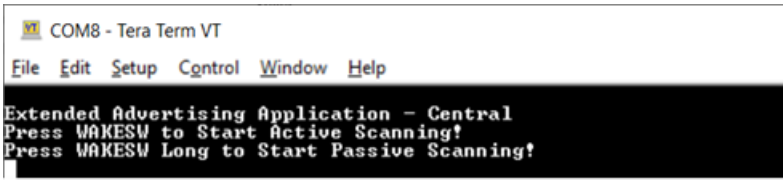
Parent topic: Low-power extended advertising Peripheral and Central

Usage The setup requires two supported platforms, one for the adv_ext_peripheral, and one for the adv_ext_central.

1. Open a serial port terminal and connect it to each platform with the settings provided in the previous paragraph. The start screen is displayed after the boards are reset as shown in the figures below.

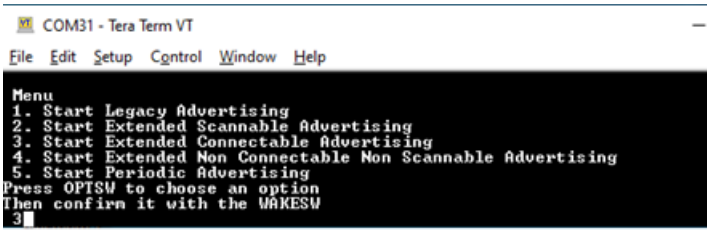


Advertizing external Peripheral start screen



Advertizing external Central start screen

2. On the board that implements the adv_ext_peripheral application, press the **WAKESW** button. The board exits Deep-sleep mode and displays the menu as shown in the figure below.



Choosing a menu option on adv_ext_peripheral

Use the **OPTSW** to choose an option. The option printed on the bottom changes every time the switch is pressed. When the option matches your intention (For example, 3 Starts Extended Connectable Advertising), press the **WAKESW** again to make a decision. The advertising type chosen is started and the board starts entering low-power between advertising events.

Next time, the **WAKESW** is pressed, an updated menu is printed (For example, at option 3 Stop Extended Connectable Advertising). There is no timeout for advertising. The board continues advertising until it is stopped, or a connection is established (for legacy and extended connectable advertising only) with an adv_ext_central device.

The connection is terminated five seconds after the central device configures notifications for the temperature value. When all advertising is off and all connections are terminated, the board enters low-power mode until the **WAKESW** button is pressed again.

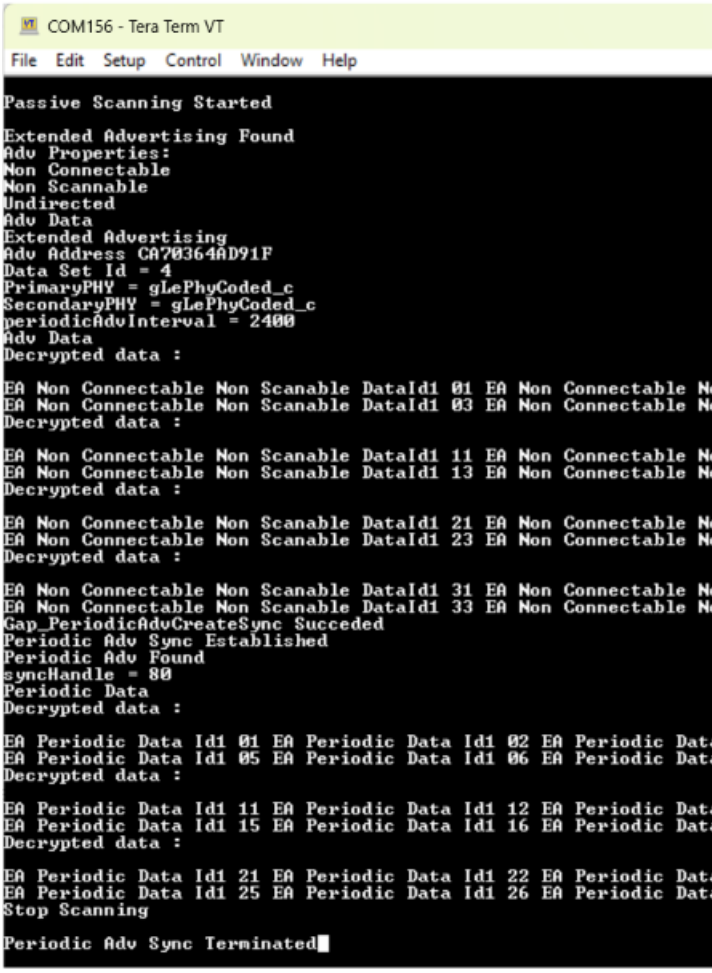
When gAppLowpowerEnabled_d is set 0, LEDs are enabled. The ADVLED flashes whenever an advertising starts and is ON otherwise. The CONNLED flashes whenever there is a connection under way and is ON otherwise.

When PAWR is started , advertising data is transmitted on subevents zero and three and responses are expected on all 5 configured response slots of these subevents. The central application responds to the periodic data received with a six bytes data composed of a three bytes random number followed by a three bytes hash performed over the random number using the local IRK in the same fashion the RPA are generated. The advertiser prints the responses, uses the bonded devices IRKs to generate the hash over the random number and attempts to connect over PAWR with the responder in case the hashes match.

```
PAWR Started
PAWR Set Subevent Data Complete
Periodic advertising response received
Adv Handle: 0
Subevent: 0
Response slot: 2
Response data: EFBE6E573231
Periodic advertising response received
Adv Handle: 0
Subevent: 3
Response slot: 2
Response data: 021CED6395B8
PAWR Set Subevent Data Complete
```

3. On the board that implements the **adv_ext_central** application, there are two options: Press **WAKESW** to start active scanning or long press **WAKESW** to start passive scanning. If catching extended scannable advertising is not an option, choose passive scanning. Otherwise, select active scanning. The device wakes up, starts scanning, and enters Deep-sleep mode. The scanning ends when the 60 seconds timeout is reached or when a connection with an adv_ext_peripheral device is established.

   During scanning, all advertisements caught from adv_ext_peripheral devices are displayed on the terminal window as shown in Figure. When an extended non-connectable, non-scannable advertising with a periodic advertising attached is detected, the *adv_ext_central*device attempts to sync with the periodic advertising train and prints the periodic advertising data on the terminal window.



Advertising caught on adv_ext_central console

If the PAWR support is enabled and the periodic advertising train the adv_ext_central has synced to is with responses the application attempts to sync with the subevents zero and three, periodic advertising data printed being slightly different. Also, the application responds to the periodic data received with a six bytes data composed of a three bytes random number followed by a three bytes hash performed over the random number using the local IRK in the same fashion the RPA are generated. The advertiser uses the bonded devices IRKs to generate the hash over the random number and attempts to connect over PAWR with the responder in case the hashes match.

```
Passive Scanning Started

Extended Advertising Found
Adv Properties:
Non Connectable
Non Scannable
Undirected
Adv Data
Extended Advertising
Adv Address C4603770BCC5
Data Set Id = 4
PrimaryPHY = gLePhyCoded_c
SecondaryPHY = gLePhyCoded_c
periodicAdvInterval = 2400
Adv Data
EA Non Connectable Non Scanable DataId1 01 EA Non Connectable Non Scanable DataId1 02
EA Non Connectable Non Scanable DataId1 03 EA Non Connectable Non Scanable DataId1 04
Gap_PeriodicAdvCreateSync Succeded
Periodic Adv Sync Established
```

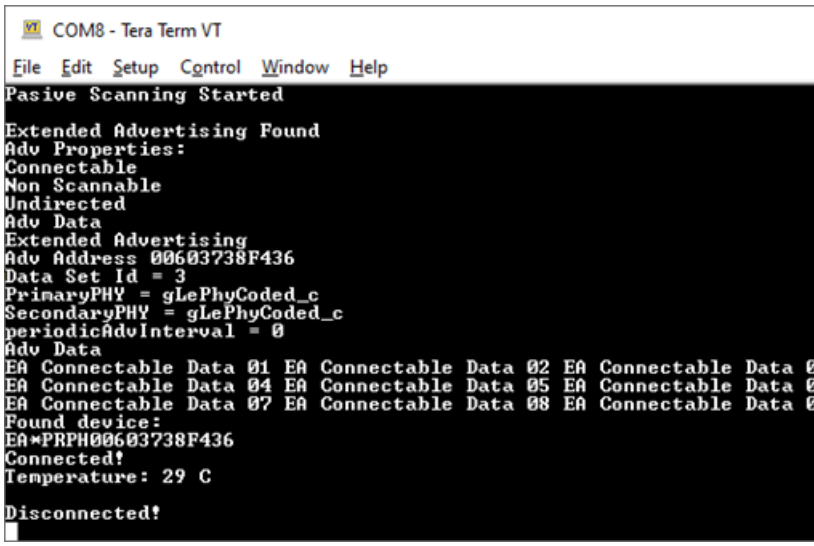
(continues on next page)

(continued from previous page)

```
Gap_SetPeriodicSyncSubevent Succeded
PAWR received
Sync Handle: 0050
Event Counter: 0007
Subevent: 0
Subevent data:
PAWR Data0 For Subevent 0
PAWR Data1 For Subevent 0
Gap_SetPeriodicAdvResponseData Succeded
Periodic Adv Set Response Data Complete
```

4. If the **adv_ext_central** connects to an **adv_ext_peripheral** device, it bonds (if no bond was previously made), does service discovery (only the first time it connects with the peripheral), configures notification and waits for notifications from the peripheral. If no data is sent within 5 seconds, the node disconnects and reenters Deep-sleep mode. The peripheral sends a notification with the value of the temperature read through an ADC from a thermistor, if present, or randomly generated, if not. When the central receives the notification, it displays it on the terminal window and disconnects in 5 seconds as shown in the figure below.

   When the 60 seconds timer expires or the connection ends, the device reenters Deep-sleep mode until the WAKESW is pressed again and all syncs with periodic advertising trains are terminated. If gAppLowpowerEnabled_d is set 0, LEDs are enabled. The SCANLED flashes, whenever the device is scanning and is ON otherwise. The CONNLED flashes, whenever there is a connection under way and is ON otherwise. See the figure below.



Connection on adv_ext_central console

   When the PAWR support is enabled and a connection with a previously bonded device takes place over PAWR, the applications behavior is similar except in this case the adv_ext_central's role is peripheral and adv_ext_peripheral's role is central.

```
PAWR Started
PAWR Set Subevent Data Complete
Periodic advertising response received
Adv Handle: 0
Subevent: 3
Response slot: 2
Response data: 564DE1FC4957
 Gap_ConnectFromPawr Succeeded
Connected!

Disconnected!
```

```
PAWR received
Sync Handle: 0050
Event Counter: 0007
Subevent: 0
Subevent data:
PAWR Data0 For Subevent 0
PAWR Data1 For Subevent 0
Gap_SetPeriodicAdvResponseData Succeded
Periodic Adv Set Response Data Complete

Connected!
Temperature: 24 C

Disconnected!
```

Parent topic: Low-power extended advertising Peripheral and Central

Parent topic:*Bluetooth LE stack and demo applications*

Over the Air Programming (OTAP) This section describes the implemented profiles and services, user interactions, and testing methods for the Bluetooth LE OTAP application.

Implemented profile and services The Bluetooth LE OTAP applications implement the GATT client and server for the custom Bluetooth LE OTAP profile and service.

- **Bluetooth LE OTAP Service** (UUID: 01ff5550-ba5e-f4ee-5ca1-eb1e5e4b1ce0)

The Bluetooth LE OTAP Service is a custom service which has 2 characteristics.

- **OTAP Control Point Characteristic** (UUID: 01ff5551-ba5e-f4ee-5ca1-eb1e5e4b1ce0). This characteristic can be written and indicated to exchange OTAP Commands between the OTAP Server and the OTAP Client. Data chunks are not transferred using this characteristic.
- **OTAP Data Characteristic**(UUID: 01ff5552-ba5e-f4ee-5ca1-eb1e5e4b1ce0). This characteristic can be written without response by the OTAP Server to transfer image file data chunks to the OTAP Client only when an image block transfer is requested via the ATT transfer method. Data chunks can also be transferred via the L2CAP credit-based PSM channels method.

The demo runs using 3 applications: an OTAP Client embedded application, an OTAP Server embedded application, and an Over the Air Programming PC application. The OTAP Client embedded application has two versions, an ATT version and a L2CAP version each using a different transfer method.

The embedded OTAP Server application is a GAP Central application which scans for devices advertising the Bluetooth LE OTAP service. After it finds one, it connects to it and configures the OTAP Control Point CCC Descriptor to receive ATT Indications from the device then it waits for OTAP commands from this device.

Once commands start arriving from the OTAP Client via ATT Indications the OTAP Server relays them via serial interface to a PC application which responds. The responses are then sent back to the OTAP Client by writing the OTAP Control Point Characteristic. The embedded OTAP Server application effectively acts as a relay between the OTAP Client to which the image is sent over the air and the Over the Air Programming PC application which has an OTAP image file constructed using a binary '*.srec*' image or a '*.bin*' image.

The OTAP Client is a GAP Peripheral which advertises the Bluetooth LE OTAP Service and waits for a connection from an OTAP Server. After an OTAP Server connects, the OTAP Client waits for it to write the OTAP Control Point CCCD and then starts sending commands via ATT Indications.

If the OTAP Client is configured to ask the data transfer via the L2CAP CoC PSM, it registers and tries to connect a predetermined L2CAP PSM before sending any commands to the OTAP Server.

Parent topic: Over the Air Programming (OTAP)

Supported platforms The following platforms support the OTAP applications:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-MCXW23
- MCXW23-EVK

Parent topic: Over the Air Programming (OTAP)

User interface After flashing two boards with the OTAP Server and OTAP Client applications respectively, the devices are in Idle mode (all LEDs flashing). To start advertising, press the **ADVSW** button on the OTAP Client. To start scanning, press the **SCANSW** button on the OTAP Server. After the two devices connect and start exchanging commands. **CONNLED** becomes solid on the OTAP Server and on the OTAP Client.

Start the OTAP Server PC application after the embedded applications are flashed to the boards. The application creates an OTAP image file using the provided executable .srec or .bin file. It then connects to the embedded OTAP Server via the configured serial interface and waits for commands. The application shows details about the image file creation and allows the OTAP upgrade image file header to be configured. The log view of the application displays the interactions between the OTAP Client and the OTAP Server.

Parent topic: Over the Air Programming (OTAP)

Usage with Over The Air Programming Tool Below is a list of requirements for usage with Test Tool for Connectivity products:

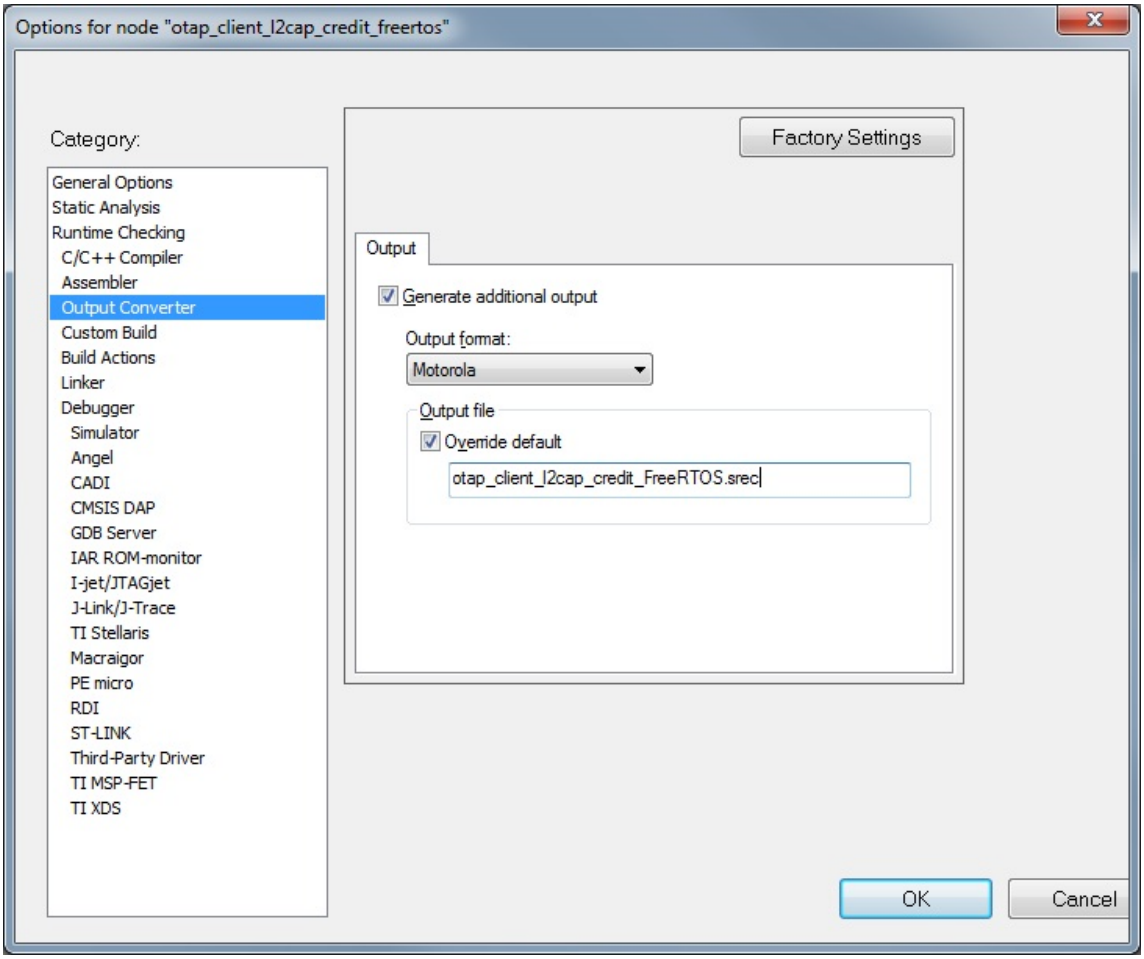
- Over The Air Programming Tool 1.4.0 or newer on CONNECTIVITY-TOOL-SUITE
- Serial COM port drivers – these are board-specific.

To run the application, follow the steps below:

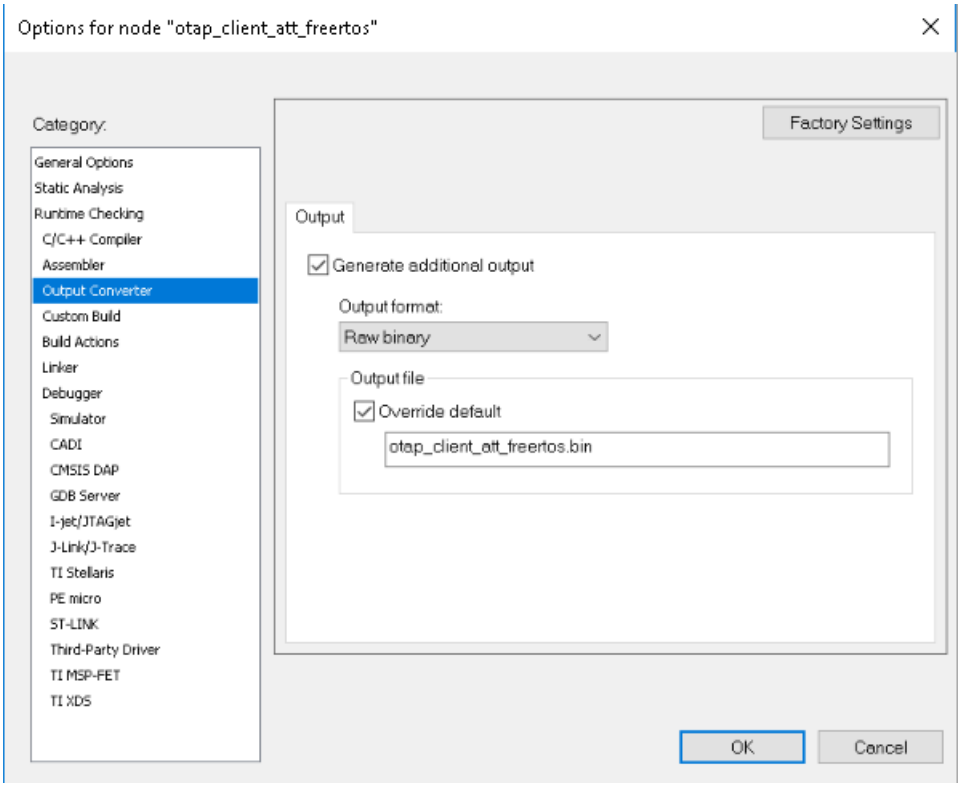
1. Flash the OTAP Server onto a supported platform and the OTAP Client to another supported platform. Make sure the board running the OTAP Server is connected to your PC and your PC has appropriate drivers for the USB to serial device on that board.
2. Create the application to send over the air. The executable must be provided in the .srec or .bin format. The .srec format executable can be obtained by using the IAR Output Converter and setting the output format to Motorola as shown in in the figure below.

   When compiling an image for the Over-the-Air update, the following changes are required:

   - The gEraseNVMLink_d linker symbol must be set to 0. For GitHub ARMGCC this may be changed in the application reconfig.cmake file.
   - For ARMGCC, gNvmErasePartitionWhenFlashing_c must be set to 0 in app_preinclude.h



3. To obtain a .bin file from IAR, select the **Raw binary** option in the **IAR Output Converter** as seen in the in the figure below.



4. To obtain a .bin file from MCUXpresso IDE, go to the **Project properties -> Settings -> Build steps**window and press the **Edit**button for the Post-build steps. A **Post-build steps** window shows up. In this window, add the following command:

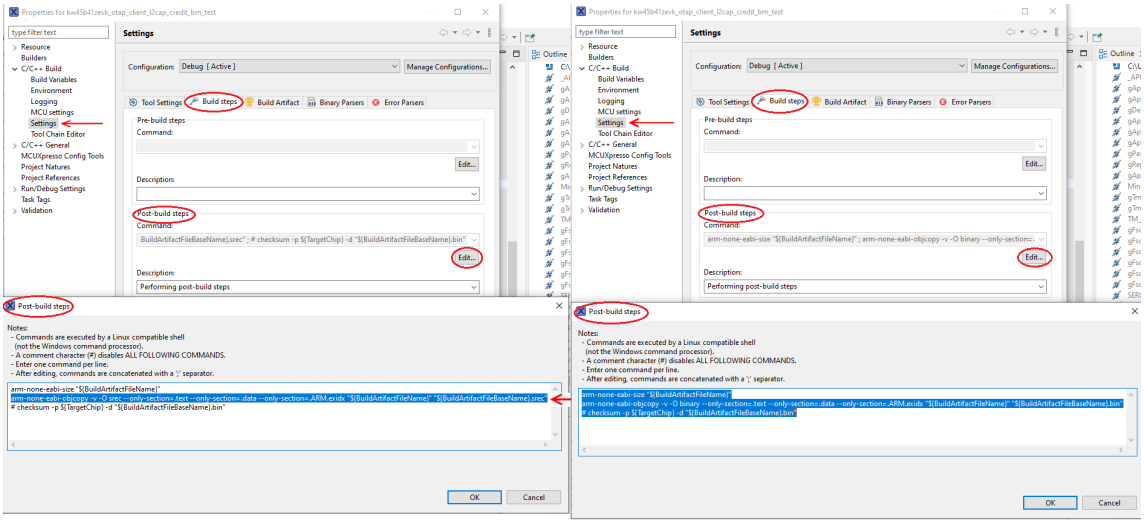
```
arm-none-eabi-objcopy -v -O binary  --only-section=.text
 --only-section=.data
--only-section=.ARM.exidx "${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin"
```

In case the command already exists, uncomment it by removing the '#' character at the beginning.

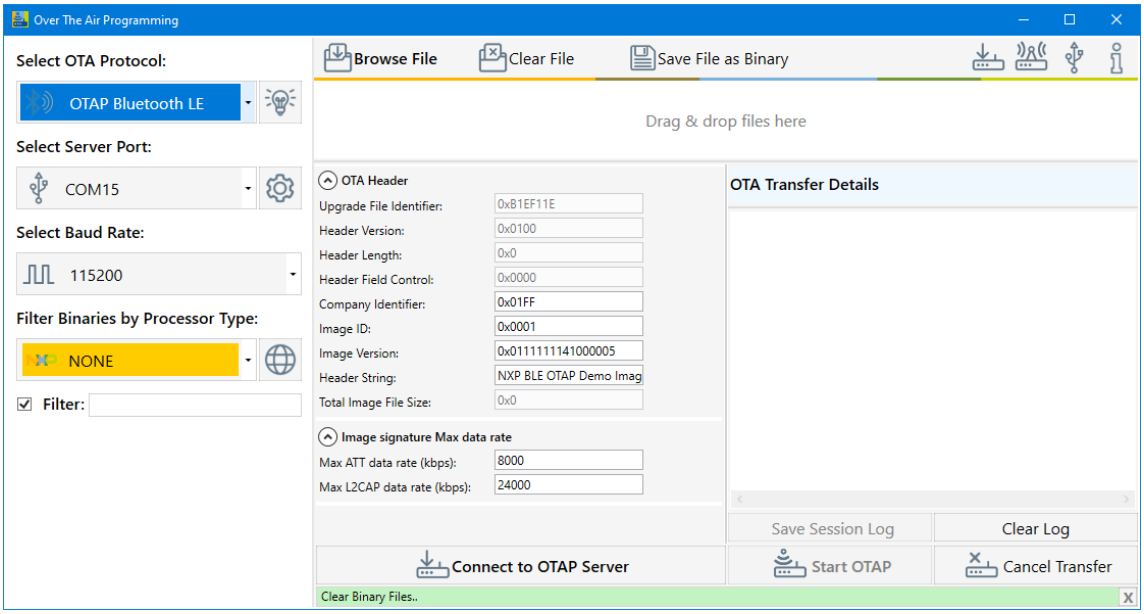
To obtain a .srec (.s19) file, add or uncomment the following post-build command in the same window:

```
arm-none-eabi-objcopy -v -O srec  --only-section=.text
 --only-section=.data --only-section=.ARM.exidx"
${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.s19"
```

This window is shown in the figure below.

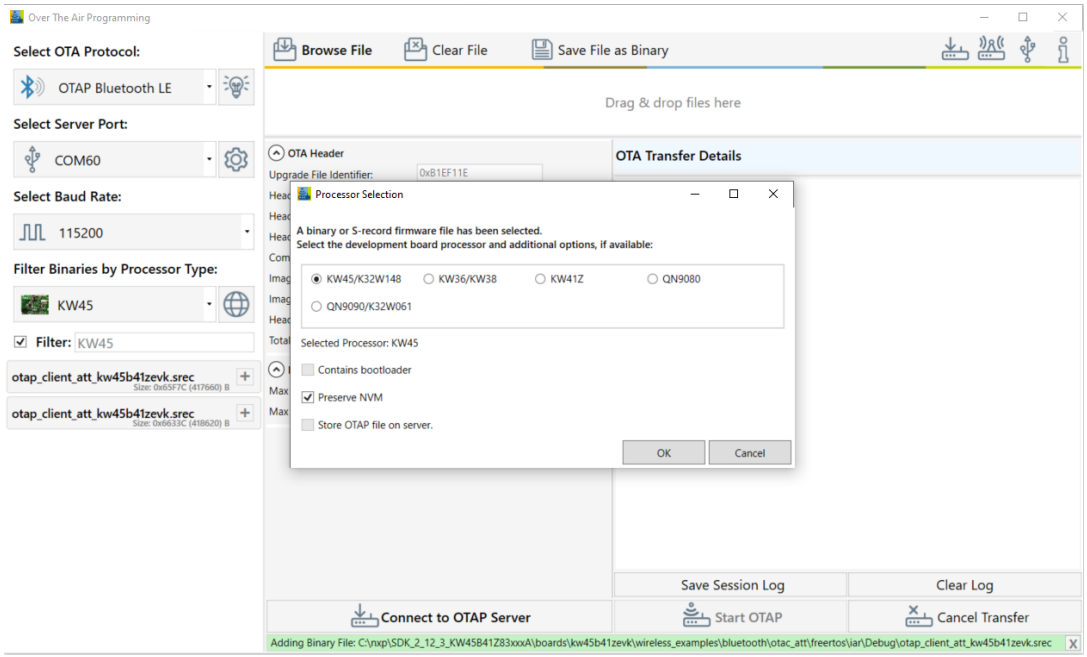


5. Start the **Over The Air Programming** application and select "**OTAP Bluetooth LE**" from the "**Select OTA Protocol**" combo box as shown in the figure below.

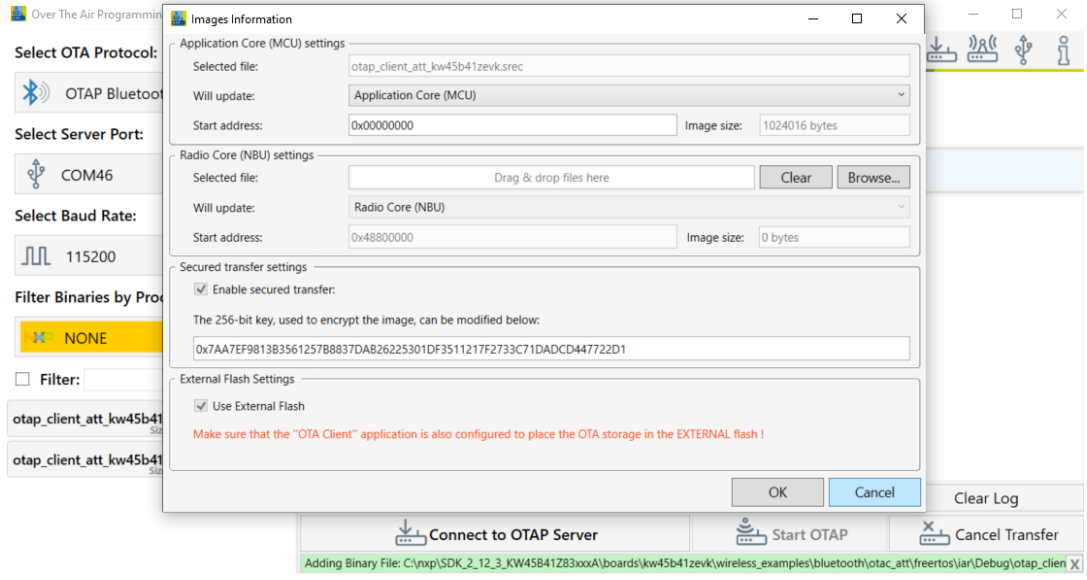


6. **Load the image file into the application, then configure the image file header and start the OTAP Server:**

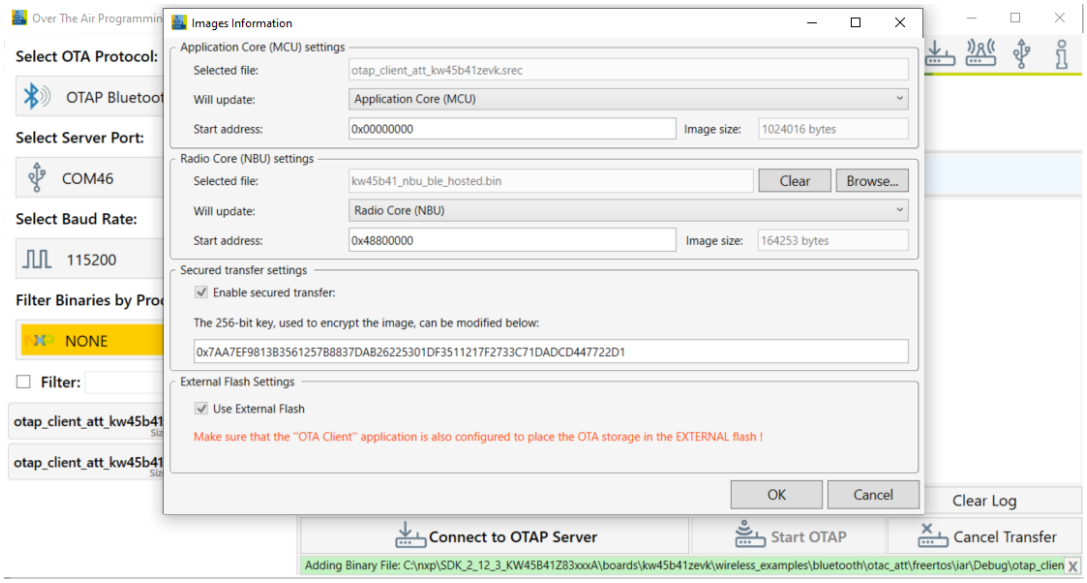
   - To select the updated image In the Over the Air Programming tool, select the "**Browse File**" button and then navigate to the .srec or .bin file containing the image to be sent to the OTAP Client. After the .srec or .bin file is chosen, a pop-up window asks to choose the target processor. In this example, the **KW45/K32W** processor will be used. Choose the **KW45/K32W** processor and press **OK**. See the figure below.



   - Once the processor is selected, a new pop-up window would appear that allows selecting the type of image as shown in the figure below. (In the specified case, we selected the KW45Z/KW45Z/K32W1(MCU).

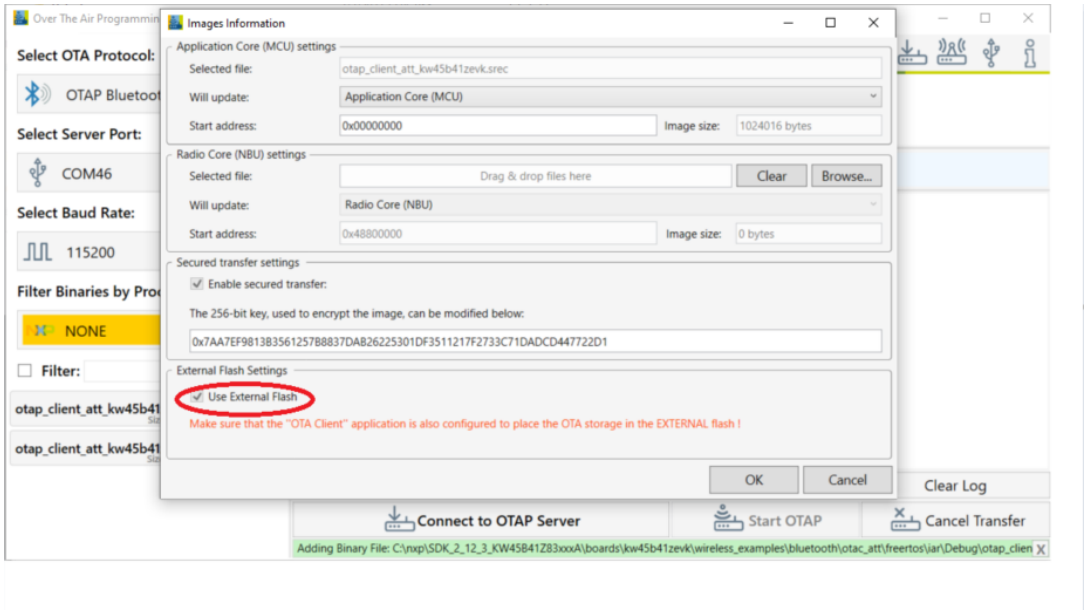


   - To update the KW45B/KW32W1 (radio) image, select it by pressing the "**Browse**" button in the M3 group. Then navigate to the .bin file as shown in the figure below.



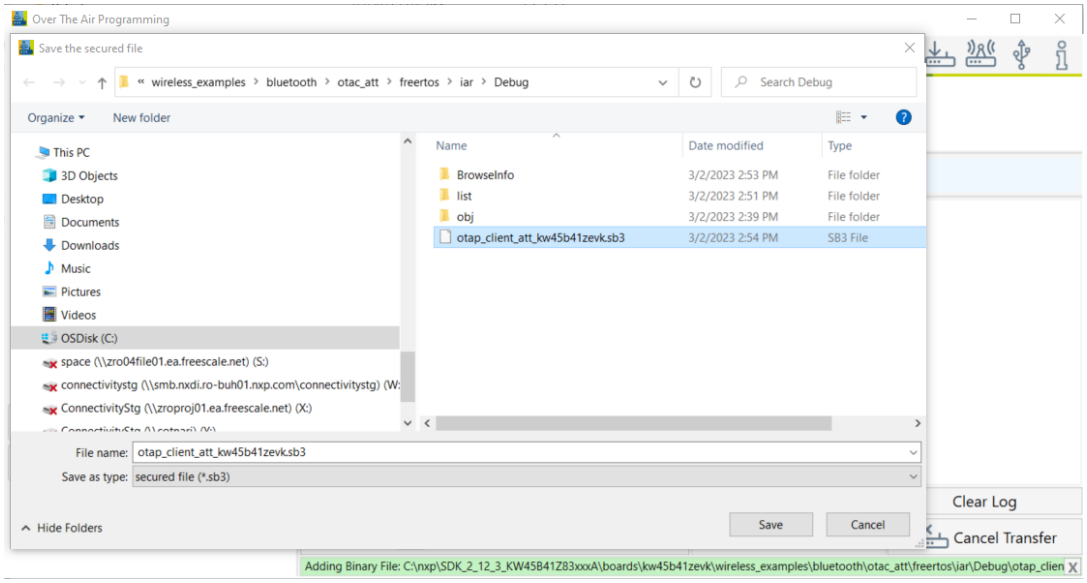
- If the OTA client has configured external memory support, then "**Use External Flash**" checkbox must be checked as in the figure below. If the OTA client has configured internal memory support, the checkbox must be left unchecked.

  This checkbox (if checked) instructs the OTA client bootloader to erase all the internal storage. This must be done only if external memory support is used as shown in the figure below. .



At this moment, click the **OK**button. A new window appears that prompts you to enter a location where the secured file should be stored. By default, the location of the original file is selected. See the figure below.

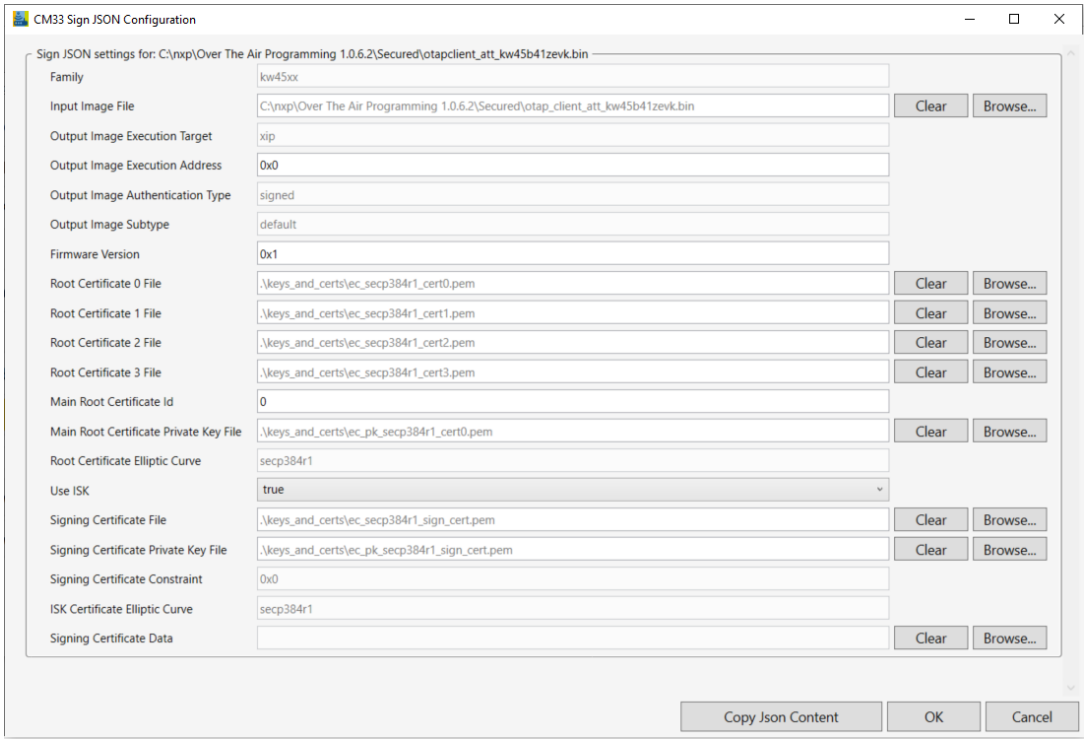
Note: The extension of the secured file is *.sb3. See the figure below.



- You can now configure two different JSON files, used to:
  - Sign the file that is uploaded to the MCU if an MCU file was selected.
  - Create the **.sb3*container that is sent OTA. The **.sb3* file can contain only the MCU file, only the radio file, or both.

If you select a file that is written on the MCU, a new window appears as shown in the figure below. This window helps in configuring the root certificates and signing the certificates, by either dragging and dropping, or browsing for new files. For details on each field of the JSON, see */Documentation/KW45JsonDescription.pdf* provided with Over the Air Programming tool.

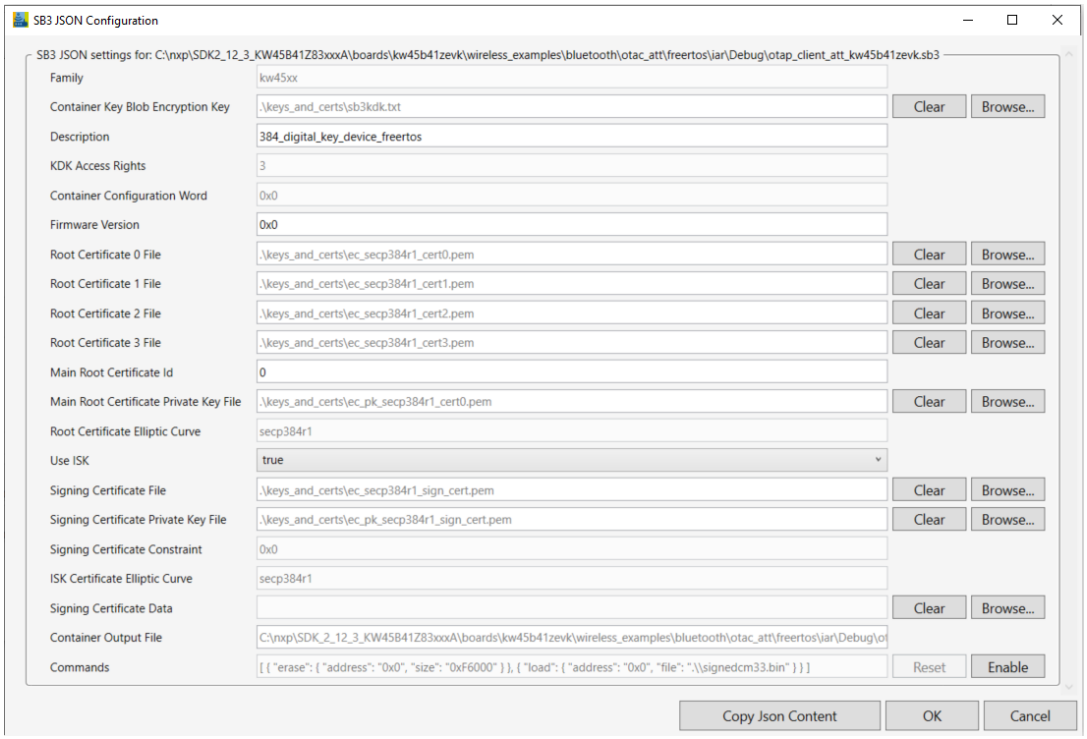
By default, the JSON is configured for the demo applications to run as shown in the figure below.



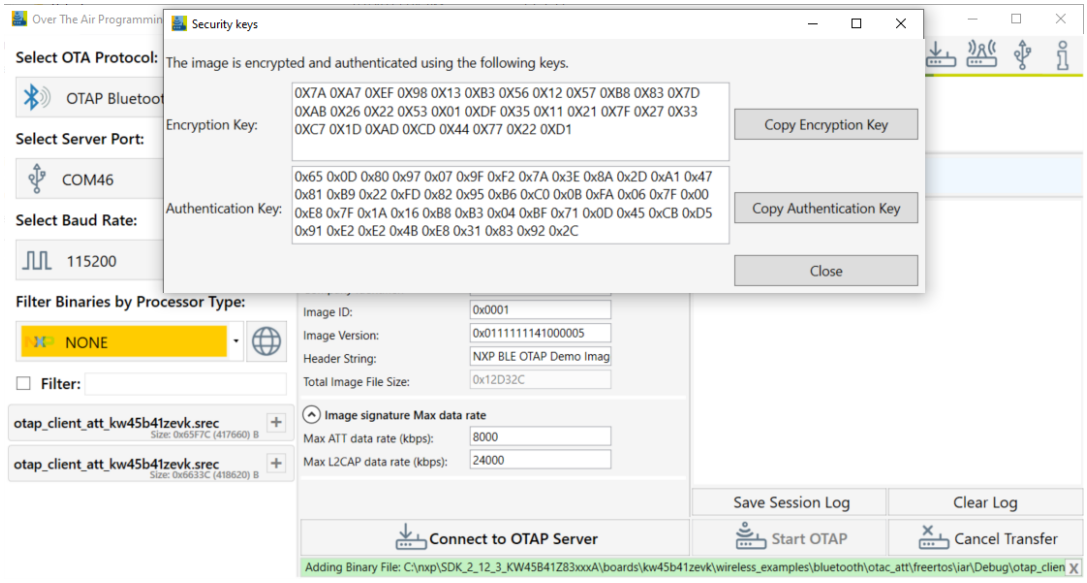
After configuring the JSON file used for signing the MCU file, a new similar window appears. As shown in the Figure, the window is designed for configuring the **.sb3* con-

tainer. This window helps you to configure the encryption key file, the root certificates, and the signing certificates by either drag and dropping or browsing for new files. For details on each field of the JSON file, see */Documentation/KW45JsonDescription.pdf* provided with Over the Air Programming tool.

By default, the JSON is configured for the demo applications to run as shown in the figure below.



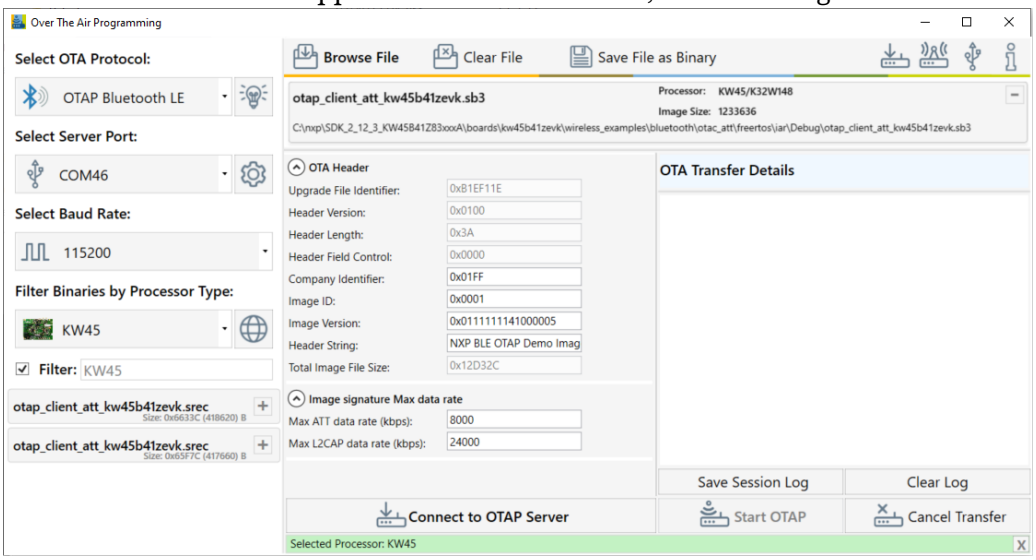
- After the .sb3 file is created, the "**Encryption Key**" and "**Authentication Key**" are presented. For the secured update to be successful, the destination board must have been provisioned with these keys through fuse burning, as described in the accompanying document. Depending on the board type, it can either be already provisioned by NXP (KW45\KW47\MCX-W71\MCX-W72 samples) or not provisioned (loose samples). See the figure below.



- The OTA Header configuration options from the "**OTA Header**" box are used by the application to build the **OTAP Image File**, which is sent over the air. The default values of

the OTA Header configuration work out of the box for the OTAP demo applications. For details about these configuration options, see the *Bluetooth LE Application Developer's Guide* document (*BLEADG*).

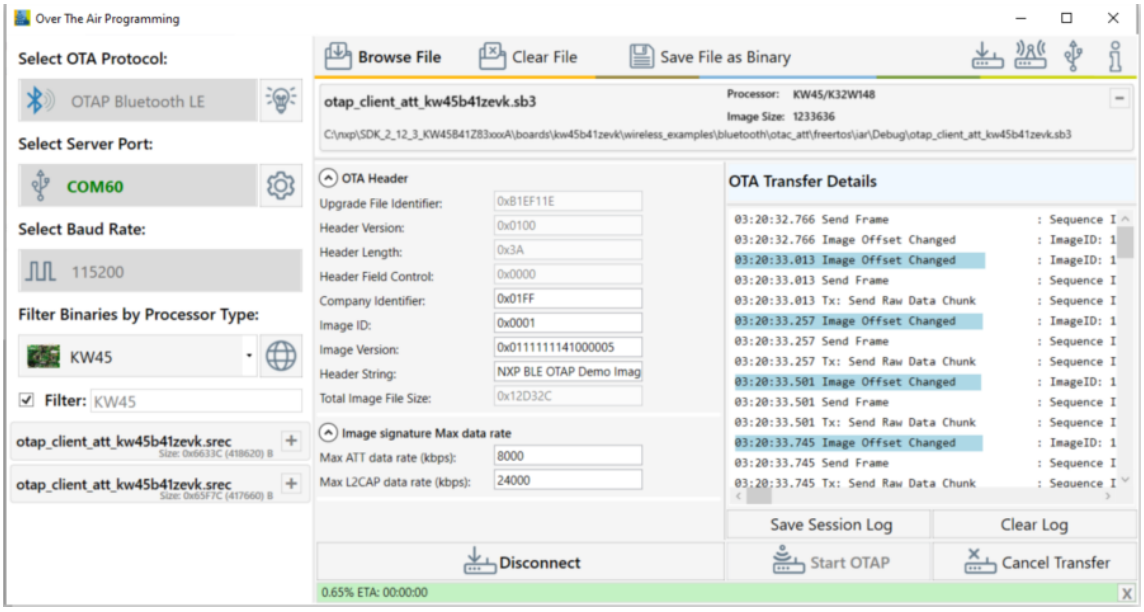
- After the image is loaded, go to the "**Select Server Port**" box, select the correct COM Port for the OTAP Server board. Also select the default baud rate of 115200 and press the "**Connect to OTAP Server**" button. A successful connection is displayed in the Message Log.
  - If the image is loaded before connecting to the OATP Server COM Port, then the OTAP Server of the application starts automatically.
  - If the connection to the COM Port is established before the image is loaded, then the "**Start OTAP**" button must be pressed to start the OTAP Server of the application. For details, see the figure below. .



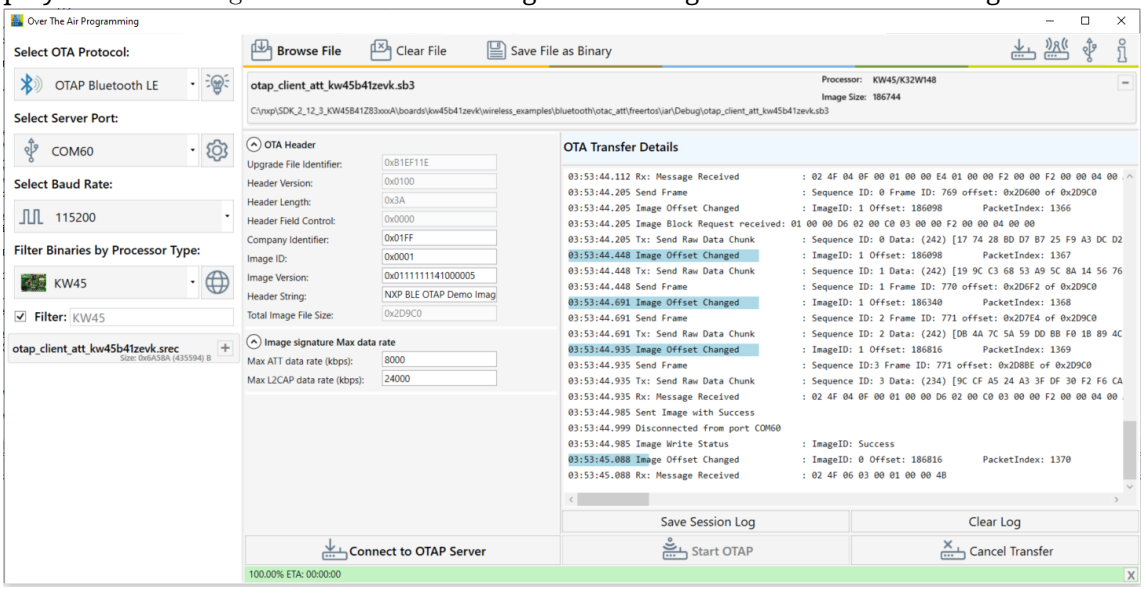
- Before starting the image transfer process, the data rate must be configured for each transfer method (ATT or L2CAP CoC). The image chunks of a block are sent over the serial interface and over-the-air without waiting for confirmation. Data rate can significantly slow down if configuration is not done correctly and errors appear in the transfer process.

  The optimal data rate depends on multiple factors. Some of these factors are listed below:
  - Distance between boards
  - Type of antenna
  - Performance of the RF circuitry between the radio and antenna
  - Type and level of noise in the environment
  - Speed of the storage medium in which the image is saved on the OTAP Client
  - Serial driver delay between PC and the OTAP Server board If the data rate is too high, then the OTAP Client receives a new chunk before it can process the previous one. In such a case, it sends an "**Unexpected Chunk Sequence Number**" error and restarts the transfer of the current block from where it left off. If the channel is too noisy, the transmitter can be flooded and some chunks might not reach the client triggering a similar type of error. The default data rate values should work for most configurations.

7. Start the embedded applications by pressing **ADVSW** first on the OTAP Client and then on the OTAP Server. The transfer progress and transfer-related messages and/or errors are shown in the application window. The duration of the transfer depends on the size of the image and the chosen data rate and transfer method. See the figure below.



8. After all the blocks are sent, the OTAP Client sends an Image Transfer Complete command to the OTAP Server. When the PC Application receives this command, it displays a Sent Image with Success message in the log window. See the figure below.



9. After the image transfer is complete, the OTAP Client triggers the bootloader and resets the MCU. The bootloader takes about 30 seconds to flash the image on the board. After this time frame, the MCU resets again and runs the new image.

Parent topic: Over the Air Programming (OTAP)

Storage type selection OTAP Clients support both internal and external storage. By default, the applications come with internal storage enabled. To configure external storage, follow the steps bellow:

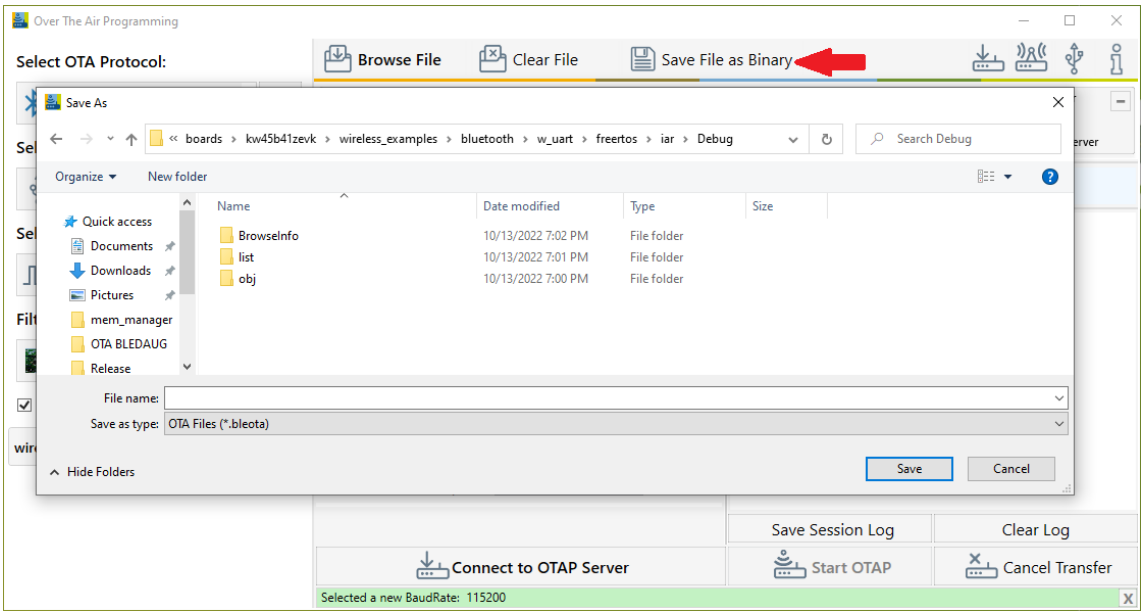
1. Set the gUseInternalStorageLink_d define to 0 in the project settings. For GitHub ARMGCC, replace CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.no_flash_selected=y with CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.external_flash=y in the application prj.conf file.
2. In app_preinclude.h, set the gAppOtaExternalStorage_c define to 1.

Usage with IoT Toolbox This is the list of requirements.

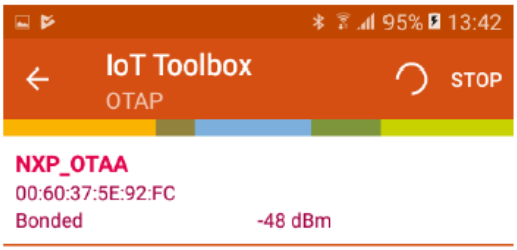
- **Mobile device** running **Android** platform or **iOS** with hardware and software supporting **Bluetooth 4.0 and later**.
- **Kinetis Bluetooth LE Toolbox** application – download from the specific application store for your device.

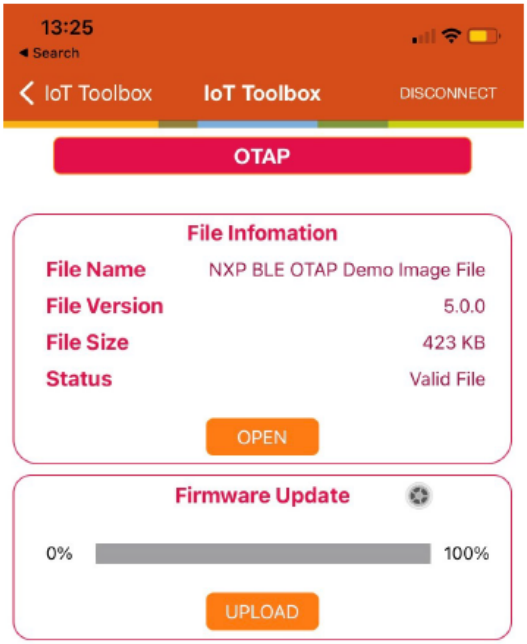
To run the application, perform the following steps:

1. Flash the OTAP Client ATT to either the KW45B41Z-EVK or the FRDM-MCXW71 platform. The Kinetis Bluetooth LE Toolbox only supports the ATT OTAP Client.
2. In order to send over the air in .bleota format, create the application. In order to load the image file into the Over the Air Programming application and create the .sb3 file, follow the instructions described in Usage with Over The Air Programming Tool. Once the .sb3 file is created, press the "**Save File as Binary**" button to create the .bleota file. See the figure below.

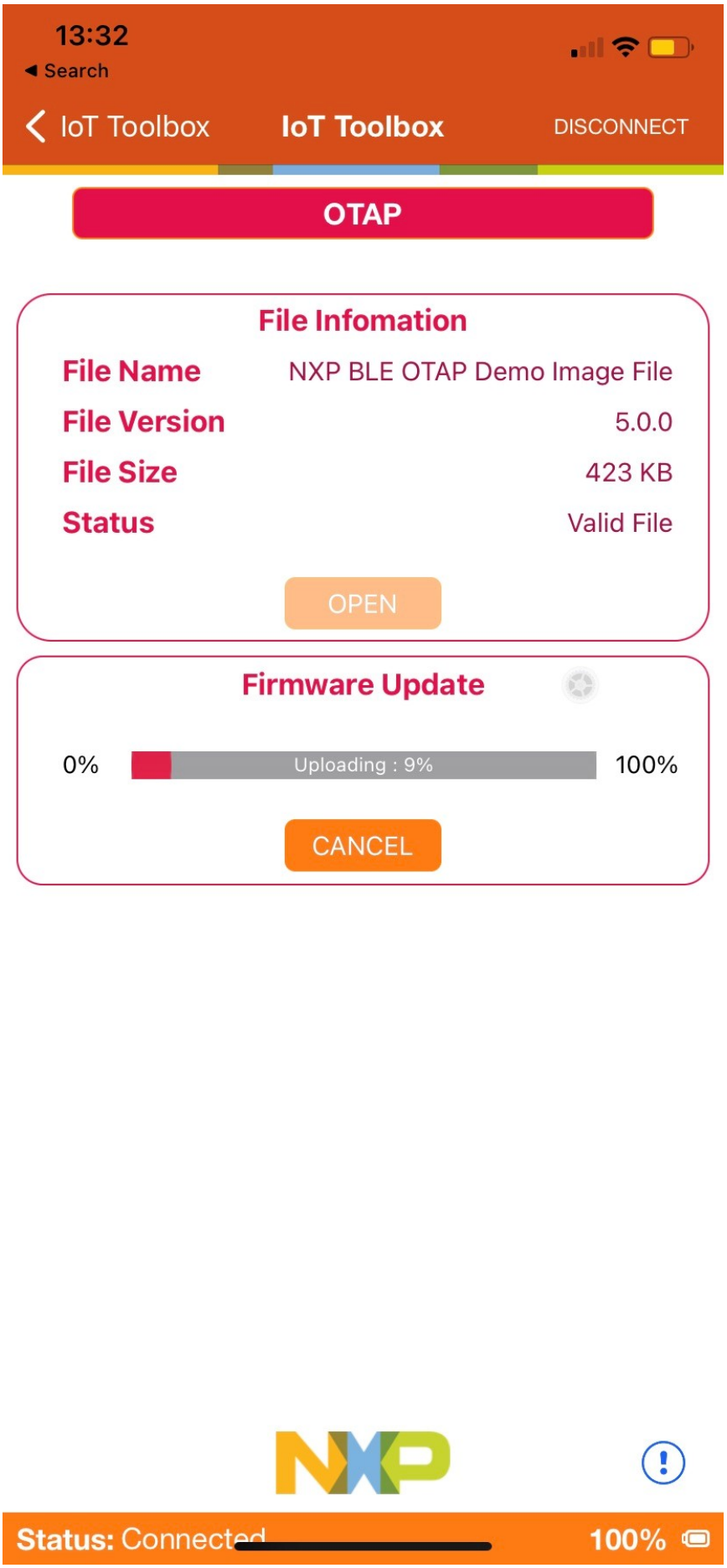


3. Start the **Kinetis Bluetooth LE Toolbox**application on your mobile device and start the **OTAP Tool**. The application starts scanning.
4. Press **ADVSW** on the board to start Advertising on the embedded OTAP Client application. The device should show up in the list of scanned devices. Touch the device in the scan list to connect to and the application performs service discovery and displays some information shown in the figure below.





5. Press the "**Open**" button and load the .bleota file to be sent over-the-air. Once the file is loaded, some information about it is displayed. Press the "**Upload**" button to start the image transfer process. A progress bar is shown while the image transfer is ongoing. The progress bar displays 100% update after a successful transfer, as shown in the figure below.


13:32
Search
IoT Toolbox
IoT Toolbox
DISCONNECT
OTAP
File Infomation
File Name
NXP BLE OTAP Demo Image File
File Version
5.0.0
File Size
423 KB
Status
Valid File
OPEN
Firmware Update
0%
Uploading : 9%
100%
CANCEL
Status: Connected
100%

6. After the image transfer is complete, the OTAP Client triggers the bootloader and resets the MCU. The bootloader takes about 30 seconds to flash the image on the board. After this time passes, the MCU resets again and runs the new image.

Parent topic: Over the Air Programming (OTAP)

Parent topic:*Bluetooth LE stack and demo applications*

Wireless UART This section describes the implemented profiles and services, user interactions, and testing methods for the Wireless UART application.

Implemented profile and services The Wireless UART application implements both the GATT client and server for the custom Wireless UART profile and services.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the ROLESW button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

Parent topic: Wireless UART

Supported platforms The following platforms support the Wireless UART application:

- KW45B41Z-EVK
- KW45B41Z-LOC
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- KW47-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- MCXW72-LOC
- FRDM-MCXW23
- MCXW23-EVK

Parent topic: Wireless UART

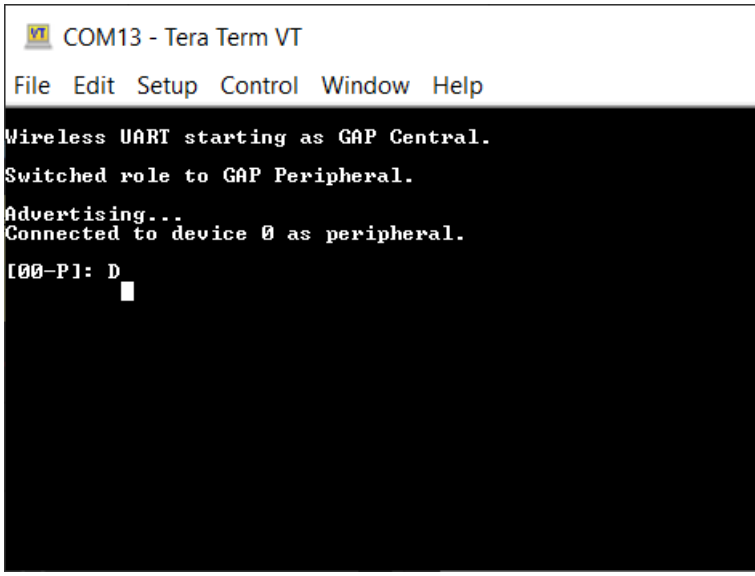
User interface After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic: Wireless UART

Usage The application is built to work with another supported platform running the same example or with the Wireless UART from the IoT Toolbox application.

When testing with two boards, perform the following steps:

1. Open a serial port terminal and connect them to the two boards, in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the role switch. Depending on the role, when pressing the **SCANSW**, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. See the figure below.



Tera Term – received text on Wireless UART

When testing with a single board and the IoT Toolbox, perform the following steps:

1. Open a serial port terminal and connect the board in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. Press the role switch button to behave as a GAP peripheral and then press the **SCANSW** button to start advertising. The IoT Toolbox app can then connect. Select UART instead of Console and start typing, as shown in the Figure.

Parent topic: Wireless UART

Parent topic:*Bluetooth LE stack and demo applications*

Bluetooth LE Shell This section describes the functionality, user interactions, and testing methods for the Bluetooth LE Shell Application.

Implemented stack features The Bluetooth LE Shell Application implements a console application that allows the user to interact with a full feature Bluetooth Low Energy stack library. It implements All GAP roles and both GATT client and server. Enabling these roles can be done using shell commands.

Parent topic: Bluetooth LE Shell

Implemented profile and services The application implements a dynamic GATT database. The user can add services at runtime and also erase the database contents. The database is always populated with the GAP and GATT services. These services cannot be erased. The user can dynamically add the following services:

- Heart Rate Service (UUID: 0x180D)
- Battery Service (UUID: 0x180F)
- Device Information Service (UUID: 0x180A)
- Internet Support Profile Service (0x1820)

Parent topic: Bluetooth LE Shell

Supported platforms The following platforms support the Bluetooth LE Shell application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

Parent topic: Bluetooth LE Shell

User interface After flashing the board, the device is in idle mode. The interaction with the board is done entirely by using the shell commands via the serial communication terminal.

Parent topic: Bluetooth LE Shell

Usage The application is built to work with any other Bluetooth LE device. To showcase the functionality, two platforms are used in the following setup.

1. Open a serial port terminal and connect them to the two boards, in the same manner described in Testing devices. The start screen is displayed after the board is reset. All LEDs are flashing on both devices.
2. Configure one of the devices as a GAP peripheral and a Heart Rate server. Change name to HRS. Start advertising on this device.

```
BLE Shell>gap devicename HRS
--> GATTDB Event: Attribute Written
HRS>gap advdata 1 6
HRS>gap advdata 8 HRS
HRS>gap advstart
HRS>
--> GAP Event: Advertising parameters successfully set.
HRS>
--> GAP Event: Advertising data successfully set.
HRS>
--> GAP Event: Advertising state changed successfully!
HRS>gattdb addservice 0x180D
--> Heart Rate
- Heart Rate Measurement Value Handle: 14
```

3. Configure the other device as a GAP central. Change its name to ‘Collector’. Start scanning and connect to the HRS device by selecting the corresponding device index from the list of scanned devices. In the example below, the HRS device is device number 2. The number of

listed scanned devices can be controller through the mShellGapMaxScannedDevicesCount_c define in shell_gap.c.

```
BLE Shell>gap devicename Collector
--> GATTDB Event: Attribute Written
Collector>gap scanstart filter
--> GAP Event: Scan started.
Collector>
--> GAP Event: Found device 0 : 880F102F500E 0 dBm
--> GAP Event: Found device 1 : NXP_CSCS 00049F000006 0 dBm
--> GAP Event: Found device 2 : HRS 00049F0000FF 0 dBm
Collector>gap connect 2
--> GAP Event: Scan stopped.
Collector>
--> GAP Event: Connected to peer 0
```

4. Optionally, the devices can be paired (gAppUsePairing_d and gAppUseBonding_d must be set in app_preinclude.h). On the collector initiate the pairing.

```
Collector>gap pair 0
--> Pairing...
--> GAP Event: Link Encrypted
--> GAP Event: Device Paired
```

5. On the Collector, start service discovery. The device discovers the GAP, GATT, and Heart Rate services.

```
Collector>gatt discover 0 -all
--> Discovered primary services: 3
--> Generic Attribute Start Handle: 1 End Handle: 4
- Service Changed Value Handle: 3
- Client Characteristic Configuration Descriptor Handle: 4
--> Generic Access Start Handle: 5 End Handle: 11
- Device Name Value Handle: 7
- Appearance Value Handle: 9
- Peripheral Preferred Connection Parameters Value Handle: 11
--> Heart Rate Start Handle: 12 End Handle: 19
- Heart Rate Measurement Value Handle: 14
- Client Characteristic Configuration Descriptor Handle: 15
- Body Sensor Location Value Handle: 17
- Heart Rate Control Point Value Handle: 19
```

6. Configure the HRS to send notifications by writing the CCCD from the Collector. Send a GATT write command with value 1 to the CCCD handle discovered, 15.

```
Collector>gatt write 0 15 0x0001
--> GATT Event: Characteristic Value Written!
```

7. Send heart rate measurement notifications from the HRS device by using the value handle obtained after adding the service in the previous step.

```
HRS>gatt notify 0 14
```

8. A notification appears on the Collector console.

```
Collector>
--> GATT Event: Received Notification
Handle: 14
Value: B400
```

Extended advertising Use the Bluetooth LE Shell application to exercise the advertising extension features:

On the GAP Peripheral device:

1. Configure the extended advertising parameters. In the below example, the advertising type is set to connectable and includes TX power and the primary PHY is set to Coded PHY.
2. Configure the extended advertising data. The Bluetooth LE Shell applications has the feature to send for test, a large data payload. Use the extended advertisement default configuration (not call "gap extadvcfg"), pass the command "gap extadvdata" with no parameters and the default data is added. The length is configurable at compile time through SHELL_EXT_ADV_DATA_SIZE and the data pattern is SHELL_EXT_ADV_DATA_PATTERN. Start the default test with call for "gap extadvstart".

   The advertising data type is set to shortened local name (8) and the advertising data content is set to test_ext_adv_data.

   Note: Users must note that extended connectable advertising does not allow for chained advertising data. The data length must be limited to what can fit in a single AUX_ADV_IND PDU (251 bytes at maximum). This means that passing the gap extadvdata with no parameters and the default value of SHELL_EXT_ADV_DATA_SIZE (500 bytes) after having set the advertising type to connectable will result in an error when trying to start advertising.
3. Start extended advertising.

```
BLE Shell>gap extadvcfg -type 65 -phy1 3
BLE Shell>gap extadvdata 8 test_ext_adv_data
BLE Shell>gap extadvstart
--> GAP Event: Extended
Advertising parameters successfully set.
--> GAP Event:
Extended Advertising data successfully set.
--> GAP Event: Advertising state changed successfully!
```

4. **On the GAP Central device**

   Set the scanning parameters. The scanning PHY is set to match the advertising PHY, in this case Coded PHY.
5. Start scanning and filter duplicates.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap scanstart filter
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : 0060375BCEC6 -23 dBm
Advertising Extended Data:
test_ext_adv_data
```

6. Set the connection initiating PHYs corresponding to the primary PHY on which the advertising is performed.
7. Connect to the desired device in the scanned devices list.

```
BLE Shell>gap connectcfg -phy 4
--> Connection Parameters:
--> Connection Interval: 200 ms
--> Connection Latency: 0
--> Supervision Timeout: 32000 ms
--> Connecting PHYs: Coded
BLE Shell>gap connect 0
BLE Shell>
-> GAP Event: Scan stopped.
BLE Shell>
--> GAP Event: Connected to peer 0
```

Parent topic: Usage

RSSI Monitor RSSI Monitor is an application that allows monitoring the RSSI of a remote peer on advertising or connection channel. The GAP peripheral device can modify the output TX power on both advertising and connection channels.

1. **On GAP peripheral device**

   Set the primary advertising PHY to Coded PHY. Start advertising and read the address. Set the TX power in dBm to a value less than 20 dBm.

```
BLE Shell>gap address
BLE Shell>
--> GAP Event: Public Address Read:C4603770BCC5
BLE Shell>gap extadvcfg -phy1 3
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>gap txpower adv 0
BLE Shell>
--> GAP Event: Success!
```

2. **On GAP Central device**

   Set the scanning PHY to Coded PHY. Start monitoring the RSSI on advertising Channel using the address of the Peripheral device. Scanning starts automatically, if it is not previously enabled.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap rssimonitor C4603770BCC5--> Reading RSSI on advertising channel:
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -29 dBm
```

   In the below example, the RSSI in monitored on a connection channel. On GAP Peripheral, start advertising in connectable mode on Coded PHY and adjust the TX power level.

```
BLE Shell>gap extadvcfg -type 65 -phy1 3
BLE Shell>gap extadvdata 8 rssimonitortest
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>
--> GAP Event: Connected to peer 0
BLE Shell>
--> GAP Event: Advertising stopped!
BLE Shell>gap txpower conn 10
```

(continues on next page)

(continued from previous page)

```
BLE Shell>
--> GAP Event: Success!
```

On the GAP Central device, start scanning on the Coded PHY. Update the connection PHY also to Coded PHY, then connect to the remote device and monitor continuously the RSSI on the connection channel.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap connectcfg -phy 4
-->  Connection Parameters:
    -->  Connection Interval: 200 ms
    -->  Connection Latency: 0
    -->  Supervision Timeout: 32000 ms
    -->  Connecting PHYs: Coded
BLE Shell>gap scanstart filter
BLE Shell>
->  GAP Event: Scan started.
BLE Shell>
-->  GAP Event: Found device 0 : C4603770BCC5 -21 dBm
 Advertising Extended Data:
rssimonitortest
gap connect 0
BLE Shell>
->  GAP Event: Scan stopped.
BLE Shell>
-->  GAP Event: Connected to peer 0
BLE Shell>gap rssimonitor 0 -c
-->  Reading RSSI from connected device:
BLE Shell>
 RSSI: -22 dBm
 RSSI: -23 dBm
 RSSI: -21 dBm
 RSSI: -22 dBm
 RSSI: -22 dBm
BLE Shell>gap rssistop
```

3. Update the PHY preference and continue monitoring the RSSI. For coded PHY, the coding scheme can be configured between S2 and S8 (500 kbit/s and 125 kbit/s).

```
BLE Shell>gap phy 0 -tx 4 -rx 4 -o 1
BLE Shell>
--> GAP Event: Phy update complete with peer 0
--> TxPhy: Coded
--> RxPhy: Coded

BLE Shell>gap phy 0 -tx 2 -rx 2
BLE Shell>
--> GAP Event: Phy update complete with peer 0
--> TxPhy: 2M
--> RxPhy: 2M

BLE Shell>gap rssimonitor 0 -c
--> Reading RSSI from connected device:
BLE Shell>
RSSI: -23 dBm
RSSI: -23 dBm
RSSI: -21 dBm
RSSI: -21 dBm
--> GAP Event: Phy update complete with peer 0
--> TxPhy: Coded
--> RxPhy: Coded
```

(continues on next page)

(continued from previous page)

```
BLE Shell>
RSSI: -22 dBm
RSSI: -21 dBm
RSSI: -22 dBm
RSSI: -23 dBm
RSSI: -23 dBm
--> GAP Event: Phy update complete with peer 0
--> TxPhy: 2M
--> RxPhy: 2M
BLE Shell>
RSSI: -22 dBm
RSSI: -21 dBm
RSSI: -21 dBm
RSSI: -20 dBm
```

Parent topic: Usage

Parent topic: Bluetooth LE Shell

Throughput feature The Bluetooth LE Shell application also has a throughput test feature that can be used to test different combinations of the parameters (connection interval, payload size, and packet count) to determine the best data-rate.

This feature requires two devices:

- GAP Peripheral: transmits the test packets
- GAP Central: receives the packets and displays a report

All throughput-related commands are grouped under the **thrput** keyword:

- thrput setparam: configures connection interval, packet count and payload size.
- thrput start tx: configures the device as a GAP Peripheral and starts advertising. Once the receiving device is connected, the packet transmission begins. The packet size and count can also be specified (-s<size_value>-c<count_value>).
- thrput start rx: configures the device as a GAP Central and starts scanning. Once a transmitter device is found, it connects to it and waits for the test to begin. The connection interval can also be configured (-ci<value>).
- thrput stop: stops the test and disconnects the devices.

Once a connection is established between the devices and initial throughput test is complete, one can start a new throughput transmission test with a new set of parameters (packet size / count).

The receiving device generates the report if no packets are received for more than three consecutive connection events.

The default configuration of the throughput test is the following:

- Packet count: 1000
- Payload size: 20 bytes

Connection interval (min, max): 160, 160 (200 ms)

The example of a test report is shown below:

```
BLE Shell>thrput start tx
BLE Shell>
-->  GAP Event: Advertising parameters successfully set.
BLE Shell>
```

(continues on next page)

(continued from previous page)

```
--> GAP Event: Advertising data successfully set.
BLE Shell>
--> GAP Event: Advertising started.
--> GAP Event: Connected to peer 0
BLE Shell>
--> GAP Event: Advertising Throughput test started.
Sending packets...
--> MTU Exchanged.
BLE Shell>
Throughput test with peer 0 has finished.

BLE Shell>thrput start rx
BLE Shell>
-> GAP Event: Scan started.
Found device:
THR_PER
0060375BCEC6
-> GAP Event: Scan stopped.
--> GAP Event: Connected to peer 0
BLE Shell>Throughput test started.
Receiving packets...


**********************************
***** TEST REPORT FOR PEER ID 0 ****
**********************************
Packets received: 1000
Total bytes: 244000
Receive duration: 5017 ms
Average bitrate: 389 kbps
**********************************
********** END OF REPORT ***********
```

Parent topic: Bluetooth LE Shell

Decision-Based Advertising Filtering (DBAF) feature The Bluetooth LE Shell application also supports the Decision-Based Advertising Filtering (DBAF) feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_DBAF_SUPPORT to 1.

This feature requires two devices:

- GAP Peripheral: transmits decision PDUs (ADV_DECISION_IND).
- GAP Central: scans for decision PDUs and handles filtering policies.

To showcase the functionality, two platforms are used in the following setup.

Steps to perform on the GAP Peripheral device:

1. Configure the extended advertising parameters to use decision PDUs. In the below example, the advertising type is set to connectable and includes TX power, uses decision PDUs and includes AdvA in the decision PDU. The primary PHY is set to Coded PHY.
2. Configure the extended advertising data using the gap extadvdecdata command. The resolvable tag and/or arbitrary data can be set using the parameters available.
3. Start extended advertising.

```
BLE Shell>gap extadvcfg -phy1 3 -type 449
BLE Shell>gap extadvdecdata -key 112233445566778899AABBCCDDEEFF00 -prand 5AC317 -decdata␣
↪6362 -datalen 2 -restag 0
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Extended Advertising Decision Data Setup Complete.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>
```

Steps to perform on the GAP Central device:

1. Set the scanning parameters to scan only decision PDUs. The scanning PHY is set to match the advertising PHY, in this case Coded PHY.
2. Set the connection parameters to use only decision PDUs and the connection initiating PHYs corresponding to the primary PHY on which the advertising is performed.

```
BLE Shell>gap scancfg -phy 4 -filter 12
BLE Shell>gap connectcfg -phy 4 -filter 2
--> Connection Parameters:
 --> Connection Interval: 200 ms
 --> Connection Latency: 0
 --> Supervision Timeout: 32000 ms
 --> Connecting PHYs: Coded
 --> Connection Filter Policy: 2
BLE Shell>
```

3. Add decision instructions using the gap adddecinstr command. A maximum of gMaxNumDecisionInstructions_c instructions can be added. If the set of instructions must be changed, the gap deldecinstr command deletes all current instructions.

```
BLE Shell>gap adddecinstr -group 1 -field 0 -criteria 1 -restagkey␣
↪112233445566778899AABBCCDDEEFF00
BLE Shell>gap adddecinstr -group 1 -field 6 -criteria 1 -advmode 6
BLE Shell>gap adddecinstr -group 0 -field 24 -criteria 1 -arbmask 00000000ffffff
BLE Shell>gap adddecinstr -group 0 -field 9 -criteria 1 -advacheck 2 -add1type 0 -add1 a6fb0d376000 -
↪add2type 0 -add2 a5fb0d376000
BLE Shell>gap adddecinstr -group 0 -field 7 -criteria 1 -rssimin -80 -rssimax 0
BLE Shell>gap adddecinstr -group 0 -field 8 -criteria 5 -lossmin 0 -lossmax 50
BLE Shell>
```

4. Set the decision instructions using the gap setdecinstr command. The instructions are used when listening for advertisements containing decision PDUs.
5. Start scanning and filter duplicates.
6. Connect to the desired device in the scanned devices list.

```
BLE Shell>gap setdecinstr
BLE Shell>
--> GAP Event: Decision Instructions Setup Complete.
BLE Shell>gap scanstart filter
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : C4603770BCC5 -23 dBm
 Advertising Extended Data:
gap connect 0
```

(continues on next page)

(continued from previous page)

```
BLE Shell>
-> GAP Event: Scan stopped.
BLE Shell>
--> GAP Event: Connected to peer 0
BLE Shell>
```

Parent topic: Bluetooth LE Shell

Periodic Advertising with Responses (PAwR) feature The Bluetooth LE Shell application also supports the Periodic Advertising with Responses feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_PAWR_SUPPORT to 1.

This feature requires two (or more) devices:

- GAP Broadcaster: transmits advertising PDUs at a fixed interval. (AUX_SYNC_SUBEVENT_IND)
- GAP Observer (one or more): scans for advertising PDUs and synchronizes with the advertising train. Sends responses PDUs in the synced subevents. (AUX_SYNC_SUBEVENT_RSP)

To showcase the functionality, two platforms are used in the following setup.

Steps to perform on the GAP Broadcaster device:

1. Configure the periodic advertising parameters. In the below example, a set of implicit values are used. There are 2 subevents per interval, each interval repeating at 1 second. There are 4 response slots configured per subevent, with the first response slot starting at 125ms from the advertising packet.

```
BLE Shell>gap periodiccfg

--> Periodic Advertising Parameters:
    --> Periodic Advertising Handle: 1
    --> Periodic Advertising Interval: 2000 ms
    --> Number of subevents: 2
    --> Interval between subevents: 156.25 ms
    --> Time between the advertising packet in a subevent and the first response slot: 125.0 ms
    --> Time between response slots: 1.250 ms
    --> Number of response slots: 4
BLE Shell>
--> GAP Event: Periodic Advertising parameters successfully set.
```

   Alternatively, the command does accept all the mentioned parameters to be changed. The below command can be used to configure the exact same values.

```
BLE Shell>gap periodiccfg -numsubevents 2 -subint 125 -rspslotdelay 100 -rspslotspace 10 -numrspslot 4
BLE Shell>
--> GAP Event: Periodic Advertising parameters successfully set.
```

2. Configure the subevent data on the advertiser for each subevent. Each instance of the gap periodicsubeventdata command sets each subevent data for the configured number of subevents. In this used example, the first command sets the data for the first subevent, the second command sets the data for the second subevent. The upper limit is the lowest of the following: SHELL_PER_ADV_MAX_NUM_SUBEVENTS from app_preinclude.h or the number of subevents set using gap periodiccfg.

```
BLE Shell>gap periodicsubeventdata 255 696E666F31
BLE Shell>gap periodicsubeventdata 255 696E666F32
```

When the last subevent data is set, any succeeding command will erase the previously set data for all subevents, and the data in the ongoing command will be set in the first subevent.

Alternatively, the command accepts more advertising data structures for one subevent, up to SHELL_EXT_ADV_DATA_MAX_AD_STRUCTURES.

```
BLE Shell>gap periodicsubeventdata 27 01CAFECAFECAFE 255 696E666F31
```

To manually clear the data set for all the currently set subevents, you may issue the following command:

```
BLE Shell>gap periodicsubeventdata -erase
```

3. Start extended advertising.

```
BLE Shell>gap extadvstart
BLE Shell>
-->  GAP Event: Extended Advertising parameters successfully set.

BLE Shell>
-->  GAP Event: Extended Advertising data successfully set.

BLE Shell>
-->  GAP Event: Advertising state changed successfully!
```

4. Start periodic advertising.

```
BLE Shell>gap periodicstart
BLE Shell>Periodic Advertising state changed successfully!
```

Steps to perform on the GAP Observer device:

1. Start scanning using the gap scanstart command. Optionally you can modify the scan parameters using the gap scancfg command.

```
BLE Shell>gap scanstart
BLE Shell>
->  GAP Event: Scan started.
```

2. Issue a periodic sync command with the target address of the Broadcaster device that is periodic advertising. You can use the gap address command on the Broadcaster to obtain the address.

```
BLE Shell>gap periodicsync -peer 006037A55E86
```

When the sync is established, events will be printed at the console.

```
-->  GAP Event: Periodic Advertising Sync Established
-->  GAP Event: Periodic V2 Device Scanned
       Event Counter: 8
       Subevent Synced: 0
       RSSI: -38 dBm
       Advertising Data: info1
```

3. Synchronize the Observer device with the subevents needed. For this example, we will synchronize the Observer with both subevents using the following command. Implicitly, the Observer synchronizes to subevent 0.

```
BLE Shell>gap periodicsyncsubevent -peradvproperties 0 -numsubevents 2 -subevents 0x00,0x01
...
BLE Shell>Set Sync Subevent command successfully completed.
```

After this, GAP Events of type Periodic V2 Device Scanned should be displayed in the console for each of the subevents synchronized.

```
--> GAP Event: Periodic V2 Device Scanned
      Event Counter: 169
      Subevent Synced: 0
      RSSI: -34 dBm
      Advertising Data: info1
--> GAP Event: Periodic V2 Device Scanned
      Event Counter: 169
      Subevent Synced: 1
      RSSI: -38 dBm
      Advertising Data: info2
```

4. The last step to have a complete exchange of data is to set the response(s) on the Observer device. The next 2 commands will set the data for the 2 subevents synced at the previous step.

```
BLE Shell>gap periodicresponsedata -responsesubevent 0 -responseslot 0 -data 255␣
↪64657669636531696E666F31
BLE Shell>gap periodicresponsedata -responsesubevent 1 -responseslot 0 -data 255␣
↪64657669636531696E666F32
```

   The -responsesubevent parameter indicates the subevent the data is set for. The -responseslot parameter indicates the response slot the data is sent at this value should be within the range 0 to the number of response slots set at periodiccfg.

   To scale the demo to use more Observer devices, repeat steps 1-3 for each of the added device. Also use example at step 4, with the adaptation of the -responseslot according to the device index. The demo is designed to have a new device for each of the response slots available. Also, change the data to reflect the response of the extra device.

   Note: *The* -data *parameter must be the last of the* periodicresponsedata *command as it can contain a volatile number of advertising data structures.*

   Alternatively, the command may contain additional advertising data structures.

```
BLE Shell>gap periodicresponsedata -responsesubevent 0 -responseslot 0 -data 27␣
↪01CAFECAFECAFE 255 64657669636531696E666F31
BLE Shell>gap periodicresponsedata -responsesubevent 1 -responseslot 0 -data 27␣
↪01CAFECAFECAFE 255 64657669636531696E666F32
```

   To manually clear the data set for all the currently set subevents, you may issue the following command:

```
BLE Shell>gap periodicresponsedata -erase
```

Initiate a connection from PAwR

PAwR feature allows for connection initiation from the Broadcaster device. To do this, the devices have to be synchronized. The minimum steps required for this are steps 1, 3, 4 on the GAP Broadcaster device and steps 1, 2, GAP Observer device.

After performing the mentioned steps, the Broadcaster device must issue the following command:

```
BLE Shell>gap connectpawr -subevent 0 -peer 00603728063E -peeraddrtype 0
```

Upon successful connection the following message will be displayed.

```
--> GAP Event: Connected to peer 0
```

Parent topic: Bluetooth LE Shell

Parent topic:*Bluetooth LE stack and demo applications*

Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK The Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK application demonstrates Generic FSK transmission/reception and Bluetooth advertising/scanning/multiple connections coexistence.

The Bluetooth LE part of this demo implements a modified version of the Wireless UART demo application, capable of multiple Bluetooth LE connections.

Based on the Hardware link-layer implementation, the Bluetooth Low Energy has a higher priority than the Generic FSK protocol and as the effect, the Generic FSK communication is executed during the Idle states (inactive periods) of the Bluetooth LE. The coexistence of the two protocols is handled internally at the Controller level.

The Bluetooth LE part of the application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the **ROLESW** button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

The Generic FSK part of the application can either enter in the receive state by double clicking the **SCANSW** button or it can start the periodic transmit by long pressing the **ROLESW** button. It cannot enter in both states at the same time.

The Generic FSK has lower priority than the Bluetooth LE. Therefore, any ongoing Generic FSK receive is paused by the Controller when Bluetooth LE activity is ongoing. The reception is automatically resumed by the Controller when there is no Bluetooth LE activity.

When using Generic FSK, packet loss should be expected. The Generic FSK communication does not offer an acknowledgement mechanism to guarantee success for packet transmission, therefore this communication method does not offer reliability. If the user wants to improve reliability, packet repetition may be used, or the user may implement a custom protocol at application level.

The first Generic FSK transmit command is buffered if there is continuous Bluetooth LE activity (for example, for continuous Bluetooth LE scanning). Any succeeding Generic FSK transmit command indicates failure in the command line interface, if the initial buffered transmit command was not sent yet.

This section describes the implemented profiles and services, user interactions, and testing methods for the Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK application.

Implemented profile and services The Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK application implements the GATT client and server for the custom Wireless UART profile and services. It also acts as a Generic FSK transmitter/receiver, repeating a custom packet, at a fixed periodic interval. It uses a predefined identifier, isolated to the address used in the Bluetooth LE protocol of the demo.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application is ready to start either Bluetooth LE scanning for Wireless UART Service, Bluetooth LE advertising Wireless UART Service, Generic FSK periodic transmit of a custom packet or Generic FSK receive, in the available slots not used by the Bluetooth LE protocol.

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Supported platforms The following platforms support the Hybrid (dual-mode) Bluetooth Low Energy and Generic FSK application:

- KW45B41Z-EVK
- KW45B41Z-LOC
- KW47-EVK
- KW47-LOC

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

User interface After flashing the board, the device is in idle mode (all LEDs flashing). To start Bluetooth LE scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

To start Generic FSK receiving, double click the **SCANSW** button and the device starts receiving packets on the same channel as the Bluetooth LE channel 37, with the network address defined in gGenFSK_NetworkAddress_c. The receiver only prints the packets that have the predefined identifier at gGenFSK_Identifier_c. To stop Generic FSK reception, double click the **ROLESW** button.

On a different device, start Generic FSK transmit by long pressing the **ROLESW** button. To stop the periodic Generic FSK transmit, long press again the **ROLESW** button, if the transmit procedure is ongoing. The long press **ROLESW** acts as a toggle for the transmit.

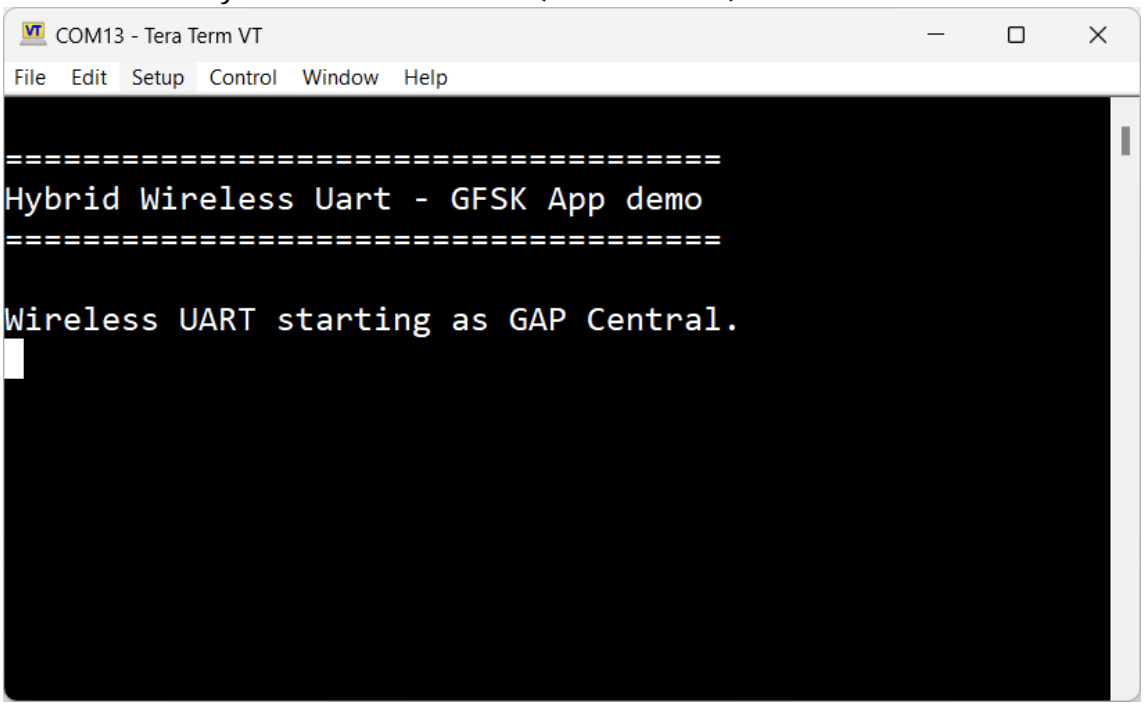
Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Usage The application is built to work with another supported platform running the same example. When testing with two boards, perform the following steps:

Case 1 (Bluetooth LE)

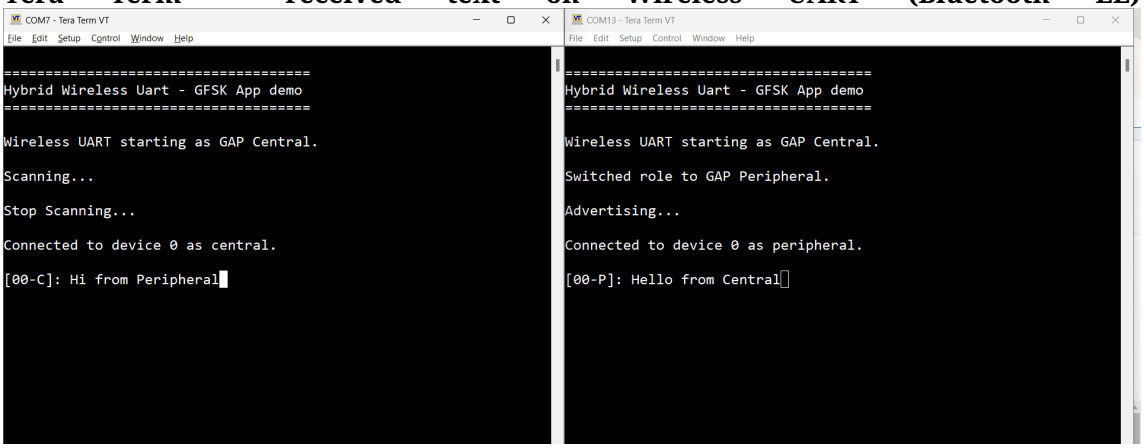
1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The figure below displays the start screen after the board is reset.

   Tera Term – Hybrid Wireless UART (Bluetooth LE) - Generic FSK start screen



2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the **ROLESW** button. Depending on the role, when pressing the **SCANSW** button, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. An output example can be observed in the figure below.

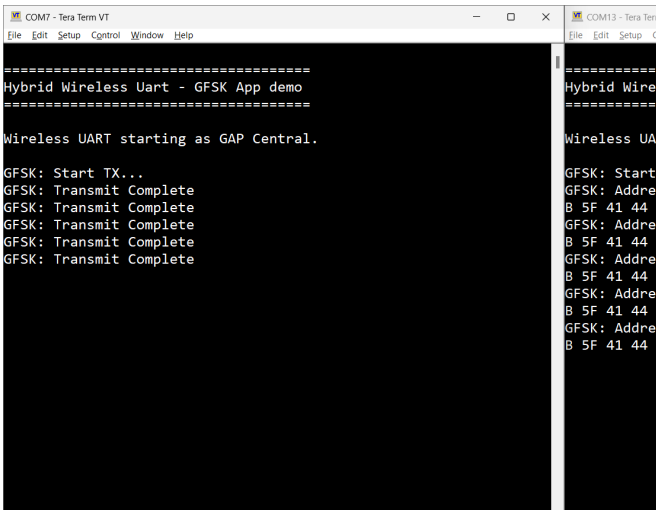
Tera Term – received text on Wireless UART (Bluetooth LE)



Parent topic: Usage

Case 2 (Generic FSK) When operating the Generic FSK mode, the following steps should be followed.

1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The start screen after the board is reset is the same as in Figure.
2. The Generic FSK communication direction is not preset. To start receiving, double click the **SCANSW** button on one board. Then, the device starts receiving packets on the same channel as the Bluetooth LE channel 37.
3. To start transmitting, long press the **ROLESW** button on the other board. The transmitting device uses an identifier known by the receiver and its packets are displayed in the CLI as shown in Figure:

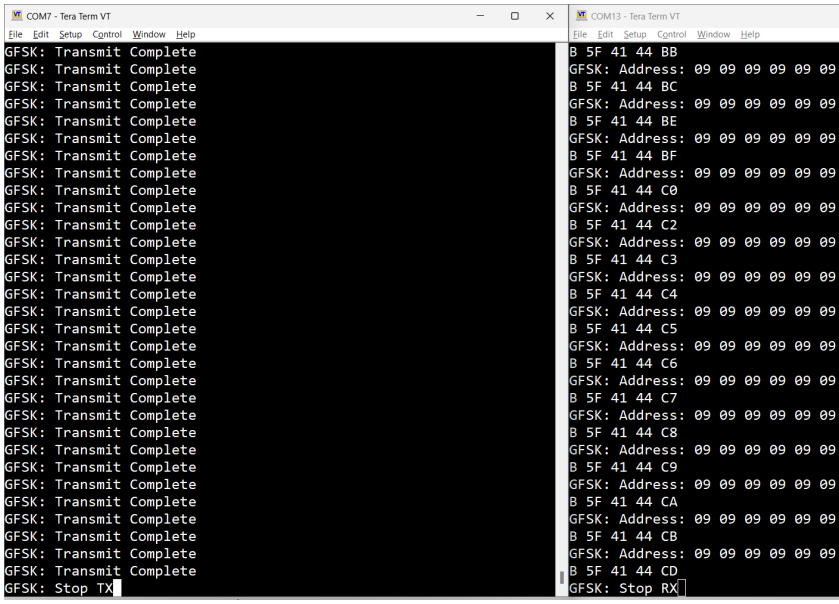


Teraterm – packet transmit/receive on Generic FSK

4. To stop Generic FSK reception, double click the **ROLESW** button.

5. To stop the periodic Generic FSK transmit operation, long press the **ROLESW** button again, if the transmit procedure is ongoing. The long press of the **ROLESW** button acts as a toggle for the transmit. At this point, both devices can reverse the direction of communication by following the exact same steps.

   See Figure.



Tera Term – stop Generic FSK activity

Parent topic: Usage

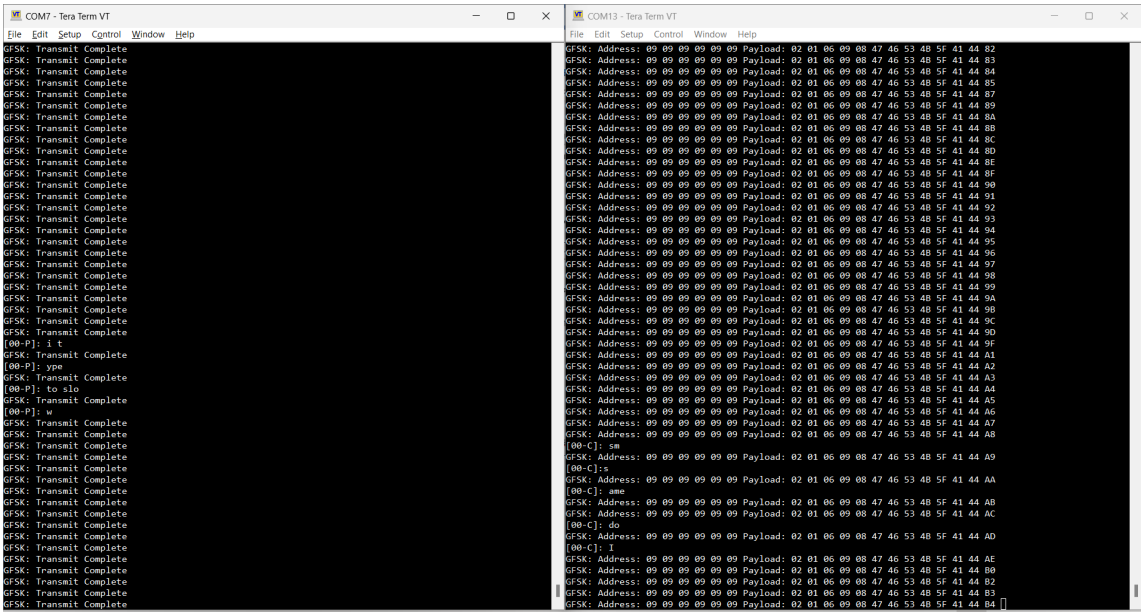
Case 3 (Dual-mode) The application is ready to combine both scenarios. It can either:

- Establish a Bluetooth LE connection and perform Generic FSK activity.
- During Generic FSK activity, the Bluetooth LE connection can be established and the Controller pauses receiving or announcing the discarded Generic FSK transmissions. This activity is resumed after the Bluetooth LE activity is finished.

To run the Dual-mode scenario, follow the steps below:

1. Establish a Bluetooth LE connection as described in Case 1 (Bluetooth LE).
2. Start Generic FSK activity as described in Case 2 (Generic FSK), independent of the Bluetooth LE roles chosen in Case 1.
3. Start typing in either of the terminals. As seen in Figure, the Bluetooth LE activity is prioritized. In this step, characters are printed in the peer's terminal and Generic FSK continues in the available slots, not used by the Bluetooth LE link.

   Teraterm – Mixed Wireless UART (Bluetooth LE) and Generic FSK activity



Parent topic: Usage

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Customization Use the steps below to change the default settings of this demo.

For Bluetooth LE, the default advertising config (gAdvParams) parameter is found in the app_config.c file. Also, the scanning parameters (gScanParams) can be found in this file.

The maximum number of connections supported by the Bluetooth LE, for this application, is defined in the app_preinclude.h file, with the macro name gAppMaxConnections_c.

Note: The Generic FSK protocol is active during the inactive periods of the Bluetooth LE protocol. The demo is currently configured to have the scan window equal to the scan interval to make the user aware of this, but this can be changed.

For Generic FSK, the following defines of interest can be found in genfsk_app.h, described below:

Name	Description
gGenFSK_Netwo	This is the network address used for the Generic FSK, in the transmitter payload. It is implicitly set to the 0x8E89BED6, but this can be reconfigured. Ensure that it is also changed on the receiver in the hybrid_gfsk.c controller file.
gGenFSK_H0Val	H0 value is used in the header.
gGenFSK_Identi	This is the identifier used by the transmitter to be filtered at the receiver. The current implementation filters the Generic FSK packets received, based on this define.
gGenFskApp_TxIn	This is the interval the transmitter will repeat the transmission of a packet. It is set in milliseconds.

Files of interest

The demo can be found in the w_uart_genfsk from the available examples.

The demo is based on the basic Wireless UART with the addition of some Generic FSK files required for working in dual-mode, described below.

File name	Description
genfsk_ap c	Application common module. Handles the HCI commands and events for the Generic FSK. Sends the events to the application.
genfsk_ap h	Application common module. Exposes public functions.
hybrid_gfs c	Controller common module. Handles initialization of Generic FSK.
hybrid_gfs h	Controller common module. Exposes public functions.

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Parent topic:*Bluetooth LE stack and demo applications*

Wireless UART Host This section describes the implemented profiles and services, user interactions, and testing methods for the Wireless UART Host application.

The Wireless UART Host application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport.

The Wireless UART Host application must be flashed together with the NCP FSCI Blackbox application.

Implemented profile and services The Wireless UART Host application implements both the GATT client and server for the custom Wireless UART profile and services.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the **ROLESW** button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

Parent topic: Wireless UART Host

Supported platforms The following platforms support the Wireless UART Host application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic: Wireless UART Host

User interface Ensure that both NCP FSCI Blackbox (for the NBU core) and Wireless UART Host (for the Application core) have been flashed to the board.

After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable

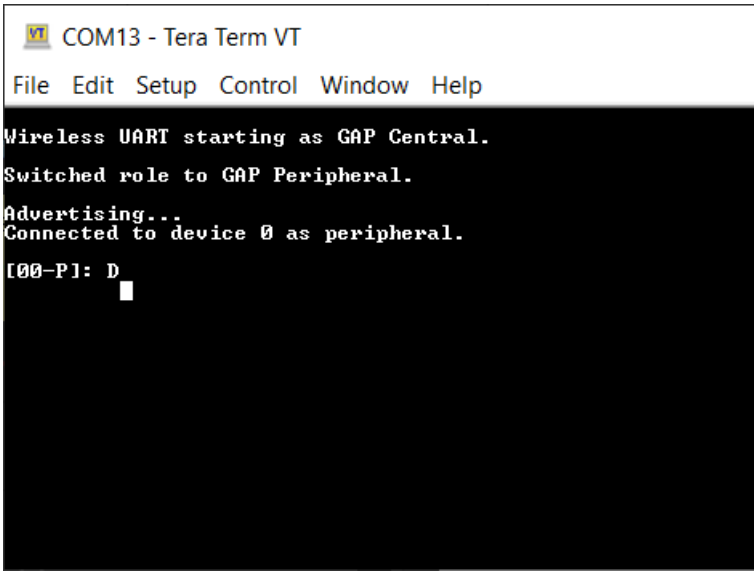
Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic: Wireless UART Host

Usage The application is built to work with another supported platform running either Wireless UART or Wireless UART Host, or with the Wireless UART from the IoT Toolbox application.

When testing with two boards, perform the following steps:

1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the role switch. Depending on the role, when pressing the **SCANSW**, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. See the figure below.



Tera Term – received text on Wireless UART

When testing with a single board and the IoT Toolbox, perform the following steps:

1. Open a serial port terminal and connect the board in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. Press the role switch button to behave as a GAP peripheral and then press the **SCANSW** button to start advertising. The IoT Toolbox app can then connect. Select UART instead of Console and start typing, as shown in the Figure.

Parent topic: Wireless UART Host

Parent topic:*Bluetooth LE stack and demo applications*

FSCI Bridge This section describes the functionality, user interactions, and testing methods for the FSCI Bridge demo application.

Description The FSCI Bridge demo application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport

The FSCI Bridge runs on the Application core and it is a simple application whose purpose is to pass FSCI commands received over the serial interface to the NCP FSCI Black Box running on the NBU core. Effectively, the functionality is identical to the regular Bluetooth FSCI Black Box application.

The demo can be used with the Test Tool for Connectivity Products - Command Console application which can be downloaded from the NXP website or using a custom application that supports the FSCI protocol and commands.

Parent topic:FSCI Bridge

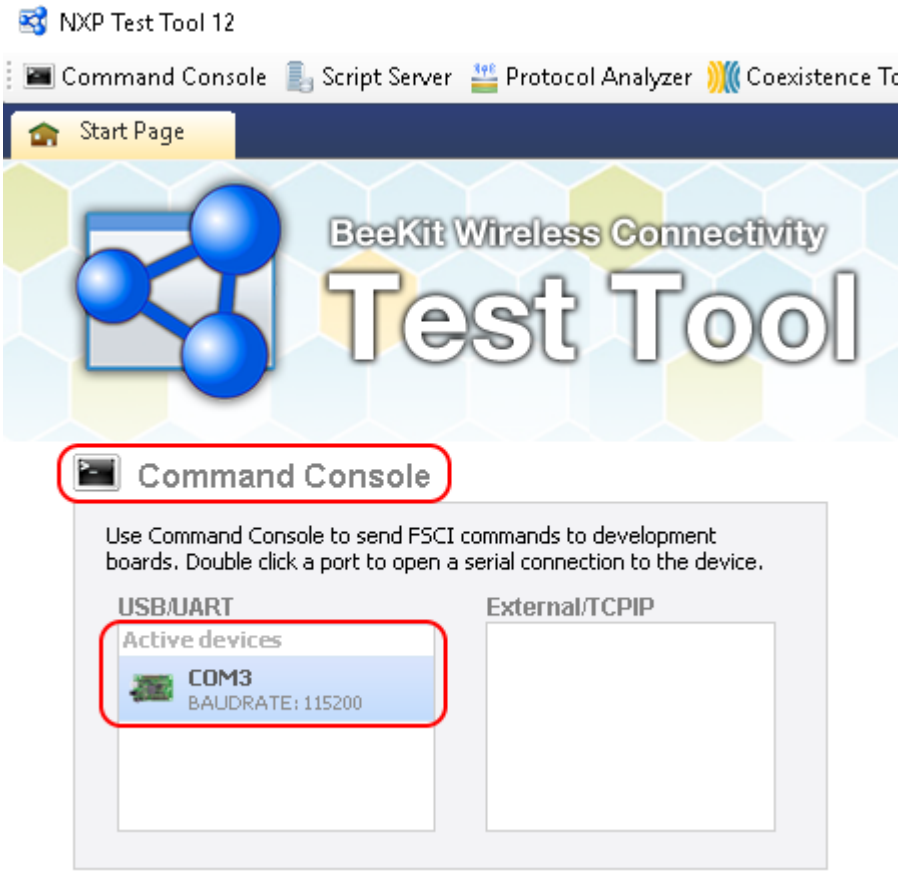
Supported platforms The following platforms support the FSCI Bridge application:

- MCX-W72-EVK
- FRDM-MCXW72

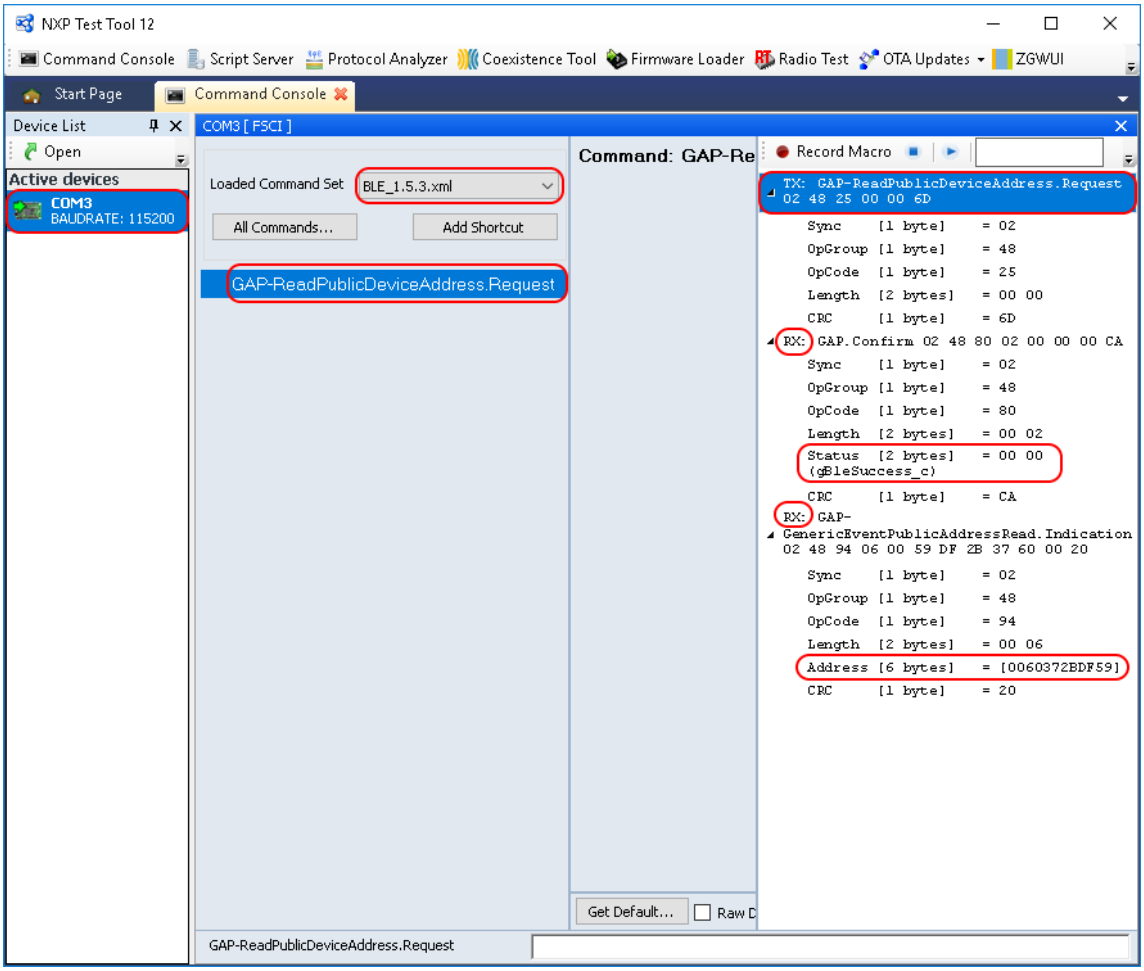
Parent topic:FSCI Bridge

Usage with Test Tool for connectivity products The FSCI Bridge demo application is designed to be used via serial interface. This can be done using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the FSCI Bridge application onto a supported board's Application core, together with the NCP FSCI Black Box application on the NBU core.
2. Connect the board to a USB port of the PC. The USB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the FSCI Bridge application runs. See Figure. The serial communication parameters are: baud rate 115200, 8N1, and no flow control.



4. Select the appropriate Test Tool XML file from the drop-down list for the release being used and send commands to the application. An example is shown in Figure.



Parent topic:FSCI Bridge

Parent topic:*Bluetooth LE stack and demo applications*

NCP FSCI Black Box

This section describes the functionality of the NCP FSCI Black Box demo application.

Description

The NCP FSCI Black Box application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport.

The NCP FSCI Black Box application runs on the NBU core and provides access to the Bluetooth LE Host Stack through FSCI to the application running on the Application core. It can be paired with the FSCI Bridge application to obtain a functionality that is identical to the regular Bluetooth FSCI Black Box application, or with the Wireless UART Host application to obtain a functionality that is identical to the regular Wireless UART application.

Parent topic:NCP FSCI Black Box

Supported platforms

The following platforms support the NCP FSCI Black Box application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic:NCP FSCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

References For more information, refer to the following documents:

- *Bluetooth Low Energy Application Developer's Guide*(KW45_K32W1_BLEADG)
- *Bluetooth Low Energy Host Stack API Reference Manual*(KW45_K32W1_BLEHSAPIRM)
- *Bluetooth Low Energy Host Stack FSCI*(Framework Serial Connectivity Interface) *API Reference Manual* (KW45_K32W1_BLEHSFSCIAPIRM)
- *Connectivity Framework Reference Manual* (KW45_K32W1_CONNFWRM)
- *Bluetooth Low Energy CCC Digital Key R3 Application Note* (AN12791) See AN12791 here.

Acronyms The following acronyms are used in this document.

Acronym	Description
ANCS	Apple Notification Center Service
ATT	Attribute Protocol
Bluetooth LE	Bluetooth Low Energy
CCC	Car Connectivity Consortium
CCCD	Client Characteristic Configuration Descriptor
DBAF	Decision-Based Advertising Filtering
EATT	Enhanced Attribute protocol
EVK	Evaluation Kit
FSCI	Framework Serial Connectivity Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
HID	Human Interface Device
HRS	Heart Rate Server
JSON	Javascript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
NVM	Non-volatile Memory
OTAP	Over the Air Programming
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RX	Receive
SDK	Software Development Kit
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2022-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy CCC Digital Key R3 Application Note

Introduction NXP provides hardware platforms to implement Bluetooth Low Energy applications implementing the Car Connectivity Consortium (CCC) Digital Key R3 functionality. This document provides an overview of the deployment and operation of such applications. This document is also a user guide for the sample applications included in the SDK release package.

Audience This document is for firmware and system developers who create CCC Digital Key-enabled products. The document also provides a high-level description of CCC Digital Key application scenarios that can be deployed on NXP development boards.

Parent topic:*Introduction*

CCC Digital Key technology overview The Car Connectivity Consortium (CCC) is a cross-industry organization advancing global technologies for smartphone-to-car connectivity solutions. CCC has developed Digital Key, a new open standard to allow smart devices such as smartphones to act as a vehicle key. Digital Key lets drivers lock and unlock their cars and even allows them to start the engine and share access to friends or valets, using their phones.

The Digital Key Release 3.0 specification enhances Digital Key Release 2.0 by adding passive, location-aware keyless access. Rather than having to pull their mobile devices out to access a car, consumers are able to leave their mobile device in their bag or pocket when accessing and/or starting their vehicle. Passive access is not only vastly more convenient and a better overall user experience, it also allows vehicles to offer new location-aware features.

To support these new features, the CCC has developed a specification based on Bluetooth Low Energy in combination with Ultra-Wideband (UWB) to enable passive keyless access and to allow secure and accurate positioning.

CCC Digital Key applications overview **Note:** The applications described below implement only the Bluetooth Low Energy functionality described by CCC Digital Key R3. They do not include any UWB or Secure Element functionality. The establishment of secure communications between the demos is simulated using dummy messages.

Digital Key Car Anchor The Digital Key Car Anchor demo application implements one of the multiple Bluetooth Low Energy anchors that can reside inside the car. The anchor performs a dual role. Inside the CCC Digital Key scope, it is a Bluetooth Low Energy peripheral which, depending on the scenario, can perform both Legacy (1M PHY) and Extended Long Range (500 kbps Coded PHY) advertising, searching for a CCC Digital Key-enabled device such as a smartphone.

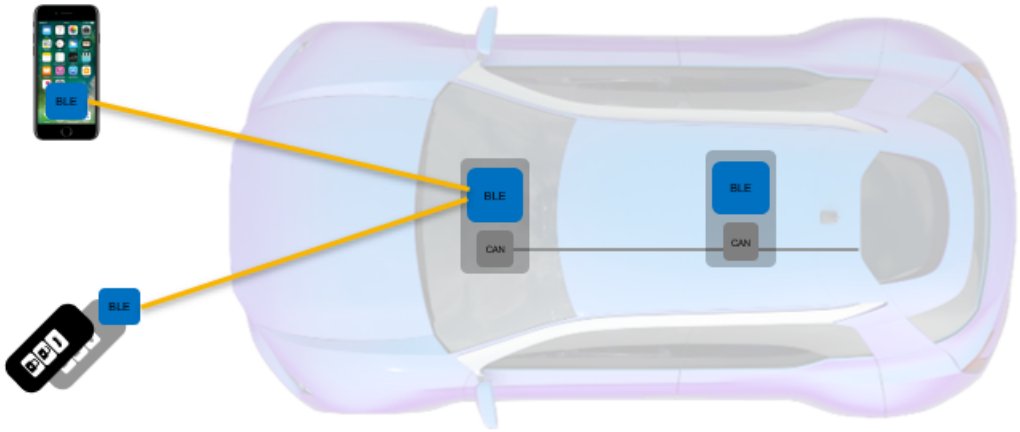
Simultaneously it can also act as a Bluetooth Low Energy central, scanning for non-CCC key fobs. There can be multiple anchors inside a car. The anchors must act as a single device and share information between them (addresses, bonding data, application specific keys, and such data) such that a smartphone can connect to any one of them depending on positioning and not detect any difference.

Parent topic:*CCC Digital Key applications overview*

Digital Key Device The Digital Key Device demo application emulates a CCC Digital Key-enabled smartphone. It acts as a Bluetooth Low Energy central device, scanning for advertising coming from a Digital Key Car Anchor.

The Figure 1 shows an example deployment of CCC Digital Key R3. Inside the car, there are two Bluetooth Low Energy anchors connected to each other. The Controller Area Network (CAN) bus connects these anchors and is used to exchange security data and other information. Any anchor can perform both advertising and scanning and it is able to connect to a CCC Digital Key R3-enabled smartphone as well as to a non-CCC key fob.

Anchor communicating with a smartphone (CCC) and a key fob (non-CCC)

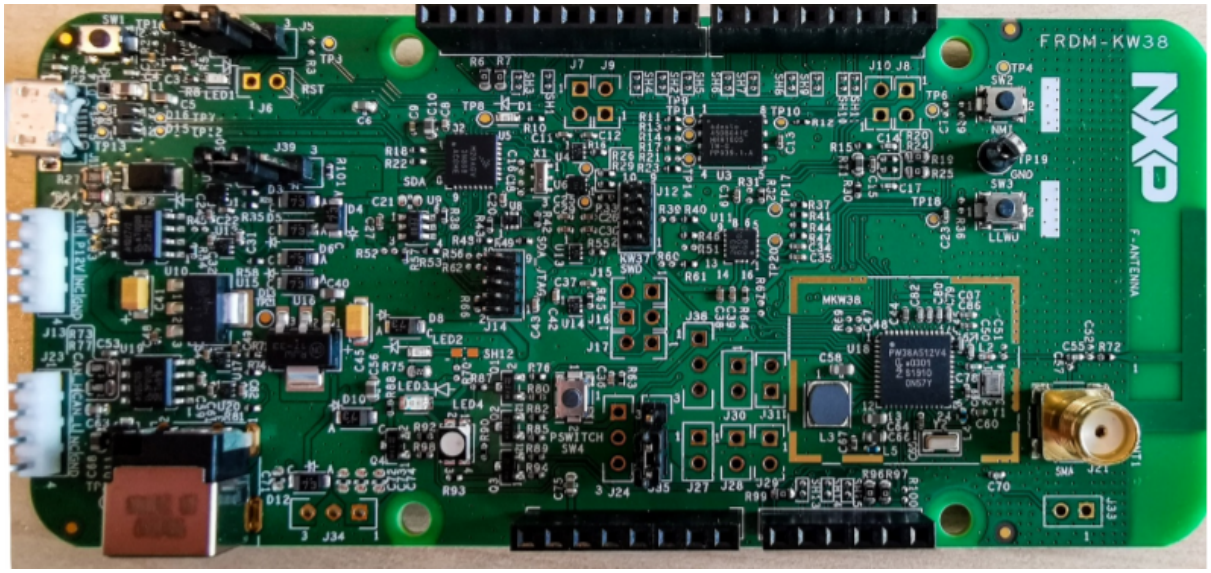


Parent topic:*CCC Digital Key applications overview*

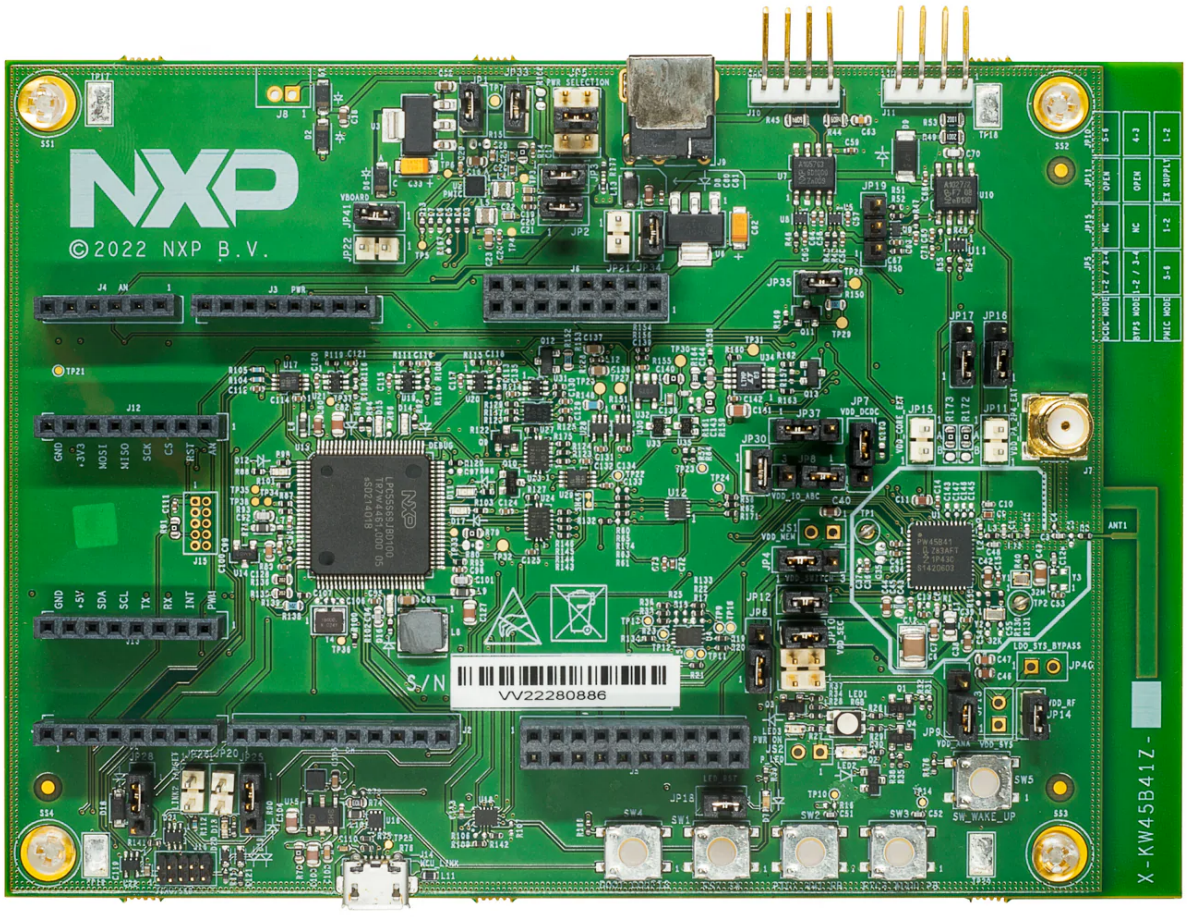
CCC Digital Key hardware platforms The CCC Digital Key R3 demo applications support the following platforms that have Bluetooth Low Energy transceiver capabilities:

- FRDM-KW38 (see Figure 1)
- KW45B41Z-EVK (see Figure 2)
- KW47-EVK (see Figure 3)
- KW47-LOC (see Figure 4)

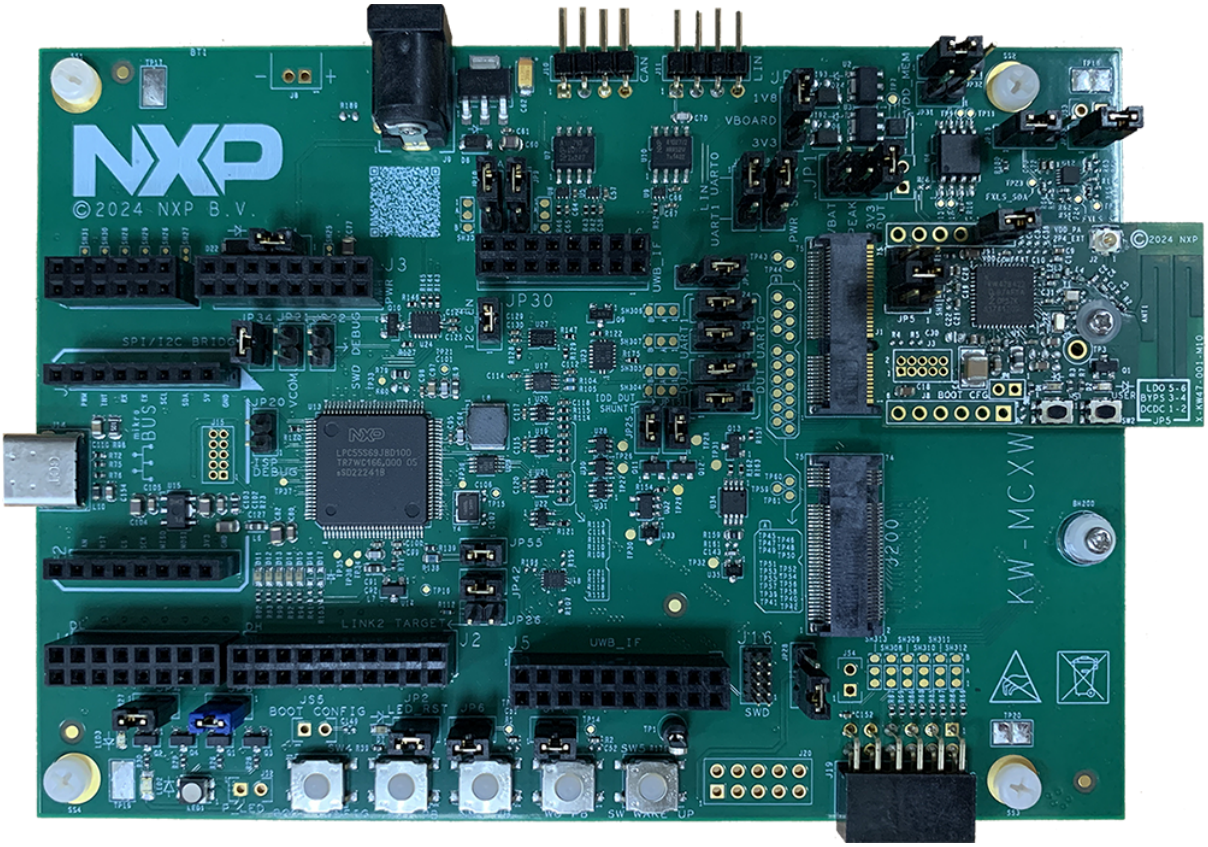
FRDM-KW38 board



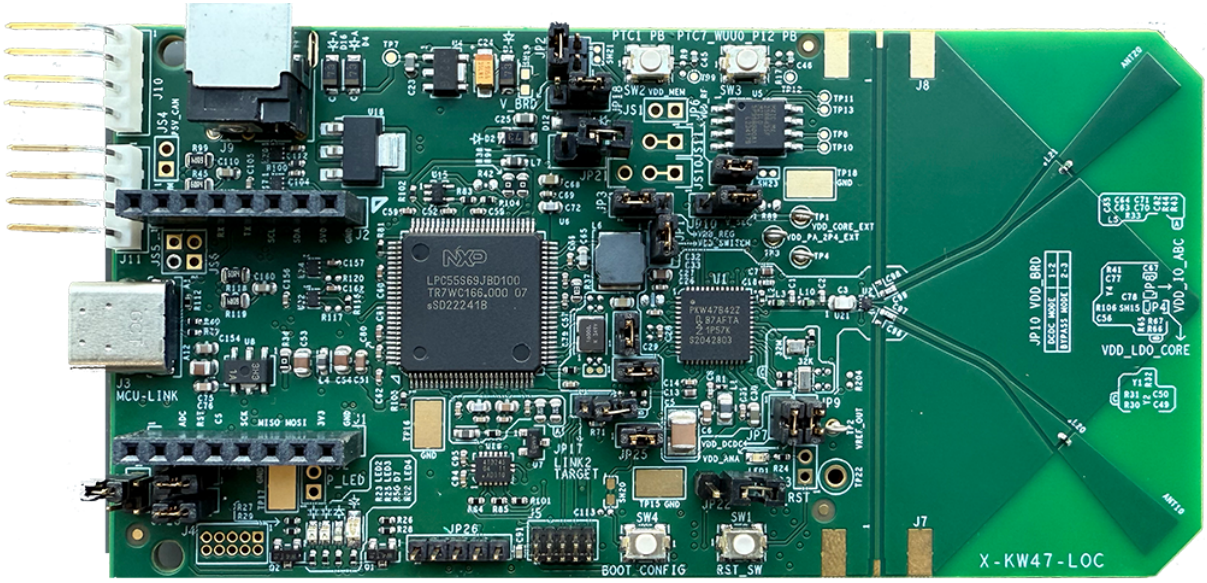
KW45B41Z-EVK board



KW47-EVK board



KW47-LOC board



Deploying CCC Digital Key applications software The software package includes the components necessary for CCC Digital Key R3 application development on the selected NXP platform. These components include:

- Bluetooth Low Energy Host and Controller software libraries (KW38)
- NBU image binaries
- Examples of demo application projects corresponding to CCC Digital Key roles (Device and Car Anchor)

- Platform peripheral drivers, platform startup code, other generic platform software

Note:

Prior to loading any wireless SDK example, update your NBU image with the provided binaries in the following folder of the SDK:

../middleware/wireless/ble_controller/bin.

The demo applications were compiled and tested with **IAR Embedded Workbench** for Arm and **MCUXpresso IDE**. It is recommended to use one of these tools.

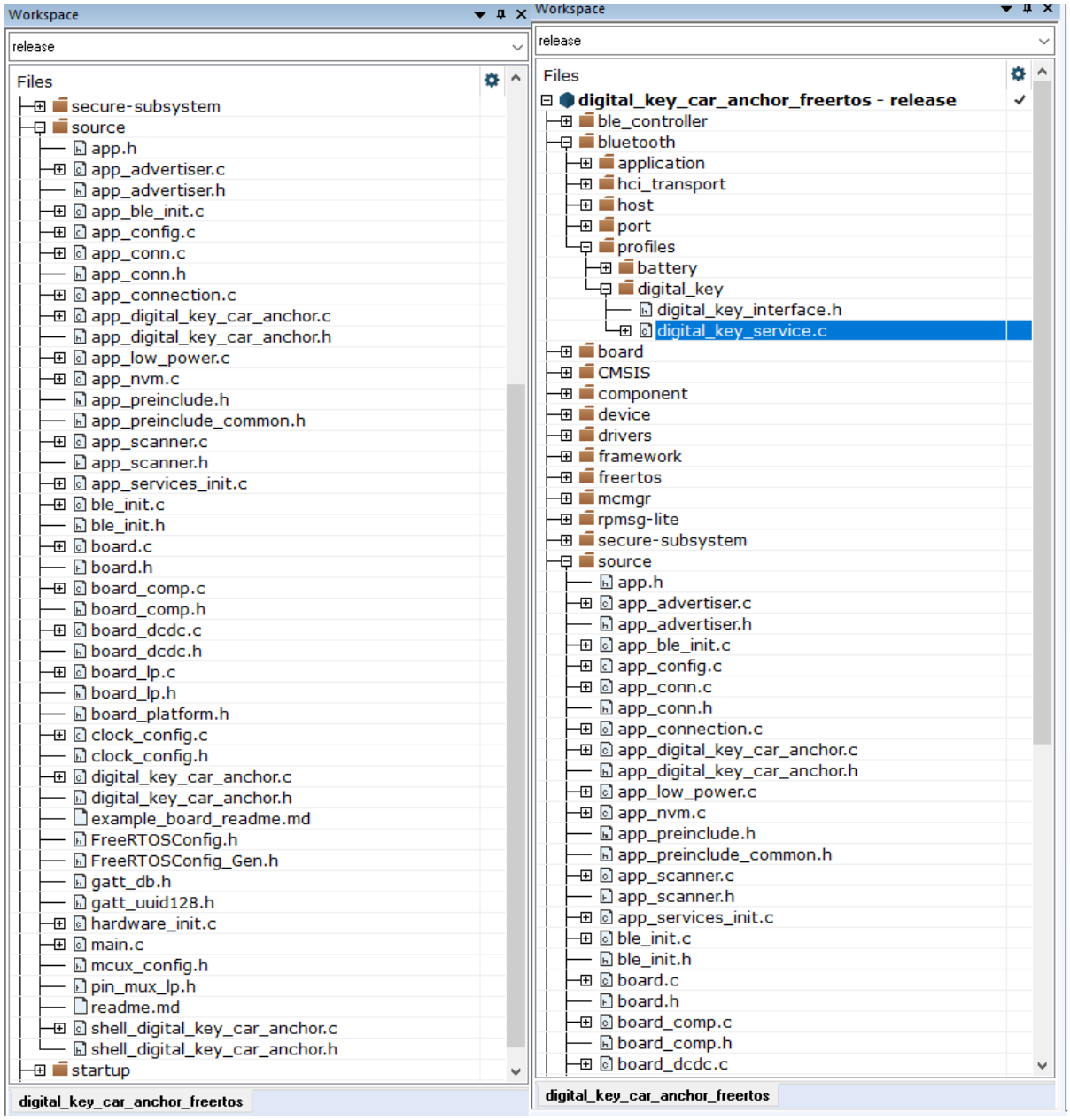
To open, build, and run any example application, see the *Bluetooth Low Energy Quick Start Guide* document of the corresponding board. The CCC Digital Key R3 examples are built and run in the same fashion as Bluetooth Low Energy examples.

The Figure 1 shows the application folder structure for the *digital_key_car_anchor* project.

The relevant files for CCC Digital Key R3 are the following:

- The main application source files source/digital_key_car_anchor.c and source/app_digital_key_car_anchor.c, and
- The files in bluetooth/profiles/digital_key, which implement the Digital Key Service defined by the CCC Digital Key R3 specification.

Application folder structure in workspace



Demo functionality overview The two demo applications provided showcase the following Digital Key R3 features:

- Owner Pairing between a Device and a Car Anchor
- Passive Entry
- Synchronizing bonding data between Car Anchors
- Connection between a Car Anchor and a non-CCC key fob

The demo applications also showcase the NXP proprietary Connection Handover and Anchor Monitoring features, as well as the A2B feature (secured synchronization of bonding data).

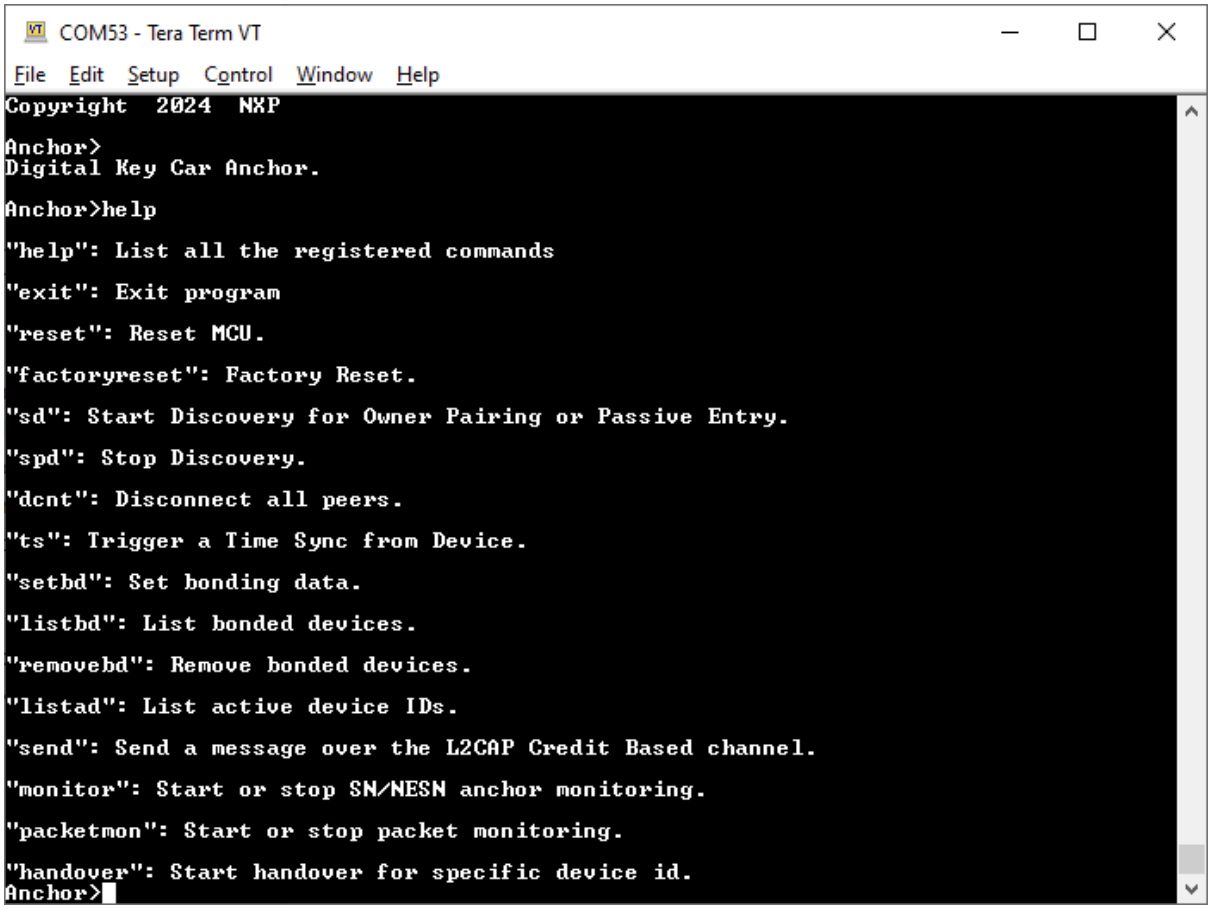
Running CCC Digital Key scenarios using the Shell Interface The following CCC Digital Key R3 examples are provisioned by default with a shell command line interface accessible via a Terminal application such as Tera Term:

- Digital Key Car Anchor

- Digital Key Device

After connecting through the Terminal application and pressing the RST switch on the board, a welcome message appears. Entering the command "help" displays the list of available commands and their descriptions. See Figure 1.

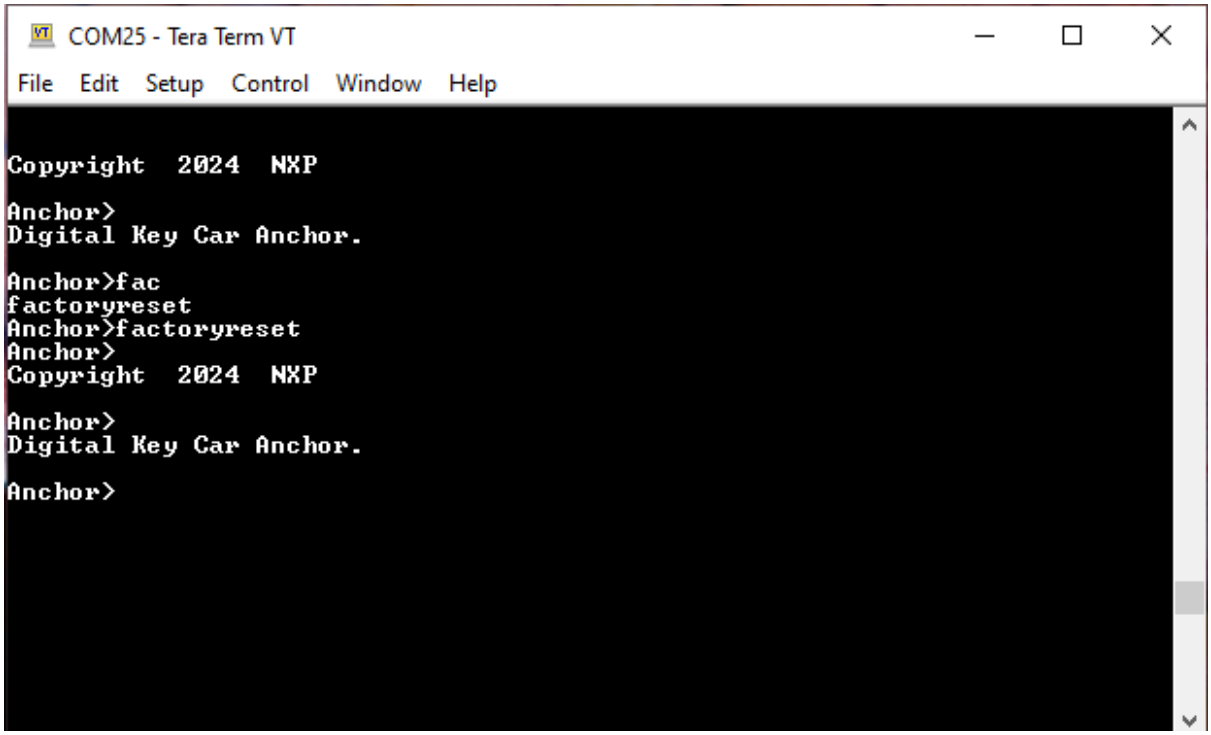
List of commands



Owner Pairing Scenario Owner Pairing establishes a bond between the Digital Key Car Anchor and the Digital Key Device as per the CCC Digital Key R3 specification. The Car Anchor starts advertising using Legacy mode on the 1M PHY. The Device scans and connects. After establishing the connection, the Device performs service discovery. This step enables it to discover the DK Service and learn the PSM value it would use in order to open an L2CAP channel to the Car Anchor. This channel is used to exchange data and security information as part of the Bluetooth Low Energy Out-of-Band (OOB) pairing process. All security information exchanged consists of dummy messages. The applications do not currently implement any of the UWB and Secure Element functionality described by CCC Digital Key R3.

The first step is to run the "factoryreset" command on both boards in order to ensure no previous bonding data is present in non-volatile memory. See Figure 1.

Factory reset on Car Anchor



The next step is to run the "sd" command (Start Discovery) on the Device and the "sd op" command on the Car Anchor (Owner Pairing advertising differs from Passive Entry, using a single Legacy advertising set on the 1M PHY). The Car Anchor starts advertising while the Device scans. Since both boards have been factory reset, they follow the Owner Pairing flow. The output of the commands is shown in Figure 2.

Owner Pairing on Car Anchor

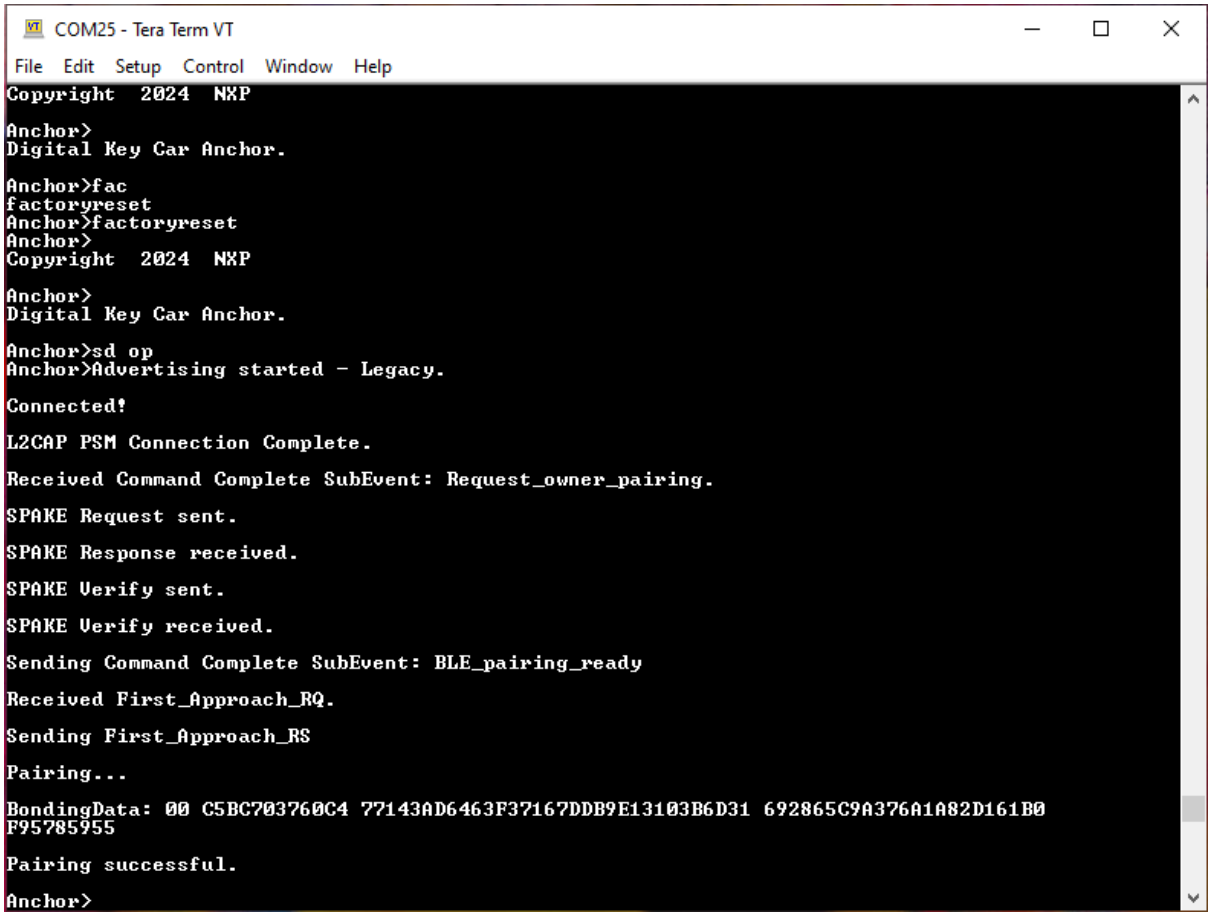
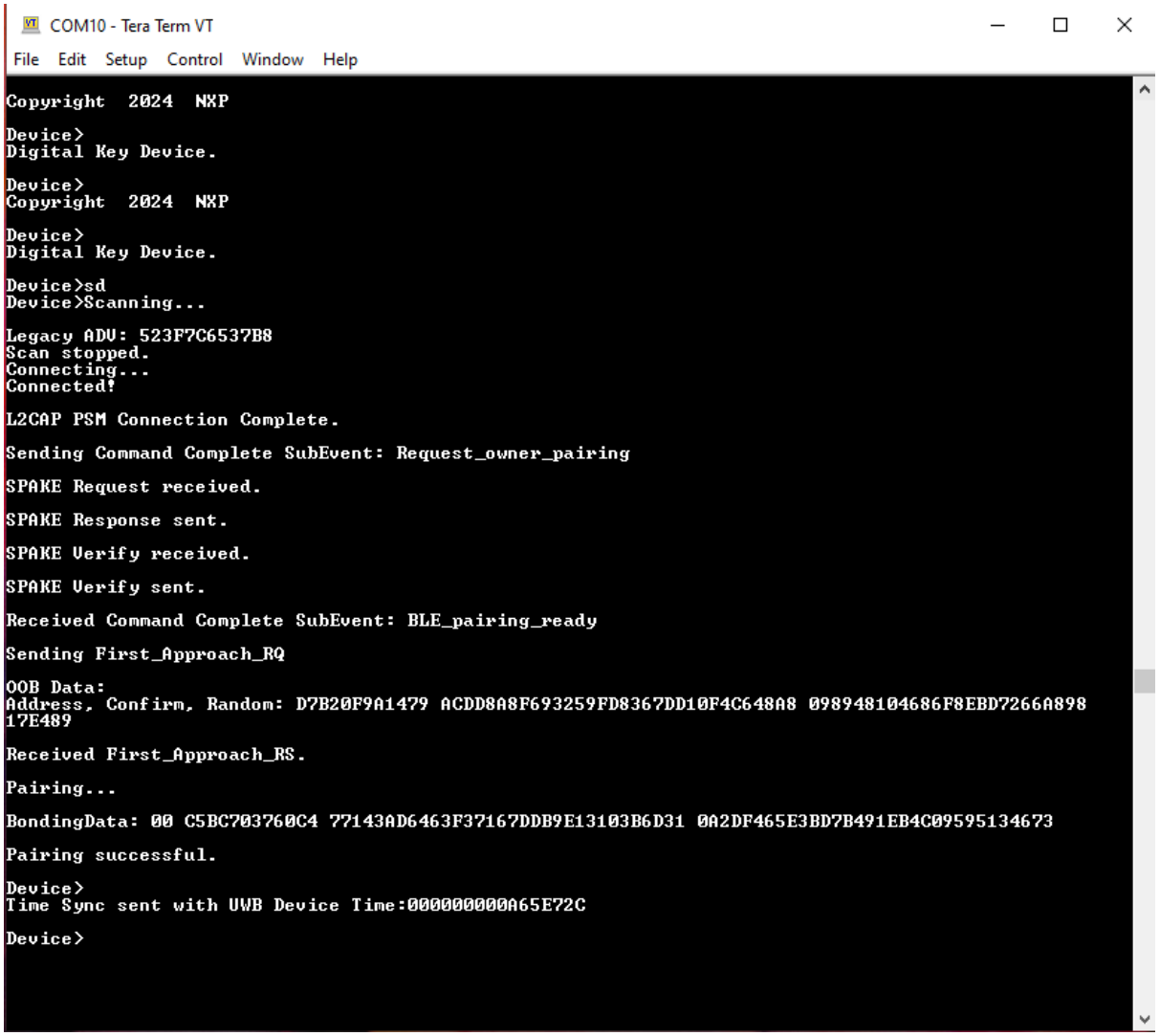


Figure 3 shows the owner pairing on the Device.

Owner Pairing on Device



Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

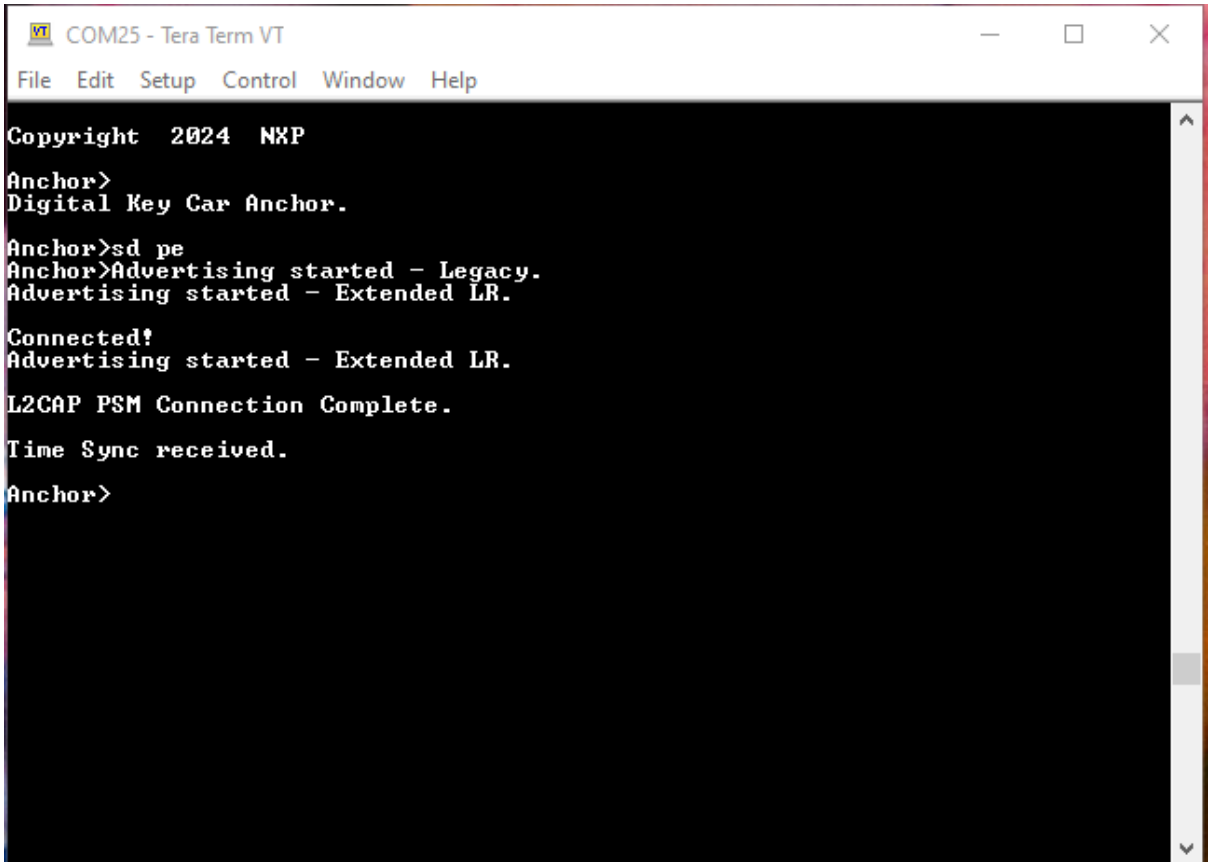
Passive Entry Scenario The Passive Entry scenario runs after Owner Pairing between a Car Anchor and a Device completes successfully.

To run the Passive Entry demo, first, reset the recently paired boards using either the **RST** switch or the "reset" command in the shell. Do NOT perform a factory reset, as this removes the bonding information from non-volatile memory.

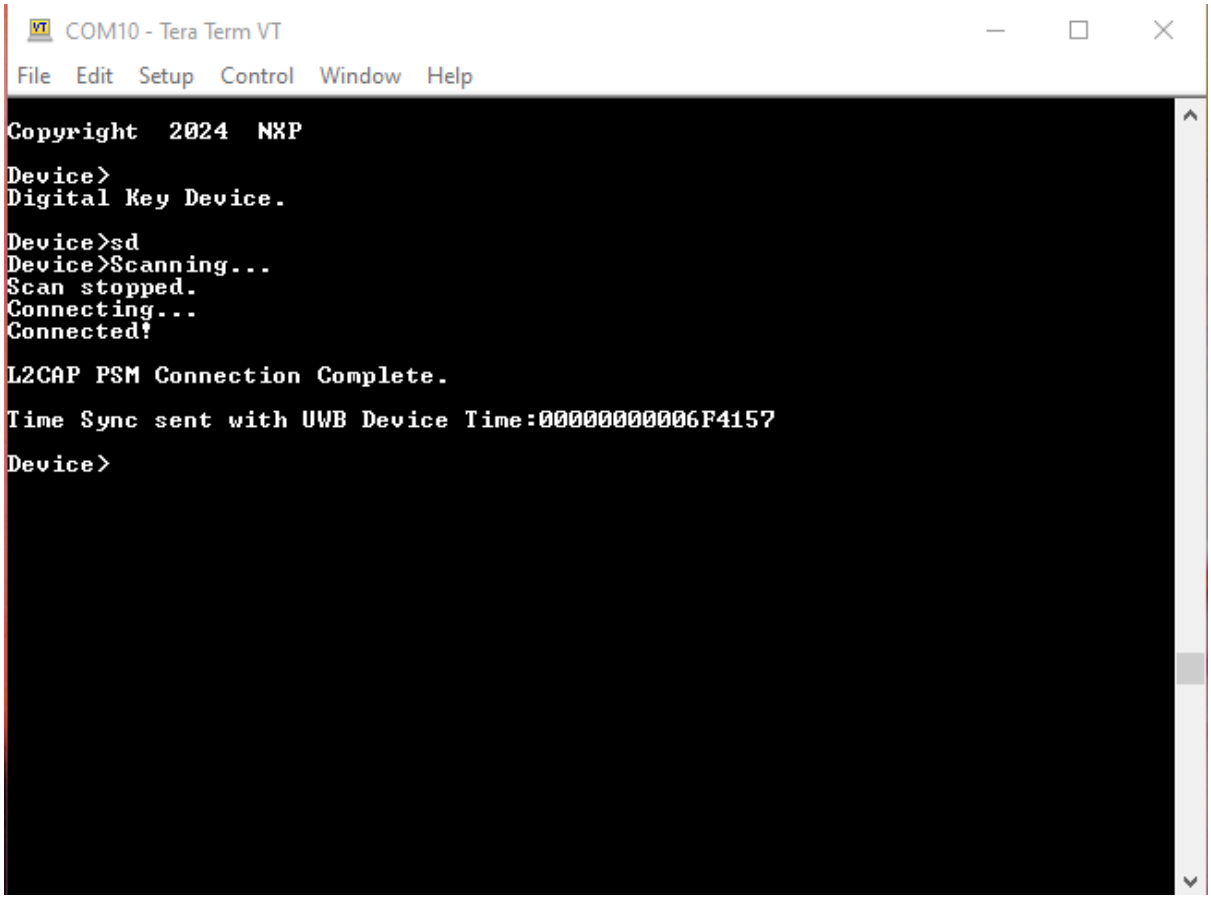
After reset, run the "sd pe" command on the Car Anchor and "sd" command on the Device as shown in Figure 1 and Figure 2. For the Passive Entry scenario, the Car Anchor advertises using two sets, one Legacy on the 1M PHY and one Extended on the Coded PHY. If bonding data is present, it follows the Passive Entry flow and stops after the successful creation of the L2CAP channel. At this point the application is functional. The output of the command is shown in Figure 1 and Figure 2. Similar to the Owner Pairing scenario, the applications implement only the Bluetooth Low Energy functionality.

Note: The CCC specification requires using coding scheme S=2 for Coded PHY advertising, equivalent to a data rate of 500 Kbits per sec. To experiment with other coding schemes, change the advertising parameters in the app_config.c file for the digital_key_car_anchor project.

Passive Entry on Car Anchor



Passive Entry on Device



Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

Synchronizing bonding data between Car Anchors **Note:** On the KW45B41Z-EVK and KW47-EVK platforms, this feature is only valid when Advanced Secure Mode is disabled (gAppSecureMode_d is set to 0 in the app_preinclude.h file). To synchronize bonding data when Advanced Secure Mode is enabled, refer to *Running the A2B scenario*.

Multiple Car Anchors can reside on a car, acting as a single Bluetooth Low Energy device as far as the Device is concerned. When the Device pairs and bonds with a Car Anchor, that bonding data must be shared with all other anchors. To showcase this functionality, the Car Anchor shell demo offers the "setbd" command.

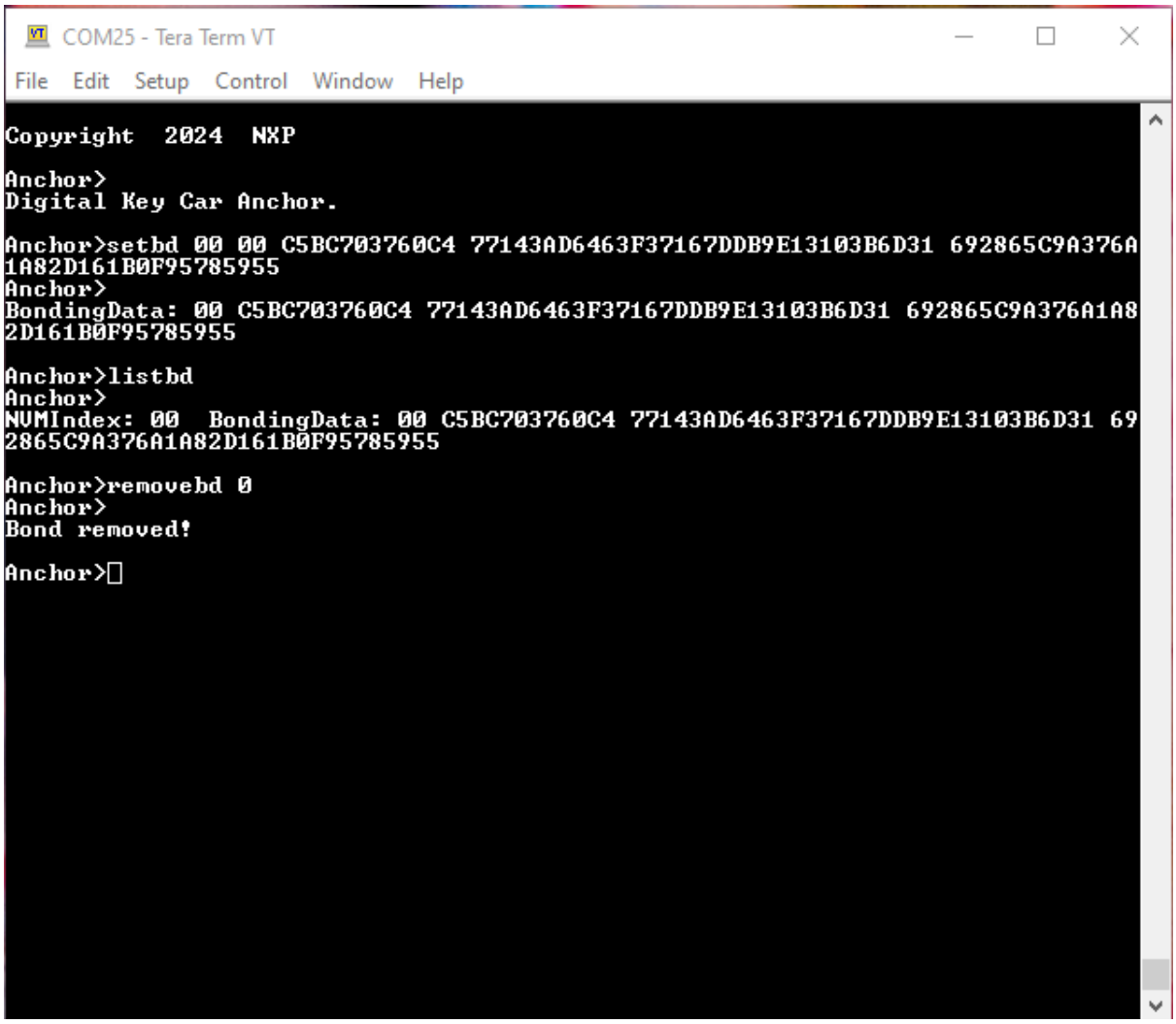
During owner pairing, the bonding data is displayed in the shell as seen in the Owner Pairing section. By passing this data to the "setbd" command on another Car Anchor, that anchor is able to connect with the original Device and perform the Passive Entry flow.

Note: Currently only the Bluetooth Low Energy bonding data is transferred between anchors. No CCC Digital Key R3-specific keys are exchanged.

Saved bonding data can also be viewed using the "listbd" command and a specific bond can be removed using the "removebd" command. See Figure 1 for details.

A bond can only be removed if a connection to that specific device is not currently active. The commands presented in this subsection are also supported on the Device for test purposes.

Adding bonding data to a Car Anchor, listing bonds and removing a bond



Note:

- To simulate anchors residing on the same car, Random Static Address required by the Digital Key protocol and the Identity Resolving Key are set at compile time for the digital_key_car_anchor project. To simulate anchors residing on different cars, change the values of the APP_BD_ADDR and APP_SMP_IRK macros in the project's app_preinclude.h.

Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

Connection with a non-CCC Key Fob Scenario The Car Anchor also supports connecting as a Bluetooth Low Energy central device (scanner) to a non-CCC key fob (advertiser). To enable scanning on the digital_key_car_anchor project set the gAppScanNonCCC_d macro to 1 in the project's app_preinclude.h before building.

After flashing the board and running "factoryreset", run the "sd op" or "sd pe" command exactly as in the Owner Pairing or Passive Entry scenarios. The Car Anchor will simultaneously advertise for a CCC Device and scan for a non-CCC key fob.

Connection with the non-CCC key fob and the subsequent application functionality is currently out of this application note's scope.

Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

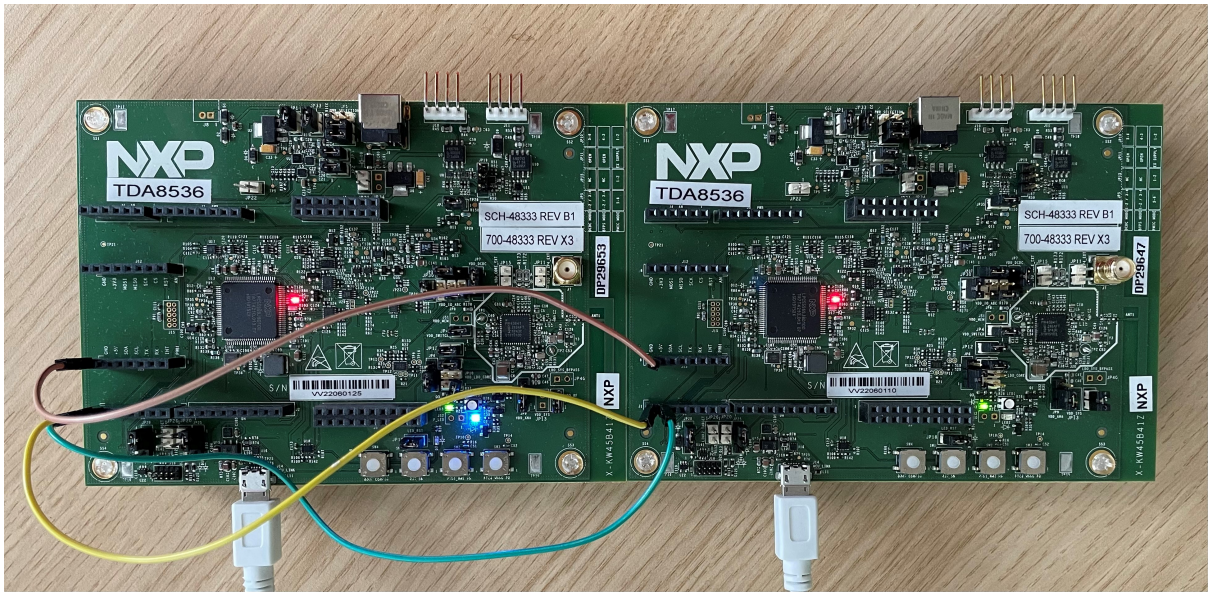
Running the Connection Handover scenario Connection Handover is an NXP proprietary feature that enables a Bluetooth Low Energy connection to be seamlessly transferred from one peripheral to another while the central device remains unaware. It is best exemplified by (but not limited to) the CCC Digital Key use case. As a person carrying a phone moves around the vehicle, the Bluetooth Low Energy connection is transferred between Car Anchors in order to ensure the best user experience. From the Device's point of view, it stays in the same initial connection.

The CCC Digital Key demos showcase the Connection Handover feature. The handover is performed on demand via a button press. In a real scenario, the handover is performed based on criteria such as RSSI. The feature is currently supported on the KW45B41Z-EVK and KW47-EVK platforms.

Prerequisites:

- Three boards (two boards act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.
- The two Car Anchors must have a serial connection via the secondary UART as shown in the image below. The UART connection stands in for a real deployment solution such as a CAN bus.
- The *gHandoverDemo_d* macro must be set to 1 in app_preinclude.h for the ***digital_key_car_anchor*** project. Figure 1 shows KW45B41Z-EVK car anchors connected via secondary UART. *(J1-1 is connected to J1-2 and J1-2 is connected to J1-1. GND connection is J13-1 to J13-1.)*
- If KW47-EVK boards are used, make the UART connection between J1-1 to J1-3 and J1-3 to J1-1.

Two Car Anchors connected via the secondary UART



Demo steps:

- Connect a Car Anchor and a Device using either the Owner Pairing or Passive Entry scenario as detailed in the previous sections. If doing Owner Pairing, the bonding data synchronization between the two Car Anchors will happen automatically over the UART connection.
- Optional: Use the "send" shell command on the Car Anchor to send a test message over the L2CAP Credit-Based channel. The message is displayed in the console on the Device.
- Press the **SW3** switch or use the handover command with the peer device ID on the Car Anchor which is connected to the Device. To see a list of connected devices with their corresponding peer device IDs the listad command may be used.
- The connection is handed over to the second Car Anchor as seen in console messages. LED2 also turns solid blue on the Car Anchor, which has taken over the connection.
- **Optional**: Use the "send" shell command on the Car Anchor that has taken over the connection. The message is displayed in the console on the Device.
- The connection can continue to be handed over back and forth between the two Car Anchors by pressing the **SW3** switch on the Car Anchor which currently has the connection. Figure 2 shows first Car Anchor connects and pairs with the Device, sends the L2CAP test message and successfully performs handover when **SW3** is pressed.

Paring and connection of First Car Anchor with Device and handover

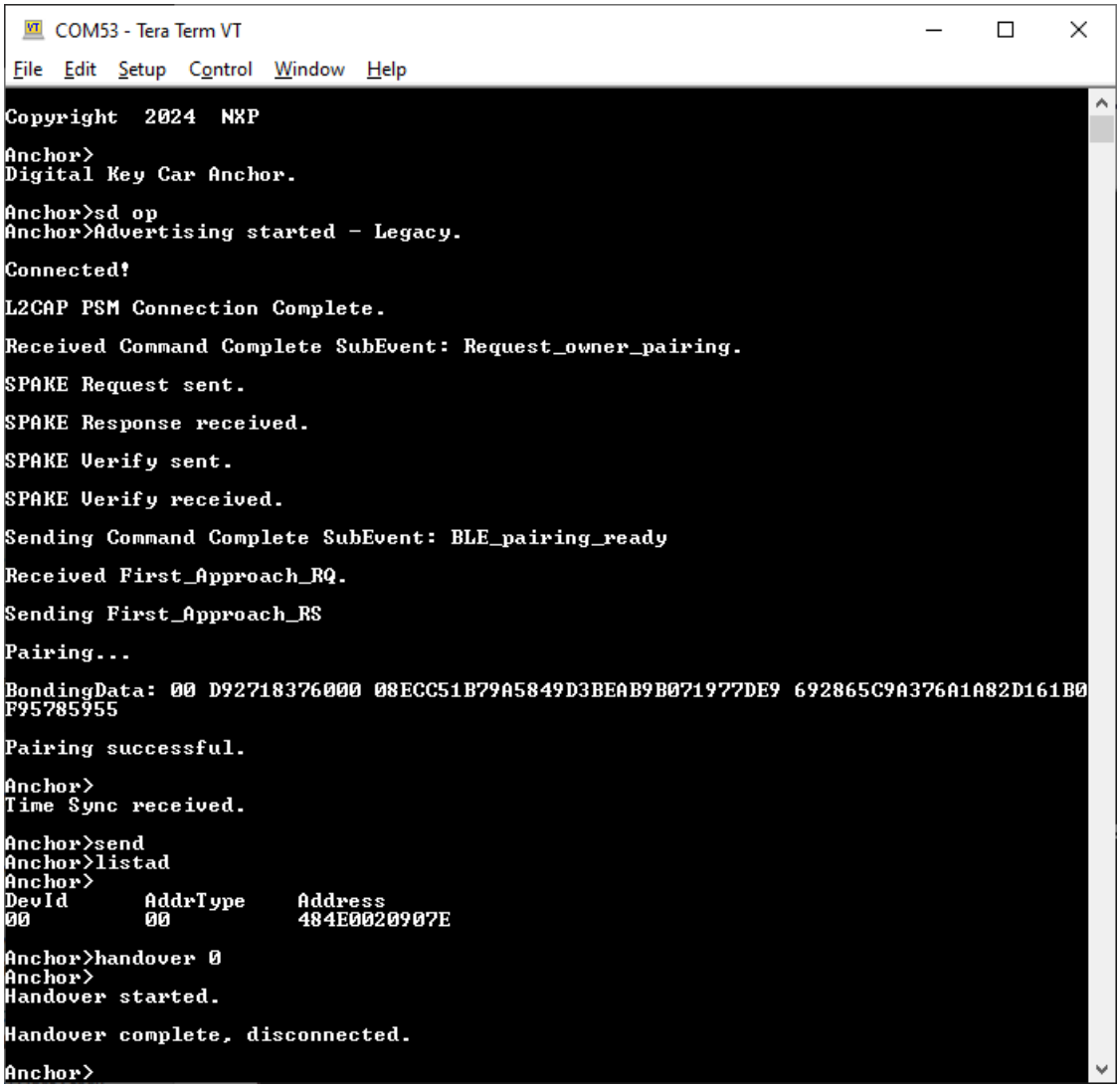


Figure 3 shows that the second Car Anchor receives the bonding data automatically via UART when the first Car Anchor pairs with the Device. After the handover is successfully completed, it sends the L2CAP test message to the Device.

Second Car Anchor receiving the bonding data automatically via UART when the first Car Anchor pairs with the Device

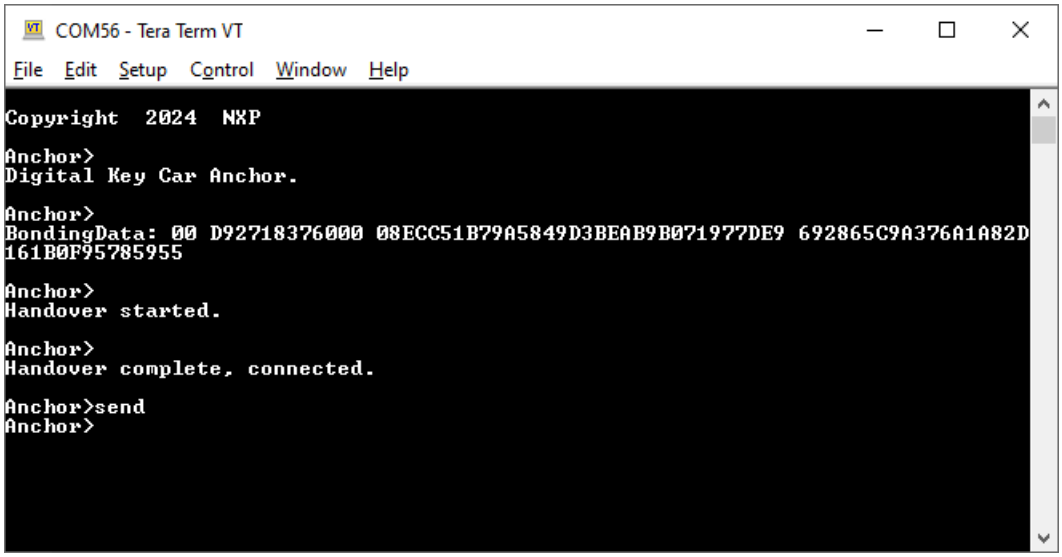
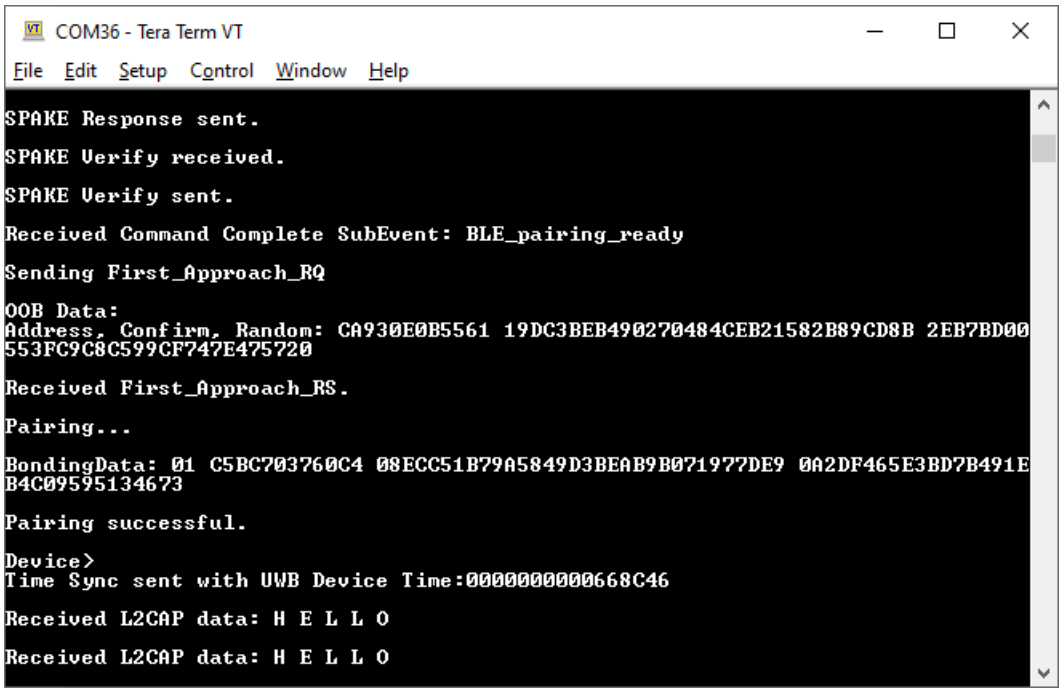


Figure 4 shows the Device receiving both L2CAP test messages. From the device's point of view, it is in a single uninterrupted connection.

The Device receives both L2CAP test messages

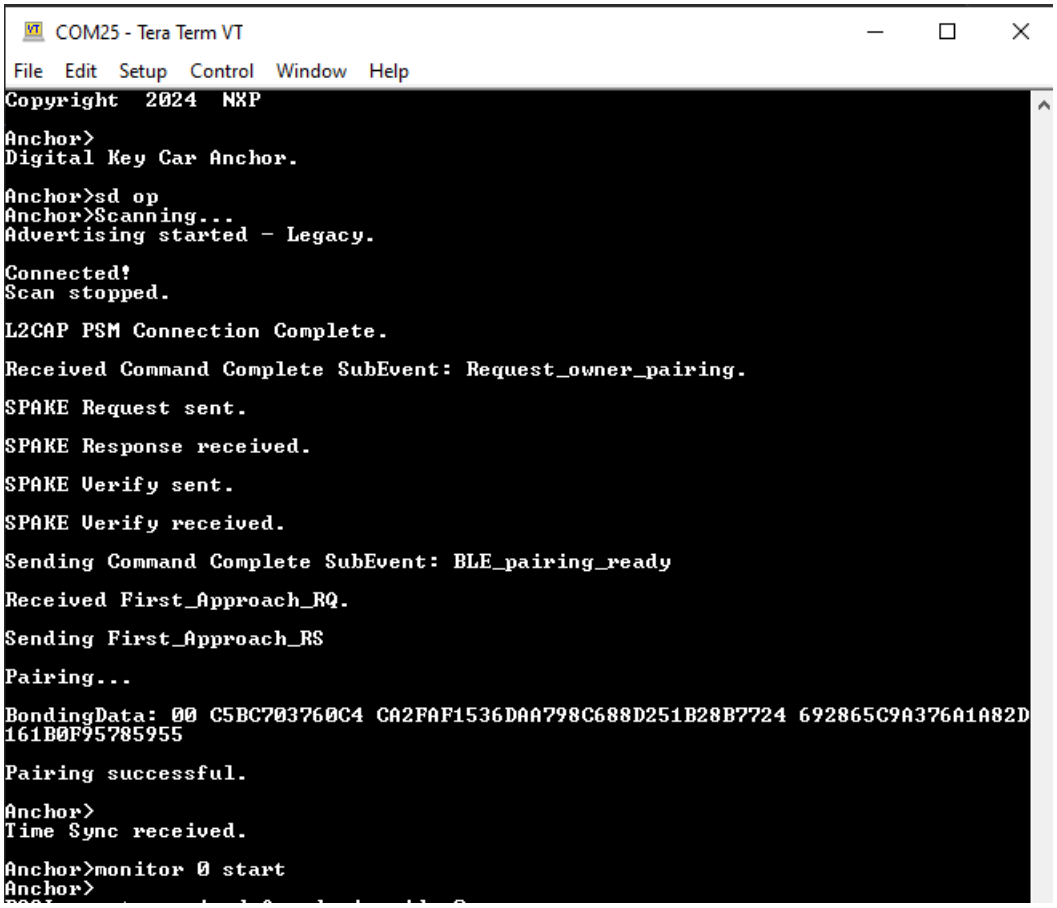


Running the Anchor Monitoring scenario Anchor Monitoring is an NXP proprietary feature that allows a second Anchor to monitor the connection between the first Anchor and the Device. The prerequisites are the same hardware setup as used for the Connection Handover scenario. Note, however, that the Connection Handover scenario cannot be performed while Anchor Monitoring is in progress.

Demo steps:

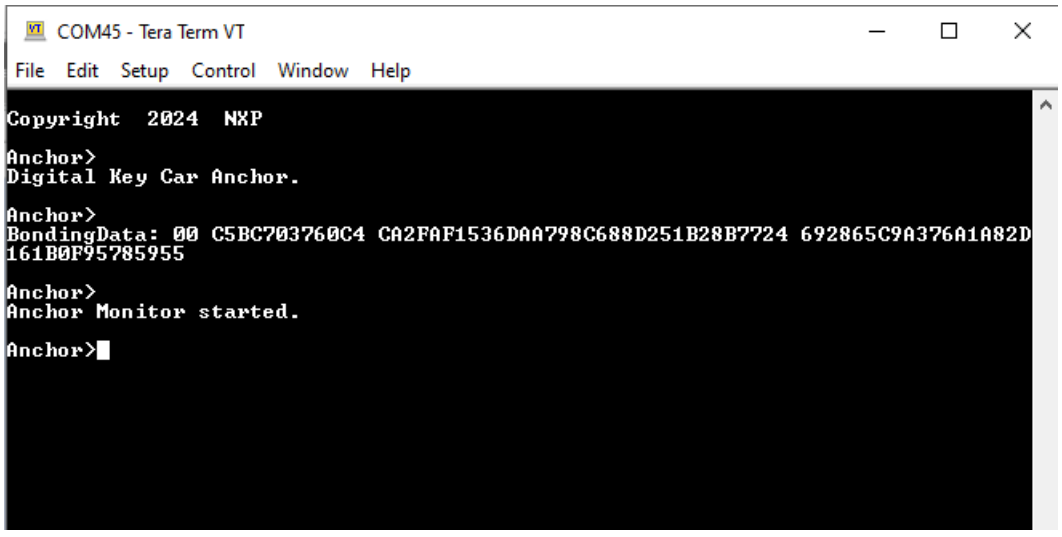
1. Connect a Car Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing Scenario and Passive Entry Scenario.

Anchor has performed Owner Pairing and the "monitor start" command is run



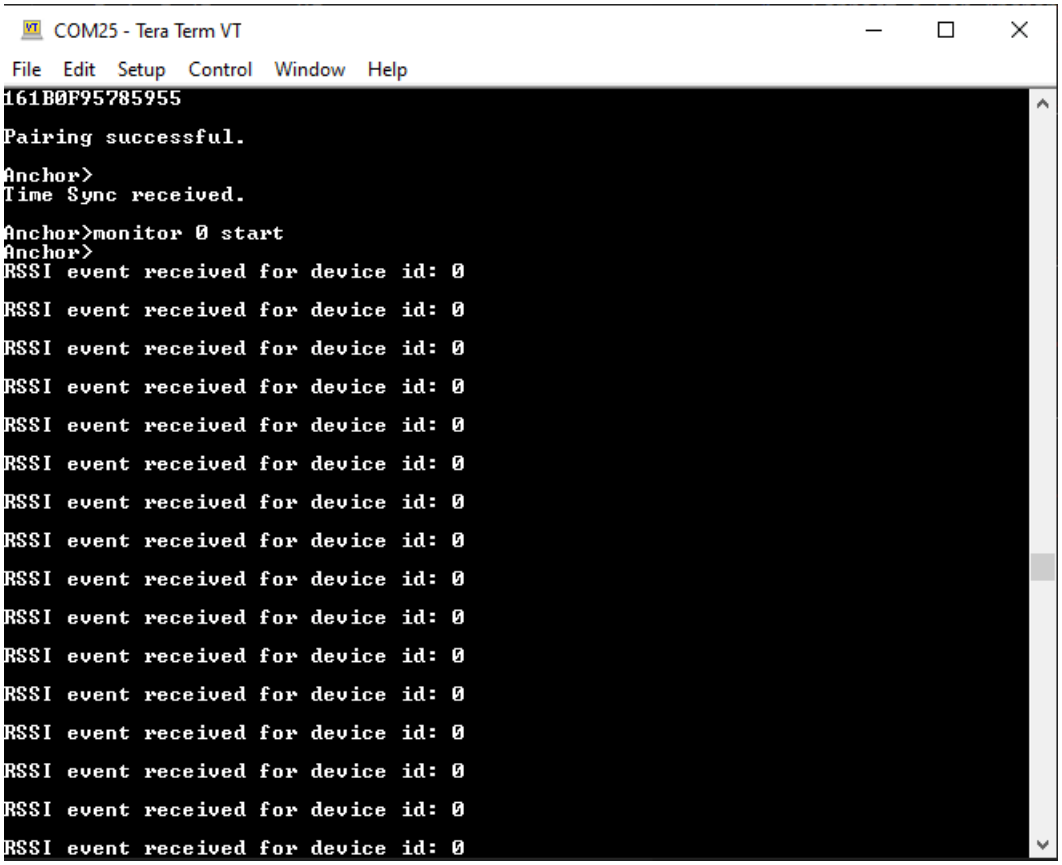
2. Run the "monitor 0 start" shell command on the Car Anchor, as seen in Figure 1.

The second Anchor has received the command over the serial interface and has started Anchor Monitoring



3. The second Car Anchor begins monitoring the connection and receives RSSI info events, which it forwards to the first Car Anchor (see Figure 2).

The first Anchor displays RSSI events received from the second Anchor.



4. The received RSSI messages are displayed in the console (see Figure 3).
5. To stop monitoring, run the "monitor 0 stop" shell command on either Car Anchor (the second one is recommended, as its console is not flooded by RSSI event messages).

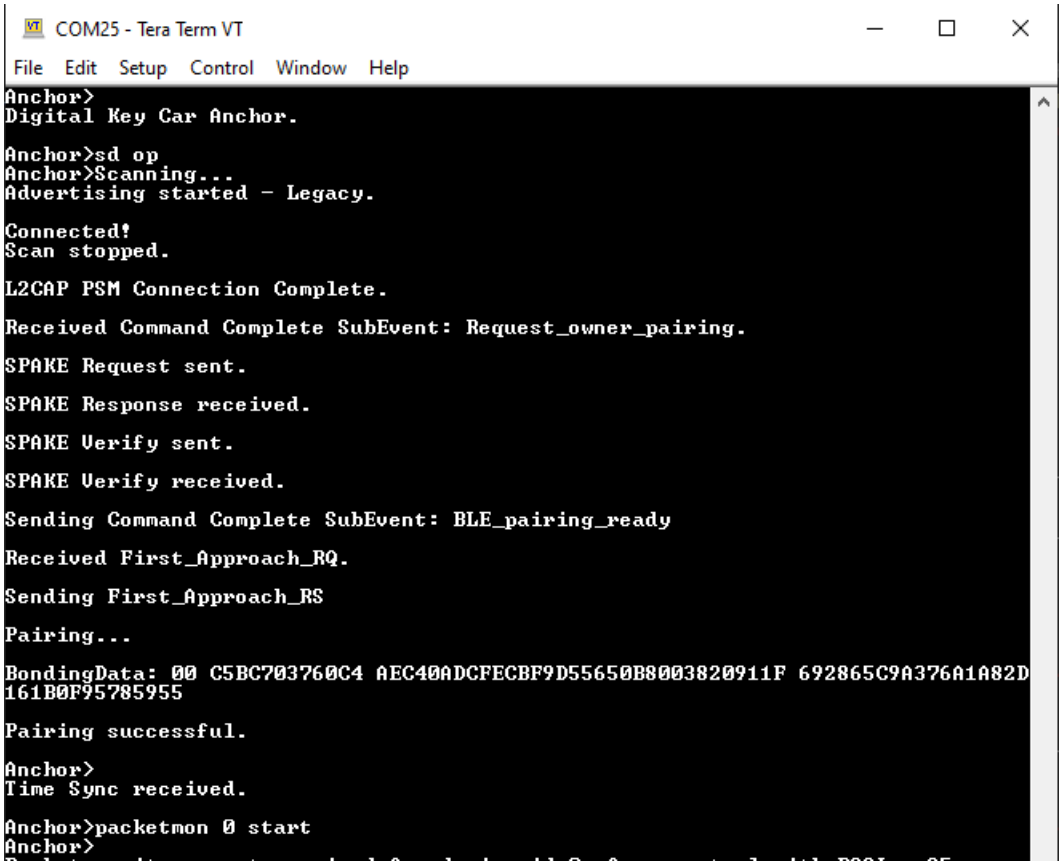
Parent topic:*Running the Connection Handover scenario*

Running the Packet Monitoring scenario Packet Monitoring is an NXP proprietary feature that allows a second Anchor to receive the packets from the connection between the first Anchor and the Device. The prerequisites are the same hardware setup as used for the *Running the Connection Handover scenario*. Note, however, that the Connection Handover scenario cannot be performed while Packet Monitoring is in progress. Packets received by the second Anchor are sent over the serial interface to the first Anchor for processing. The first Anchor determines the source of the packet based on the SN and NESN bits and displays the RSSI.

Demo steps:

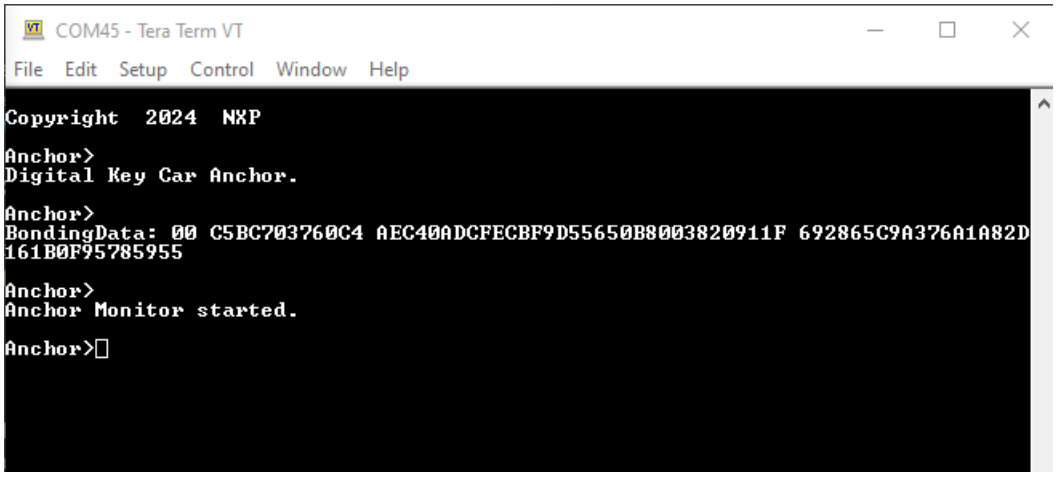
1. Connect the first Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing Scenario and Passive Entry Scenario.

Anchor has performed Owner Pairing and the "packetmon start" command is run



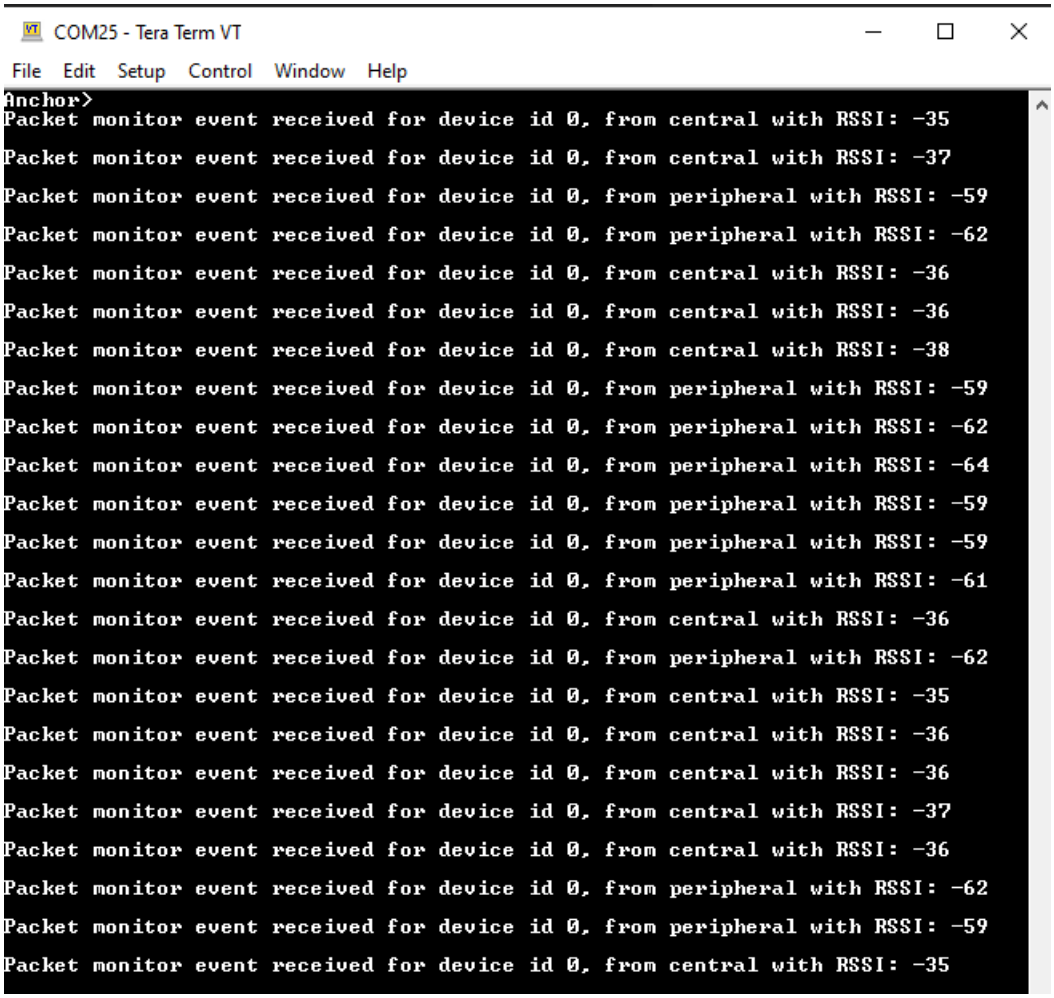
2. Run “packetmon 0 start” on the first Anchor, as shown in Figure 1.

The second Anchor has received the command over the serial interface and has started Packet Monitoring



3. The second Anchor begins monitoring the connection and receives packet information, which it forwards to the first Anchor. See Figure 2.

The first Anchor displays the RSSI and source device of the received packets from the second Anchor



4. The first Anchor receives the packet information from the second Anchor and determines the source of the packet based on the SN and NESN bits. The source and the RSSI are displayed in the console. See Figure 3.
5. To stop monitoring, run "packetmon 0 stop" shell command on either Anchor (the second one is recommended, as its console is not flooded by packet information messages).

Parent topic:*Running the Connection Handover scenario*

Running the A2B scenario The A2B feature allows for the secure transfer of Bluetooth Low Energy security keys information between the Car Anchor devices. The feature is currently supported on the following platforms:

- KW45B41Z-EVK with a lifecycle state of OEM-Open or higher, this feature is provided by the EdgeLock Secure Enclave.
- KW47-EVK with a lifecycle state of OEM-Open or higher, this feature is provided by th EdgeLock Secure Enclave.

The Car Anchor demo application makes use of the A2B feature to securely synchronize the local IRK and the Bonding Data (LTK and peer IRK) with another Car Anchor.

Prerequisites:

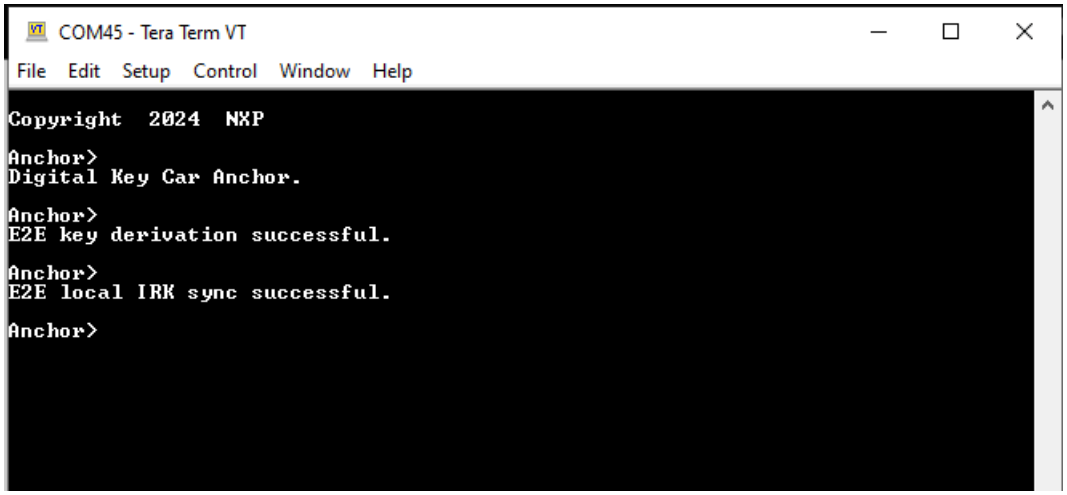
- Three boards (two act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.
- The two Car Anchors must be connected via a serial interface as described in the section above.

- Advanced Secure Mode must be enabled. The following macros must be defined and set to 1 in app_preinclude.h for the ***digital_key_car_anchor*** project:
  - gAppSecureMode_d
  - gA2BEnabled_d
- One of the Car Anchors must be configured with the gA2BInitiator_d macro set to 1, this is called Car Anchor A, and the other must be configured with the gA2BInitiator_d macro set to 0, called Car Anchor B. Car Anchor A triggers the EdgeLock-to-EdgeLock (E2E) key derivation and local IRK synchronization. Therefore, it should be started up after Car Anchor B.

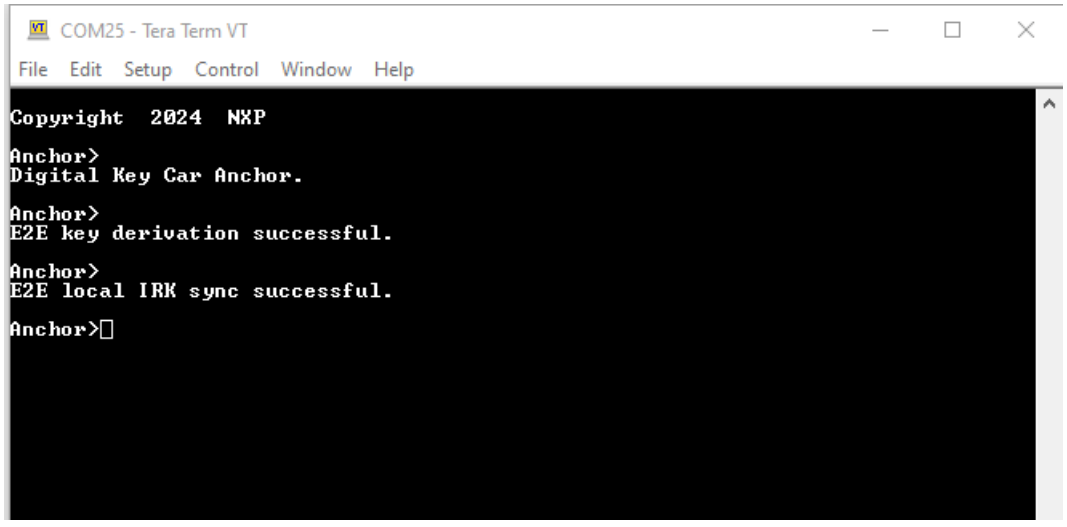
Demo steps:

- Start Car Anchor B, then Car Anchor A. At initialization the E2E key is derived and the local IRK of Car Anchor A is sent as a secure blob to Car Anchor B in order for both Car Anchors to have the same local IRK as shown in the Figure 1 and Figure 2.

Car Anchor A starts up, derives the E2E key and syncs with Car Anchor B

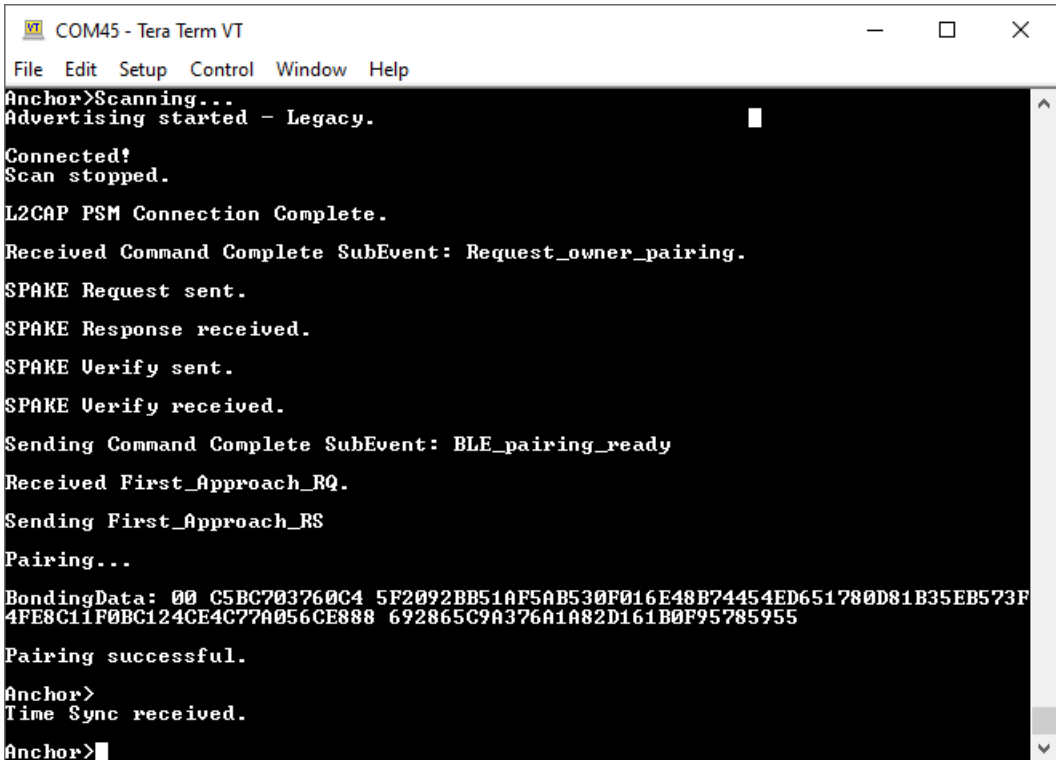


Car Anchor B starts up and syncs with Car Anchor B

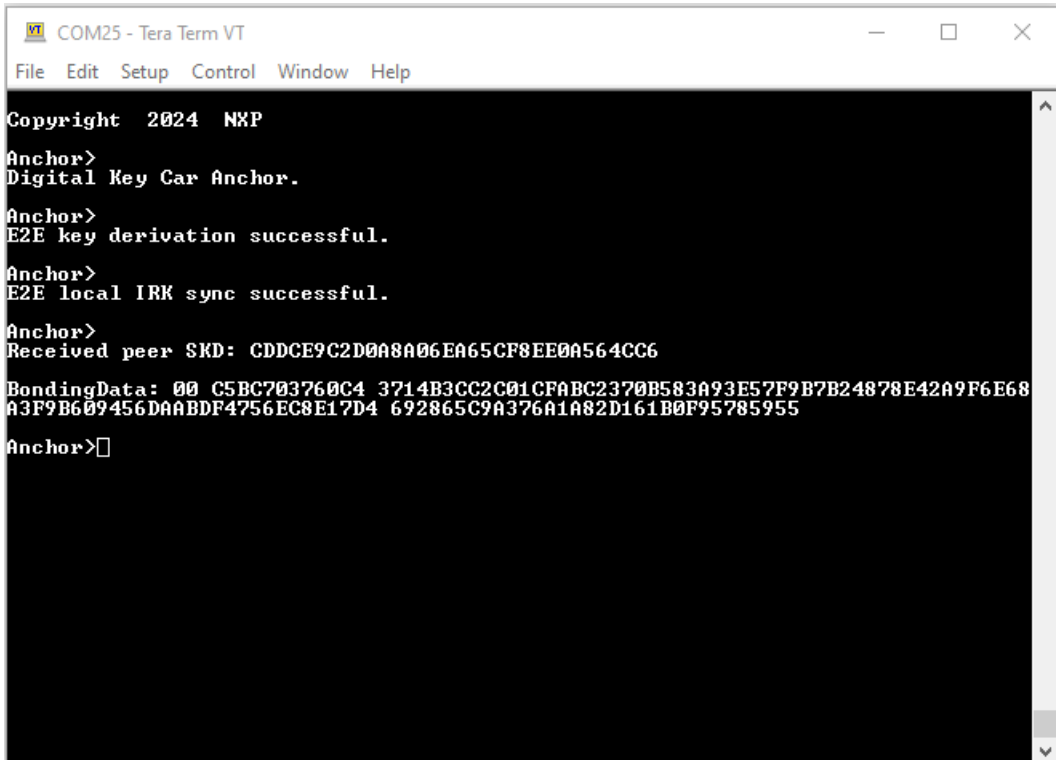


- Trigger the Owner Pairing scenario by sending the "sd" command on the Device and "sd op" on Car Anchor A. Once the bond is created, the Bonding Data containing secured blobs for the LTK and peer IRK is sent to Car Anchor B. The Car Anchors display the Bonding Data in shell with the LTK in blob form (the LTK is never available in plain text) and the peer IRK in plain text. Now both Car Anchors have the same Bonding Data. Refer Figure 3 and Figure 4.

Car Anchor A pairs with Device

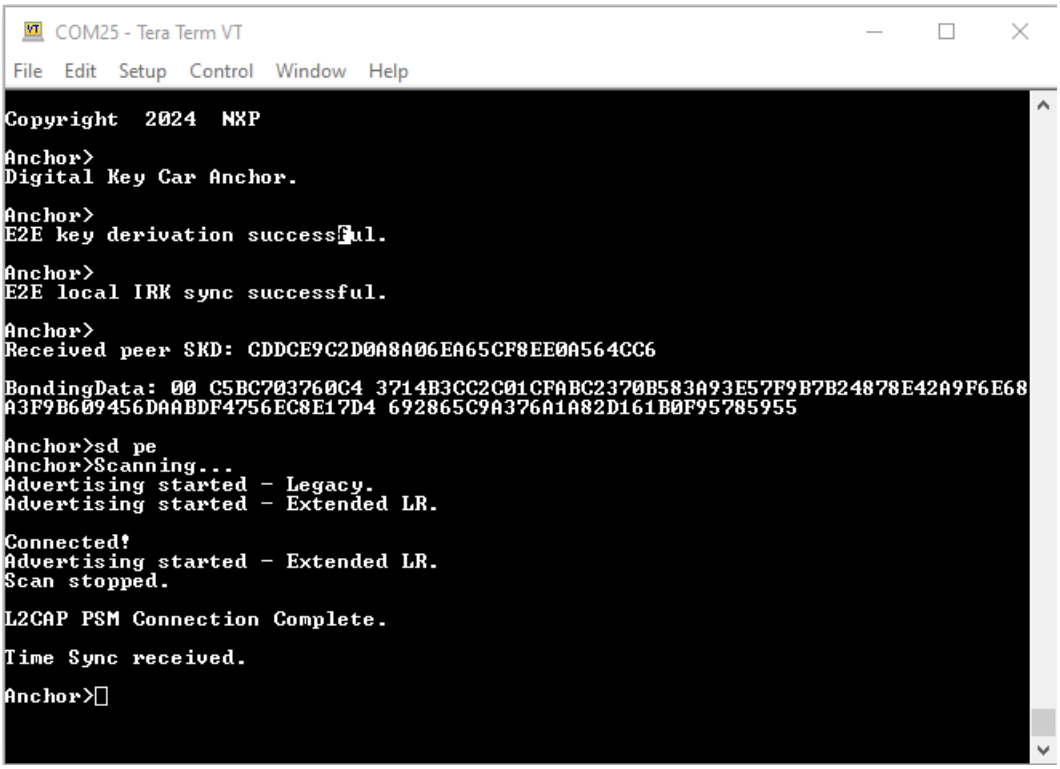


Car Anchor B has received the Bonding Data via the serial interface



- Disconnect Car Anchor A by sending the "dcnt" command.
- Once Car Anchor A is disconnected, send "sd" on the Device and "sd pe" on Car Anchor B. Car Anchor B is now able to perform a Passive Entry connection, without pairing as shown in Figure 5.

Car Anchor B performs Passive Entry



Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF) Decision Based Advertising Filtering can be used in the Passive Entry scenario. Using DBAF optimizes the scanning operation performed by the Digital Key Device of the advertisements sent by the Digital Key Car Anchor devices. This is accomplished by using 'ADV_DECISION_IND' PDUs, which contain the advertising address in the extended header. Using these PDUs allows the scanner to perform address resolution without the need to retrieve the AUX_ADV_IND PDU sent on the data channels.

The DBAF feature is supported on the following platforms:

- KW45B41Z-EVK
- KW47-EVK

Enabling DBAF To enable DBAF, modify the Digital Key applications as per the steps listed below:

- In app_preinclude.h file, set gBLE60_DecisionBasedAdvertisingFilteringSupport_d to TRUE.

Parent topic:*Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF)*

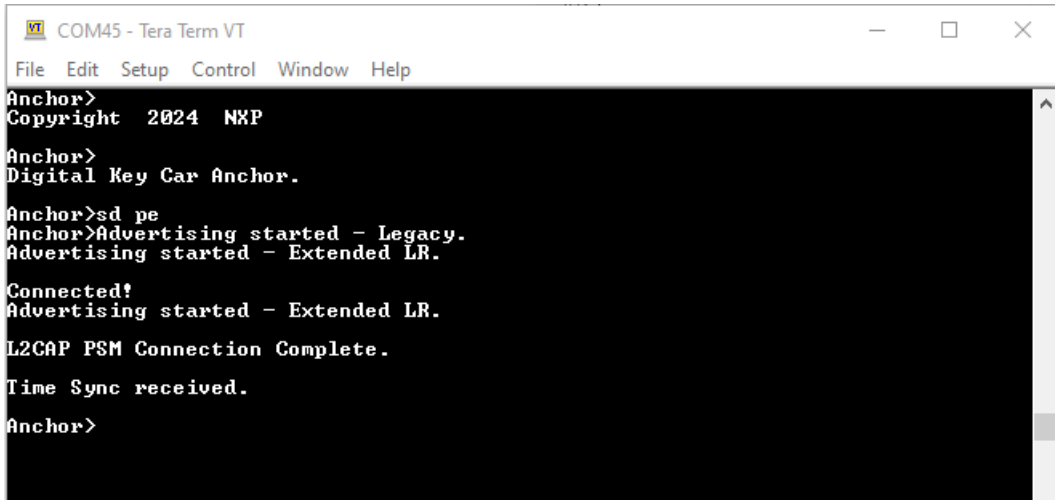
Running the Passive Entry with DBAF scenario The Passive Entry with DBAF scenario runs after Owner Pairing between a Car Anchor and a Device successfully completes.

1. To run the Passive Entry with DBAF demo, reset the recently paired boards using either the **RST** switch or the "reset" command in the shell. Do NOT perform a factory reset, as this removes the bonding information from non-volatile memory.
2. After reset, run the "sd pe" command on the Car Anchor and "sd pe" command on the Device as shown in Figure 1 and Figure 2.

Passive Entry with DBAF on Device



Passive Entry with DBAF on Car Anchor



After following these steps, the Passive Entry scenario as described in Passive Entry Scenario runs.

Parent topic:*Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF)*

Intrusion Detection System The Intrusion Detection System (IDS) is a proprietary feature which allows the Car Anchor to monitor anomalous events related to the traffic and conditions generated during the operation of the Bluetooth LE software stack. These events are detected at Host and Controller level. The application registers a callback where it receives and handles reports.

The IDS feature is supported on the following platforms:

- KW45B41Z-EVK
- KW47-EVK

To enable IDS, modify the Digital Key Car Anchor as per the steps listed below:

- In app_preinclude.h file, set gIntrusionDetectionSystem_d to TRUE.

References For more information, refer to the following documents:

- Bluetooth Low Energy Application Developer's Guide (KW45_K32W1_BLEADG)
- Bluetooth Low Energy Host Stack API Reference Manual (KW45_K32W1_BLEHSAPIRM)

- Bluetooth Low Energy Host Stack FSCI (Framework Serial Connectivity Interface) API Reference Manual

  (KW45_K32W1_BLEHSFSCIRM)
- Connectivity Framework Reference Manual (KW45_K32W1_CONNFWRM)
- Bluetooth Low Energy CCC Digital Key with Channel Sounding Application Note (AN13979)

Acronyms and abbreviations *Table 1* lists the acronyms used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
CCC	Car Connectivity Consortium
DBAF	Decision Based Advertising Filtering
IDE	Integrated Design Environment
IDS	Intrusion Detection System
E2E	EdgeLock-to-EdgeLock
IRK	Identity Resolving Key
LTK	Long Term Key
OOB	Out-of-Band
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
UWB	Ultra-Wideband

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2020-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy Host Stack FSCI Application Programming **Kinetis FSCI Host Application Programming Interface**

About This Document This document provides a detailed description for the Host Application Programming Interface (Host API) implementing the Framework Serial Connectivity Interface (FSCI) on a peripheral port such as UART, USB, and SPI. The Host API can be deployed from a PC tool or a host processor to perform control and monitoring of a wireless protocol stack running on the microcontroller. The software modules and libraries implementing the Host API is the Host Software Development Kit (SDK).

This version of the document describes the Bluetooth® Low Energy stack running on the Wireless Connectivity Microcontrollers (MCUs), which are interfaced from a high-level OS (Linux® OS, Windows® OS) by the Host API and the Host SDK.

Audience

This document is for software developers who create tools and multichip partitioned systems using a serial interface to a Bluetooth LE *black box* firmware running on a microcontroller.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Deploying Host Controlled Firmware IAR Embedded Workbench for Arm® (EWARM) and MCUXpresso IDE are the development toolchains used to deploy the Bluetooth LE host stack software applications.

Detailed information about how to build, deploy and debug an IAR or MCUXpresso IDE-based project are presented in the *Bluetooth Low Energy Demo Applications User Guide*.

Bluetooth LE application configuration

To exercise the Host API, the Bluetooth LE *black box* firmware is required to be flashed on a compatible platform. The Bluetooth LE *black box* is represented by the ble_fscibb application firmware that can be interfaced and configured with FSCI commands over the serial interface.

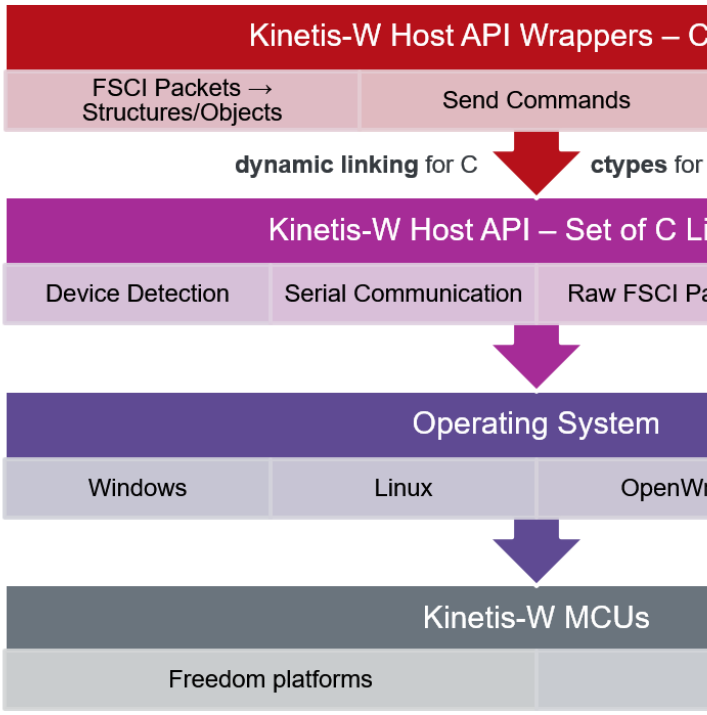
The user can compile the *black box* image of the ble_fscibb software application using IAR Embedded Workbench for Arm (EWARM) or MCUXpresso IDE. For information on how to build the application see additional documentation provided in the package.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Host Software Overview The FSCI (Framework Serial Communication Interface) module allows interfacing the protocol stack with a host system or PC tool using a serial communication interface.

FSCI can be demonstrated using various host software, one being the set of Linux OS libraries exposing the Host API described in this document. The NXP Test Tool for Connectivity Products PC application is another interfacing tool, running on the Windows OS. Bluetooth LE stack makes use of XML files which contain detailed meta-descriptors for commands and events transported over the FSCI.

The FSCI module is explained in the Wireless Framework - Services - FSCI documentation.



Wireless host software system block diagram

Directory tree

```
host_sdk
   hsdk
      demo                          A set of programs to demonstrate functionality.
         FSCIBootloader.c
         GetKinetisDevices.c           Outputs all Kinetis devices available on serial to the console.
         Makefile
         bin                        Folder containing libraries ".so".
      doc                           Folder containing documentation.
      include                       All the headers used are present in this folder.
         physical                   Headers specific to the physical serial bus or PCAP interface used by␣
↪the NXP device.
            PhysicalDevice.h
            PCAP
               PCAPDevice.h
            SPI
               SPIConfiguration.h      Handles SPI slave bus configuration(max speed Hz, bits per word).
               SPIDevice.h             Encapsulates an OS SPI device node into a well-defined structure.
            UART
               UARTConfiguration.h       Handles serial port configuration (baudrate, parity).
               UARTDevice.h              Encapsulates an OS UART device node into a well-defined structure.
               UARTDiscovery.h           Handles the discovery of UART connected devices.
         protocol                   Headers specific to the transmission of frames.
            Framer.h                 A state machine implementation for sending/ receiving frames.
            FSCI                     Headers specific to the FSCI protocol.
               FSCIFrame.h
               FSCIFramer.h
         sys                        General purpose headers for interaction with the OS, message queues and␣
↪more.
            EventManager.h              Handles event registering, notifying and callback submission.
            hsdkError.h              Macros for error reporting.
            hsdkLogger.h             Logger implementation for debugging.
            hsdkOSCommon.h              Interaction with OS specifics.
```

(continues on next page)

(continued from previous page)

```
            MessageQueue.h              A standard message queue implementation (linked list).
            RawFrame.h                  Describes the format of a frame, independent of the protocol.
            utils.h                     Various functions to manipulate structures and byte arrays.
        physical                        Implementation of the physical UART/SPI serial bus or PCAP interface
↪module.
            PCAP
                PCAPDevice.c
            PhysicalDevice.c
            SPI
                SPIConfiguration.c
                SPIDevice.c
            UART
                UARTConfiguration.c
                UARTDevice.c
                UARTDiscovery.c
        protocol                        Implementation of the protocol module relating to FSCI.
            Framer.c
            FSCI
                FSCIFrame.c
                FSCIFramer.c
        res
            77-mm-usb-device-blacklist.rules    Udev rules for disabling ModemManager.
            hsdk.conf                   Configuration file to control FSCI-ACKs.
        sys                             Implementation of the system/OS portable base module.
            EventManager.c
            hsdkEvent.c
            hsdkFile.c
            hsdkLock.c
            hsdkLogger.c
            hsdkSemaphore.c
            hsdkThread.c
            MessageQueue.c
            RawFrame.c
            utils.c
        ConnectivityLibrary.sln         Microsoft Visual Studio solution file.
        HSDK.rc
        Makefile
        README.md
        resource.h
    hsdk-c                              Described in Host API C Bindings
    hsdk-python                         Described in Host API Python Bindings
    README.txt
```

Device detection The Wireless Host SDK can detect every USB attached peripheral device to a PC. On Linux OS, this is done via udev. Udev is the device manager for the Linux OS kernel and was introduced in Linux OS 2.5. Using the manager, the Wireless Host API can provide the Linux OS path for a device (for example, /dev/ttyACM0) and whether the device is a supported USB device, based on the vendor ID/product ID advertised. Upon device insertion, the USB cdc_acm kernel module is triggered by the kernel for interaction with TWR, USB and FRDM devices.

On Windows OS, attached peripherals are retrieved from the registry path HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\SERIALCOMM, resulting in names such as *COMxx* which must be used as input strings for the Python scripts which require a device name.

Serial port configuration The Host SDK configures a serial UART port with the following default values:

Host SDK – UART default values

Configuration	Value
Baudrate	115200
ByteSize	EIGHTBITS
StopBits	ONE_STOPBIT
PARITY	NO_PARITY
HandleDSRControl	0
HandleDTRControl	ENABLEDTR
HandleRTSControl	ENABLERTS
InX	0
OutCtsFlow	1
OutDSRFlow	1
OutX	0

The library only allows the possibility to change the baudrate, as this is the most common scenario.

> **NOTE**: For devices using a USB connection interfaced directly (where the USB stack runs on the device and the system is NOT using an OpenSDA UART to USB converter), the baudrate is not necessary and setting it has no effect.

The Host SDK configures a serial SPI port with the following default values:

Host SDK – SPI default values

Configuration	Value
Transfer Mode	Mode SPI_MODE_0
Maximum SPI transfer speed (Hz)	1 MHz
Bits per word	8

The library only allows the possibility to change the maximum SPI transfer speed.

Logger The Host SDK implements a logger functionality which is useful for debugging. Adding the compiler flag USE_LOGGER enables this functionality.

When running programs that make use of the Host SDK, a file named hsdk.log appears in the working directory. This is an excerpt from the log:

```
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Allocated memory for PhysicalDevice
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Initialized device's message queue
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created event manager for device
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created threadStart event
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created stopThread event
HSDK_INFO - [6684] [PhysicalDevice]AttachToConcreteImplementation:Attached to a concrete␣
↪implementation
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created and start device thread
HSDK_INFO - [6684] [Framer]InitializeFramer:Allocated memory for Framer
HSDK_INFO - [6684] [Framer]InitializeFramer:Created stopThread event
HSDK_INFO - [6684] [Framer]InitializeFramer:Initialized framer's message queue
HSDK_INFO - [6684] [Framer]InitializeFramer:Created event manager for framer
```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Linux OS Host Software Installation Guide

Libraries Prerequisites

Packages: build-essential, libudev, libudev-dev, libpcap, libpcap-dev. Use apt-get install on Debian-based distributions. The Linux OS kernel version must be greater than 3.2.

Installation

```
$ pwd
/home/user/hsdk
$ make
$ find build/ -name "\*.so"
build/libframer.so
build/libsys.so
build/libphysical.so
build/libuart.so
build/libfsci.so
build/libspi.so
$ sudo make install
```

By default, make generates sharedlibraries (having .so extension). The step make install (superuser privileges required) copies these libraries to /usr/local/lib, which is part of the default Linux OS library path. The installation prefix may be changed by passing the variable PREFIX, e.g. make install PREFIX=/usr/lib. The user is responsible for making sure that PREFIX is part of the system's LD_LIBRARY_PATH.

Static libraries can be generated instead, by modifying the LIB_OPTION variable in the Makefile from dynamic to static (.a extension).

make install also disables the ModemManager control for the connected devices. Otherwise, the daemon starts sending AT commands that affect the responsiveness of the afore-mentioned devices in the first 20 seconds after plug in.

Demos Installation

```
$ pwd
/home/user/hsdk/demo
$ make
$ ls bin/
GetKinetisDevices
```

These demos are provided in this package:

- GetKinetisDevices: this program detects the boards connected to the serial line and outputs the path to the console:

  ```
  $ ./GetKinetisDevices
  NXP Kinetis-W device on /dev/ttyACM0.
  NXP Kinetis-W device on /dev/ttyACM1.
  ```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Windows OS Host Software Installation Guide

Libraries Prerequisites

Microsoft Visual Studio® is required to build the host software. Open the solution file ConnectivityLibrary.sln and build it for either Win32 or x64, depending on your setup requirements. The output directory contains a file named HSDK.dll, which can be thought of as a bundle of all the shared libraries from Linux OS, except for SPI (libspi.so). Currently, SPI interface to the board is not supported by the Windows host software.

Prebuilt HSDK.dll files are available under directory hsdk-python\lib.

Installation

The host software for the Windows OS is designed to work in a Python environment by contrast to the Linux OS where standalone C demos also exist.

Download and install the Python 3.10.x/32b package from Python Downloads. When customizing the installation options, check Add python.exe to Path.

Using prebuilt library

1. Depending on your Python environment architure (not Windows architecture) copy the appropriate HSDK.dll from hsdk-python\lib\[x86|x64]to \<Python Directory\>\DLLs.
2. Download and install Visual C++ Redistributable Packages for Microsoft Visual Studio, depending on the Windows architecture of your system (vcredist_x86.exe or vcredist_x64.exe) from Microsoft Visual C++ Redistributable latest supported downloads.
3. Download and install the Microsoft Visual C++ Compiler for Python as described in the python Windows Complilers.

Using local built library

1. Depending on your Python environment architure (not Windows architecture), build the appropriate Microsoft Visual Studio solution configuration and then copy HSDK.dll to \<Python Directory\>\DLLs.
2. Download and install the Microsoft Visual C++ Compiler for Python as described in the python Windows Complilers.

Optionally, copy the hsdk\res\hsdk.conf to \<Python Directory\>\DLLs to control the behavior of the FSCI-ACK synchronization mechanism.

Demos See *Host API Python Bindings*.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Host API C Bindings The Host SDK includes a set of C bindings to interface with a *black-box*. Bindings are generated from the matching FSCI XML file that is available in the stack software package under tools\wireless\xml_fsci. The bindings are designed to be platform agnostic, with a minimal set of OS abstraction symbols required for building. Thus, the files can be easily integrated on a wide range of host platforms.

Directory Tree

```
hsdk-c/
  demo
    HeartRateSensor.c       Source file that implements the Bluetooth LE Heart Sensor Profile.
    Makefile
  inc
    ble_sig_defines.h       Standard Bluetooth SIG UUID values.
    cmd_<name>.h            Generated from the matching FSCI <name>.xml file.
    os_abstraction.h        Provides OS dependent symbols for building the interface.
  README.md
  src
    cmd_<name>.c            Generated from the matching FSCI <name>.xml file.
    evt_<name>.c            Generated from the matching FSCI <name>.xml file.
    evt_printer_<name>.c    Generated from the matching FSCI <name>.xml file.
```

Tests and examples Tests and examples that make use of the C bindings are placed in the hsdk-c/demo directory. Example of usage:

```
$ cd hsdk-c/demo/
$ make
$ ./HeartRateSensor /dev/ttyACM0
[...]
--> Setup finished, please open IoT Toolbox -> Heart Rate -> HSDK_HRS
```

Development Header file cmd_<name>.h is generated from the corresponding <name>.xml FSCI XML file.

- Enumerations

```
/* Indicates whether the connection request is issued for a specific device or for all the devices in the␣
↪White List - default specific device */
typedef enum GAPConnectRequest_FilterPolicy_tag {
    GAPConnectRequest_FilterPolicy_gUseDeviceAddress_c = 0x00,
    GAPConnectRequest_FilterPolicy_gUseWhiteList_c     = 0x01
} GAPConnectRequest_FilterPolicy_t;
```

- Structures

```
typedef PACKED_STRUCT GAPConnectRequest_tag {
    uint16_t ScanInterval;          // Scanning interval - default 10ms
    uint16_t ScanWindow;            // Scanning window - default 10ms
    GAPConnectRequest_FilterPolicy_t FilterPolicy;          // Indicates whether the connection␣
↪request is issued for a specific device or for all the devices in the White List - default specific device
    GAPConnectRequest_OwnAddressType_t OwnAddressType;      // Indicates whether the address␣
↪used in connection requests will be the public address or the random address - default public address
    GAPConnectRequest_PeerAddressType_t PeerAddressType;    // When connecting to a specific␣
↪device, this indicates that device's address type - default public address
    uint8_t PeerAddress[6];       // When connecting to a specific device, this indicates that device's␣
↪address
    uint16_t ConnIntervalMin;       // The minimum desired connection interval - default 100ms
    uint16_t ConnIntervalMax;       // The maximum desired connection interval - default 200ms
    uint16_t ConnLatency;           // The desired connection latency (the maximum number of␣
↪consecutive connection events the Slave is allowed to ignore) - default 0
    uint16_t SupervisionTimeout;    // The maximum time interval between consecutive over-the-air␣
↪packets; if this timer expires, the connection is dropped - default 10s
    uint16_t ConnEventLengthMin;    // The minimum desired connection event length - default 0ms
    uint16_t ConnEventLengthMax;    // The maximum desired connection event length - default␣
↪maximum possible, ~41 s
    bool_t usePeerIdentityAddress;  // TRUE if the address defined in the peerAddressType and␣
↪peerAddress is an identity address
} GAPConnectRequest_t;
```

- Container for all possible event types

```
typedef struct bleEvtContainer_tag {
    uint16_t id;
    union {
        [...]
        GAPConnectionEventConnectedIndication_t GAPConnectionEventConnectedIndication;
        [...]
    } Data;
} bleEvtContainer_t;
```

- Prototypes for sending commands

```
memStatus_t GAPConnectRequest(GAPConnectRequest_t *req, void *arg, uint8_t fsciInterface);
```

Header file os_abstraction.h provides the required symbols for building the generated interface. When integrating in a project different that Host SDK, the user needs to provide the implementation for

```
void FSCI_transmitPayload (
    void *arg,              /* Optional argument passed to thefunction */
    uint8_t og,             /* FSCI operation group */
    uint8_t oc,             /* FSCI operation code */
    void *msg,              /* Pointer to payload */
    uint16_t msgLen,        /* Payload length */
    uint8_t fsciInterface   /* FSCI interface ID */
);
```

that creates and sends a FSCI packet (0x02 | og | oc | msgLen | msg | crc +- fsciInterface) on the serial interface. Source files cmd_<name>.c, evt_<name>.c and evt_printer_<name>.c are generated from the correspondent <NAME>.xml FSCI XML file.

- Functions that handle command serialization in cmd_<name>.c

```
memStatus_t GAPConnectRequest(GAPConnectRequest_t *req, void *arg, uint8_t fsciInterface) {
    /* Sanity check */
    if (!req)
    {
        return MEM_UNKNOWN_ERROR_c;
    }
    FSCI_transmitPayload(arg, 0x48, 0x1C, req, sizeof(GAPConnectRequest_t), fsciInterface);
    return MEM_SUCCESS_c;
}
```

- Event dispatcher in evt_<name>.c

```
void KHC_BLE_RX_MsgHandler(void *pData, void *param, uint8_t fsciInterface) {
    if (!pData || !param)
    {
        return;
    }

    fsciPacket_t *frame = (fsciPacket_t *)pData;
    bleEvtContainer_t *container = (bleEvtContainer_t *)param;
    uint8_t og = frame->opGroup;
    uint8_t oc = frame->opCode;
    uint8_t *pPayload = frame->data;
    uint16_t id = (og << 8) + oc, i;

    for (i = 0; i < sizeof(evtHandlerTbl) / sizeof(evtHandlerTbl[0]); i++)
    {
        if (evtHandlerTbl[i].id == id)
        {
            evtHandlerTbl[i].handlerFunc(container, pPayload);
            break;
        }
    }
}
```

- Handler functions to perform event de-serialization in evt_<name>.c

```
static memStatus_t Load_GAPConnectionEventConnectedIndication(bleEvtContainer_t *container,␣
↪uint8_t *pPayload)
{
    GAPConnectionEventConnectedIndication_t *evt = &(container->Data.
↪GAPConnectionEventConnectedIndication);

    uint32_t idx = 0;
```

(continues on next page)

(continued from previous page)

```
    /* Store (OG, OC) in ID */
    container->id = 0x489D;

    evt->DeviceId = pPayload[idx]; idx++;
    FLib_MemCpy(&(evt->ConnectionParameters.ConnInterval), pPayload + idx, sizeof(evt->
↪ConnectionParameters.ConnInterval)); idx += sizeof(evt->ConnectionParameters.ConnInterval);
    FLib_MemCpy(&(evt->ConnectionParameters.ConnLatency), pPayload + idx, sizeof(evt->
↪ConnectionParameters.ConnLatency)); idx += sizeof(evt->ConnectionParameters.ConnLatency);
    FLib_MemCpy(&(evt->ConnectionParameters.SupervisionTimeout), pPayload + idx, sizeof(evt->
↪ConnectionParameters.SupervisionTimeout)); idx += sizeof(evt->ConnectionParameters.
↪SupervisionTimeout);
    evt->ConnectionParameters.MasterClockAccuracy = (GAPConnectionEventConnectedIndication_
↪ConnectionParameters_MasterClockAccuracy_t)pPayload[idx]; idx++;
    evt->PeerAddressType = (GAPConnectionEventConnectedIndication_PeerAddressType_
↪t)pPayload[idx]; idx++;
    FLib_MemCpy(evt->PeerAddress, pPayload + idx, 6); idx += 6;
    evt->peerRpaResolved = (bool_t)pPayload[idx]; idx++;
    evt->localRpaUsed = (bool_t)pPayload[idx]; idx++;

    return MEM_SUCCESS_c;
}
```

- Event status console printer in evt_printer_<name>.c

```
void SHELL_BleEventNotify(void *param)
{
    bleEvtContainer_t *container = (bleEvtContainer_t *)param;

    switch (container->id) {
        [...]
        case 0x489D:
            shell_write("GAPConnectionEventConnectedIndication");
            shell_write(" -> ");
            switch (container->Data.GAPConnectionEventConnectedIndication.PeerAddressType)
            {
                case GAPConnectionEventConnectedIndication_PeerAddressType_gPublic_c:
                    shell_write(gPublic_c);
                    break;
                case GAPConnectionEventConnectedIndication_PeerAddressType_gRandom_c:
                    shell_write(gRandom_c);
                    break;
                default:
                    shell_printf("Unrecognized status 0x%02X", container->Data.
↪GAPConnectionEventConnectedIndication.PeerAddressType);
                    break;
            }
            break;
        [...]
    }
}
```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Host API Python Bindings

Prerequisites Python 3.10.x/32b is necessary to run the Python bindings. The bindings use the Host API C libraries. On Linux and OS X operating systems, these are called from the installation

location which is /usr/local/lib, while on Windows OS the library file is loaded in <Python Install Directory>\DLLs.

Platform setup To run scripts from the command line, the PYTHONPATH must be set accordingly, so that the interpreter can find the imported modules.

Linux OS

Adding the source folder to the PYTHONPATH can be done by editing ~/.bashrc and adding the following line:

```
export PYTHONPATH=\$PYTHONPATH:/home/.../hsdk-python/src
```

Most of the Python scripts operate on boards connected on a serial bus and superuser privileges must be employed to access the ports. After running a command prefixed with sudo, the environment paths become those of root, so the locally set PYTHONPATH is not visible anymore. That is why /etc/sudoers is modified to keep the environment variable when changing user.

Edit /etc/sudoers with your favorite text editor. Modify:

```
Defaults env_reset -> Defaults env_keep=”PYTHONPATH”
```

As an alternative to avoid modifying the :sudoers file, the PYTHONPATH can be adjusted programmatically, as in the example below:

```
import sys

if sys.platform.startswith('linux'):
    sys.path.append('/home/user/hsdk-python/src')
```

Windows OS

Add the source folder to the PYTHONPATH by following these steps:

1. Navigate to My Computer > Properties > Advanced System Settings > Environment Variables > System Variables.
2. Modify existing or create new variable named PYTHONPATH, with the absolute path of tools\wireless\host_sdk\hsdk-python\src.

Directory Tree

```
lib                                 Compiled host SDK libraries for Windows.
    README.md
    x64
        HSDK.dll
    x86
        HSDK.dll
src
    com
        nxp
            wireless_connectivity
                commands
                    ble             Generated files for Bluetooth LE support.
                        ble_sig_defines.py
                        enums.py
                        events.py
                        frames.py
                        gatt_database_dynamic.py
                        heart_rate_interface.py
                        ___init___.py
                        operations.py
```

(continues on next page)

(continued from previous page)

```
                spec.py
                sync_requests.py
            comm.py
            firmware                Generated files for OTA/FSCI bootloader support.
                enums.py
                events.py
                frames.py
                ___init___.py
                operations.py
                spec.py
                sync_requests.py
            fsci_data_packet.py
            fsci_frame_description.py
            fsci_operation.py
            fsci_parameter.py
            ___init___.py
        hsdk
            CFsciLibrary.py
| |         config.py              Configuration file for the Python Host SDK subsystem.
            CUartLibrary.py
            device
                device_manager.py
                ___init___.py
                physical_device.py
            framing
                fsci_command.py
                fsci_framer.py
                ___init___.py
            ___init___.py
            library_loader.py
            ota_server.py
            singleton.py
            sniffer.py
            utils.py
        ___init___.py
        test                        Test and proof of concept scripts.
            hrs.py                  Script implementing a Bluetooth LE Heart Sensor profile .
            ___init___.py
    ___init___.py
___init___.py
___init___.py
```

Functional description The interaction between Python and the C libraries is made by the ctypes module. Ctypes provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python. Because the use of shared libraries is a requirement, the LIB_OPTION variable must remain set on “dynamic” in hsdk/Makefile. Ctypes made into mainline Python starting with version 2.5.

The Python Bindings expose Bluetooth LE familiar API in the com.nxp.wireless_connectivity.commands package. Such a package contains the following modules:

```
bluetooth le
  enums.py          – Classes that resemble enums, constants are generated here.
  events.py         – Observer classes that can build an object from a byte array and deliver it to the user.
  frames.py         – Classes that map on Bluetooth LE/Firmware FSCI messages.
  operations.py     – Each Operation class encapsulates a request and one or multiple events that are to be␣
↪generated by the request.
  spec.py           – This file describes the name, size, order and relationship between the command␣
↪parameters.
```

(continues on next page)

(continued from previous page)

```
  sync_requests.py    – Each Synchronous Request is a method which sends a request and returns the␣
↪triggered event.
```

Tests and examples are placed in the package com.nxp.wireless_connectivity.test.

Bluetooth LE Heart Rate Service use case The Heart Rate Service is presented as use case for using the API of a Bluetooth LE *black box*, located in the example hsdk-python/src/com/nxp/wireless_connectivity/test/hrs.py.

The example populates the GATT Database dynamically with the GATT, GAP, heart rate, battery and device information services. It then configures the Bluetooth LE stack and starts advertising. There are also two connect and disconnect observers to handle specific events.

The user needs to connect to the Bluetooth LE or hybrid *black box* through a serial bus port that is passed as a command line argument, for example, /dev/ttyACM0.

```
# python hrs.py -h
usage: hrs.py [-h] [-p] serial_port

Bluetooth LE demo app which implements a ble_fsci_heart_rate_sensor.

positional arguments:
  serial_port Kinetis-W system device node.
optional arguments:
  -h, --help  show this help message and exit
  -p, --pair  Use pairing
```

It is important to first execute a CPU reset request to the Bluetooth LE *black box* before performing any other configuration to reset the Buetooth LE stack. This is done by the following command:

```
FSCICPUReset(serial_port, protocol=Protocol.BLE)
```

User sync request example

It is recommended for the user to access the Bluetooth LE API through sync requests. GATTDBDynamicAddCharacteristicDeclarationAndValue API is used as an example:

```
def gattdb_dynamic_add_cdv(self, char_uuid, char_prop, maxval_len, initval, val_perm):
    '''
    Declare a characteristic and assign it a value.

    @param char_uuid: UUID of the characteristic
    @param char_prop: properties of the characteristic
    @param maxval_len: maximum length of the value
    @param initval: initial value
    @param val_perm: access permissions on the value
    @return: handle of the characteristic
    '''
    ind = GATTDBDynamicAddCharacteristicDeclarationAndValue(
        self.serial_port,
        UuidType=UuidType.Uuid16Bits,
        Uuid=char_uuid,
        CharacteristicProperties=char_prop,
        MaxValueLength=maxval_len,
        InitialValueLength=len(initval),
        InitialValue=initval,
        ValueAccessPermissions=val_perm,
        protocol=self.protocol
    )
```

(continues on next page)

(continued from previous page)

```
    if ind is None:
        return self.gattdb_dynamic_add_cdv(char_uuid, char_prop, maxval_len, initval, val_perm)

    print '\tCharacteristic Handle for UUID 0x%04X ->' % char_uuid, ind.CharacteristicHandle
    self.handles[char_uuid] = ind.CharacteristicHandle
    return ind.CharacteristicHandle
```

Sync request internal implementation

As an example, for the GATTDBDynamicAddCharacteristicDeclarationAndValue API, the command is executed through a synchronous request. The sync request code creates an object of the following class:

```
class GATTDBDynamicAddCharacteristicDeclarationAndValueRequest(object):

    def __init__(self,
UuidType=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestUuidType.Uuid16Bits, Uuid=[],
CharacteristicProperties=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestCharacteristicProperties.
↪gNone_c,
MaxValueLength=bytearray(2), InitialValueLength=bytearray(2), InitialValue=[],
ValueAccessPermissions=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestValueAccessPermissions.
↪gPermissionNone_c):
        '''
        @param UuidType: UUID type
        @param Uuid: UUID value
        @param CharacteristicProperties: Characteristic properties
        @param MaxValueLength: If the Characteristic Value length is variable, this is the maximum length; for␣
↪fixed lengths this must be set to 0
        @param InitialValueLength: Value length at initialization; remains fixed if maxValueLength is set to 0,␣
↪otherwise cannot be greater than maxValueLength
        @param InitialValue: Contains the initial value of the Characteristic
        @param ValueAccessPermissions: Access permissions for the value attribute
        '''
        self.UuidType = UuidType
        self.Uuid = Uuid
        self.CharacteristicProperties = CharacteristicProperties
        self.MaxValueLength = MaxValueLength
        self.InitialValueLength = InitialValueLength
        self.InitialValue = InitialValue
        self.ValueAccessPermissions = ValueAccessPermissions
```

An operation is represented by an object of the following class:

```
class GATTDBDynamicAddCharacteristicDescriptorOperation(FsciOperation):

    def subscribeToEvents(self):
        self.spec = Spec.GATTDBDynamicAddCharacteristicDescriptorRequestFrame
        self.observers =[GATTDBDynamicAddCharacteristicDescriptorIndicationObserver(
↪'GATTDBDynamicAddCharacteristicDescriptorIndication'), ]
        super(GATTDBDynamicAddCharacteristicDescriptorOperation, self).subscribeToEvents()
```

The Spec object is initialized and set to zero in the FSCI packet any parameter not passed through the object of a class, depending on its length. Also, when defining such an object, the parameters may take simple integer, boolean or even list values instead of byte arrays, the values are serialized as a byte stream.

The observer is an object of the following class:

```
class GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationObserver(Observer):
    opGroup = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.opGroup
    opCode = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.opCode
```

(continues on next page)

(continued from previous page)

```
    @overrides(Observer)
    def observeEvent(self, framer, event, callback, sync_request):
    # Call super, print common information
    Observer.observeEvent(self, framer, event, callback, sync_request)
    #Get payload
    fsciFrame = cast(event, POINTER(FsciFrame))
    data = cast(fsciFrame.contents.data, POINTER(fsciFrame.contents.length * c_uint8))
    packet = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.
↪getFsciPacketFromByteArray(data.contents, fsciFrame.contents.length)
    # Create frame object
    frame = GATTDBDynamicAddCharacteristicDeclarationAndValueIndication()
    frame.CharacteristicHandle = packet.getParamValueAsNumber("CharacteristicHandle")
    framer.event_queue.put(frame) if sync_request else None

    if callback is not None:
        callback(frame)
    else:
        print_event(self.deviceName, frame)
    fsciLibrary.DestroyFSCIFrame(event)
```

The status of the request is printed at the console by the following general status handler:

```
def subscribe_to_async_ble_events_from(device, ack_policy=FsciAckPolicy.GLOBAL):
    ble_events = [
        L2CAPConfirmObserver('L2CAPConfirm'),
        GAPConfirmObserver('GAPConfirm'),
        GATTConfirmObserver('GATTConfirm'),
        GATTDBConfirmObserver('GATTDBConfirm'),
        GAPGenericEventInitializationCompleteIndicationObserver(
↪'GAPGenericEventInitializationCompleteIndication'),
        GAPAdvertisingEventCommandFailedIndicationObserver(
↪'GAPAdvertisingEventCommandFailedIndication'),
        GATTServerErrorIndicationObserver('GATTServerErrorIndication'),
        GATTServerCharacteristicCccdWrittenIndicationObserver(
↪'GATTServerCharacteristicCccdWrittenIndication')
    ]

    for ble_event in ble_events:
        FsciFramer(device, FsciAckPolicy.GLOBAL, Protocol.BLE, Baudrate.BR115200).addObserver(ble_
↪event)
```

Connect and disconnect observers

The following code adds observers for the connect and disconnect events in the user class:

```
class BLEDevice(object):
    '''
    Class which defines the actions performed on a generic Bluetooth LE device.
    Services implemented: GATT, GAP, Device Info.
    '''
    self.framer.addObserver(
        GAPConnectionEventConnectedIndicationObserver('GAPConnectionEventConnectedIndication'),
        self.cb_gap_conn_event_connected_cb)
    self.framer.addObserver(GAPConnectionEventDisconnectedIndicationObserver(
↪'GAPConnectionEventDisconnectedIndication'),
    self.cb_gap_conn_event_disconnected_cb)
```

where the callbacks are:

```
def cb_gap_conn_event_connected_cb(self, event):
    '''
```

(continues on next page)

(continued from previous page)

```
    Callback executed when a smartphone connects to this device.
    @param event: GAPConnectionEventConnectedIndication
    '''
    print_event(self.serial_port, event)
    self.client_device_id = event.DeviceId
    self.gap_event_connected.set()

def cb_gap_conn_event_disconnected_cb(self, event):
    '''
    Callback executed when a smartphone disconnects from this device.
    @param event: GAPConnectionEventdisConnectedIndication
    '''
    print_event(self.serial_port, event)
    self.gap_event_connected.clear()
```

From an Android™ or iOS-based smartphone, the user can use the Bluetooth LE Toolbox application in the Heart Rate profile. Random heart rate measurements in the range 60-100 are displayed every second, while battery values change every 10 seconds.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

FSCI Messages XML

Overview The Framework Serial Communication Interface (FSCI) module supports interfacing the BLE Host Stack with a host or a PC tool (Test Tool for Connectivity Products) using a serial communication line. If the FSCI is used with the NXP Test Tool for Connectivity products, the tool can use an XML file containing a detailed meta-description for serial commands and events. This XML file is provided in the software packages that include this document.

Bluetooth LE Host Stack interface set

The Bluetooth LE Host Stack layers that offer access using FSCI are L2CAP, L2CAP CB, GATT, GATTDB, GAP, FSCI, and NVM. Each layer provides primitives that an upper layer (profile/application) uses to access services of that layer. This document includes a detailed list of the provided primitives.

Basic requirements

Following are the hardware requirements for using FSCI API RM:

- IAR Embedded Workbench for Arm architecture
- NXP development boards (to be loaded with the **ble_fscibb** application)
- Optional Bluetooth LE sniffer for testing
- PC tool (Test Tool for Connectivity Products) or another host processor capable of communicating through UART, USB, SPI, or I2C

FSCI message description This chapter describes the L2CAP, L2CAP CB, GATT, GATTDB, GAP, FSCI, and NVM commands (requests) and events (confirms and indications) in detail.

The BLE.xml document is found in the ble_fsci_app subfolder, and provides an XML representation of all these commands and events. The BLE_x.x.x.xml document is found in the ..\tools\wireless\xml_fsci\ subfolder, and provides an XML representation of all these commands and events.

How to interpret the XML

The XML is structured in <Commands> and <Events> tags, the contents grouped by layer. Each group contains the description <GroupDesc> and multiple <Cmd> tags (as shown below).

```
<BLE>
 <Commands>
  <L2CAP>
   <GroupDesc>L2CAP Commands</GroupDesc>
   <Cmd>
    <CmdName>L2CAP-Init.Request</CmdName>
    <CmdDesc>L2CAP initialization function</CmdDesc>
    <CmdHeader>41 01</CmdHeader>
    <CmdParms>
    </CmdParms>
   </Cmd>
  [...]
  </L2CAP>
  <L2CAPCB>
  [...]
  </L2CAPCB>
 [...]
 </Commands>
 <Events>
  [...]
 </Events>
 [...]
</BLE>
```

Every command or event is described in a <Cmd> that contains:

- the name - <CmdName>
- the description - <CmdDesc>
- the opGroup and opCode - <CmdHeader>
- the parameters - <CmdParms>

The parameters of a command/event are specified using the <Parm> tag and contain:

- the name - <ParmName>
- the description - <ParmDesc>
- the size - <ParmSize>
- the type - <ParmType>
- the last value - <ParmLastValue>
- the default value - <ParmDefaultValue>
- the type specifics - <ParmTypeSpecifics>

The type specifics tag is used where needed to specify certain behaviours:

- <ParmItem> for specifying *enum* items
- <ParmState> for specifying the state of a variable (e.g. *ReadOnly*)
- <ParmRange> for specifying the range of the values taken by a variable
- <Depend> for specifying a dependence on another variable (e.g.an *array*'s dependence on the length variable)
- <Parm> for specifying the parameters of a *struct*

FSCI message example For example, the L2CAP-SendAttData.Request command has the following XML structure:

```
<Cmd>
 <CmdName>L2CAP-SendAttData.Request</CmdName>
 <CmdDesc>Sends a data packet through ATT Channel</CmdDesc>
 <CmdHeader>41 03</CmdHeader>
 <CmdParms>
  <Parm>
   <ParmName>DeviceId</ParmName>
   <ParmDesc>The DeviceId for which the command is intended</ParmDesc>
   <ParmSize>1</ParmSize>
   <ParmType>tInt</ParmType>
   <ParmLastValue>00</ParmLastValue>
   <ParmDefaultValue>00</ParmDefaultValue>
  </Parm>
  <Parm>
   <ParmName>PacketLength</ParmName>
   <ParmDesc>Length of the ATT data packet to be sent</ParmDesc>
   <ParmSize>2</ParmSize>
   <ParmType>tInt</ParmType>
   <ParmLastValue>0001</ParmLastValue>
   <ParmDefaultValue>0001</ParmDefaultValue>
   <ParmTypeSpecifics>
    <ParmRange>0x0001-0xFFFF</ParmRange>
   </ParmTypeSpecifics>
  </Parm>
  <Parm>
   <ParmName>Packet</ParmName>
   <ParmDesc>The ATT data packet to be transmitted</ParmDesc>
   <ParmSize>1</ParmSize>
   <ParmType>tArray</ParmType>
   <ParmLastValue>00</ParmLastValue>
   <ParmDefaultValue>00</ParmDefaultValue>
   <ParmTypeSpecifics>
    <Depend>
     <DependName>PacketLength</DependName>
     <DependMask>0xFFFF</DependMask>
     <DependShift>0x0000</DependShift>
    </Depend>
   </ParmTypeSpecifics>
  </Parm>
 </CmdParms>
</Cmd>
```

This translates to:

Tag	Value	Description
<CmdName	L2CAP-SendAttData.Request	Name of the command/event
<CmdDesc	Sends a data packet through ATT Channel	Description of the command/event
<CmdHead	41 03	opGroup and opCode of the command/event in hexadecimal format
<CmdParm	<Parm>	List of the parameters used in the command/event
<ParmNam	DeviceId	Name of the parameter
<ParmDesc	The DeviceId for which the command is intended	Description of the parameter
<ParmSize	1	Parameter size (in bytes)
<ParmType	tInt	Type of the parameter
<ParmLast	00	Last value of the parameter in hexadecimal format
<ParmDefa	00	Default value of the parameter in hexadecimal format
<ParmType	<ParmRange>	Specify range of the parameter
	<Depend>	Specify dependency of the parameter “Packet” of type “tArray” of the variable <DependName> that is the PacketLength

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

References For more information, refer to the NXP website or contact your local Field Application Engineer (FAE).

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Acronyms and Abbreviations

Acronym	Description
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
FSCI	Framework Serial Communication Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GATTDB	Generic Attribute Profile Database
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaptation
L2CAP CB	Logical Link Control and Adaptation Credit Based
SM	Security Manager

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Bluetooth Low Energy Connection Handover

Introduction Connection Handover is an NXP proprietary feature that enables a Bluetooth Low Energy connection to be seamlessly transferred from one device to another while the connected peer remains unaware.

This document explains how to integrate the NXP Bluetooth Low Energy Connection Handover feature in an application and provides detailed explanation of the most commonly used APIs and code examples.

Prerequisites The Bluetooth Low Energy Connection Handover feature can be integrated in an application that fulfills the following requirements:

- Supported platforms: KW45, KW47.
- Bluetooth Low Energy application that is using the lib_ble_OPT_host_cm33_iar.a host library.
- Communication channel between the Source and Target devices involved in the Connection Handover procedure.

Integration and APIs

Integration All the APIs referenced in this section are available in the connection handover library.

The available libraries are listed below:

- lib_ble_handover_cm33_iar.a (for IAR projects)
- lib_ble_handover_cm33_gcc.a (for MCUX or armgcc projects)

Parent topic:*Bluetooth LE Connection Handover Integration and APIs*

Connection Handover API availability

Gap_HandoverGetDataSize

```
bleResult_t Gap_HandoverGetDataSize(deviceId_t deviceId, uint32_t *pDataSize);
```

Description: Returns the size in octets of the Handover Data for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- pDataSize: Pointer to the size in octets of the Handover Data.

Returns: gBleSuccess_c or error.

Gap_HandoverGetData

```
bleResult_t Gap_HandoverGetData(deviceId_t deviceId, uint8_t *pData);
```

Description: Copies the data used by the Handover feature for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- pData: Pointer to memory location where the Handover data is to be copied.

Returns: gBleSuccess_c or error.

Gap_HandoverSetData

```
bleResult_t Gap_HandoverSetData(uint8_t *pData);
```

Description: Sets the data used by the Handover feature.

Parameters:

- pData: Pointer to memory location where the Handover data is found.

Returns: gBleSuccess_c or error.

Gap_HandoverSetLlPendingData

```
bleResult_t Gap_HandoverSetLlPendingData(uint16_t connectionHandle, uint8_t *pTxData);
```

Description: Sets the LL data pending on the target device to be transmitted after handover connect.

Parameters:

- connectionHandle: Handover connection identifier.
- pTxData: Pointer to memory location where the pending data is found.

Returns: gBleSuccess_c or error.

Gap_HandoverFreeData

```
bleResult_t Gap_HandoverFreeData(void);
```

Description: Frees reference to handover data provided through Gap_HandoverSetData().

Returns: gBleSuccess_c or gBleInvalidState_c if handover data reference is not set.

Gap_HandoverGetTime

```
bleResult_t Gap_HandoverGetTime(void);
```

Description: Gets timing information from Link Layer to be used in Handover process.

Returns: gBleSuccess_c or error.

Gap_HandoverSuspendTransmit

```
bleResult_t Gap_HandoverSuspendTransmit(deviceId_t deviceId, bleHandoverSuspendTransmitMode_t␣
↪mode, uint16_t eventCounter, uint8_t noOfConnIntervals);
```

Description: Suspends TX for the given connection.

Parameters:

- deviceId: Peer device identifier.
- mode: Mode.
- eventCounter: Event counter.
- noOfConnIntervals: Number of connection intervals during which TX is suspended. Use 0 for manual resume.

Returns: gBleSuccess_c or error.

Gap_HandoverResumeTransmit

```
bleResult_t Gap_HandoverResumeTransmit(deviceId_t deviceId);
```

Description: Resumes TX for the given connection.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorNotification

```
bleResult_t Gap_HandoverAnchorNotification(deviceId_t deviceId, bleHandoverAnchorNotificationEnable_t
↪enable, uint8_t noOfReports);
```

Description: Enables or disables anchor notifications for the given peer.

Parameters:

- deviceId: Peer device identifier.
- enable: Enables or disables notifications.
- noOfReports: Number of reports to receive (0 for manual stop).

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorSearchStart

```
bleResult_t Gap_HandoverAnchorSearchStart(gapHandoverAnchorSearchStartParams_t *pSearchParams);
```

Description: Starts anchor search with the given parameters.

Parameters:

- pSearchParams: Pointer to structure containing anchor search parameters. The structure contains:
  - startTime625: Slot of the anchor point timing of the connection event.
  - startTimeOffset: Slot offset of the anchor point timing of the connection event.
  - lastRxInstant: Last successful access address reception instant (unit 625 us).
  - accessAddress: Access address (4 bytes).
  - crcInit: CRC initialization value (3 bytes).
  - channelMap: Channel map (5 bytes, FF FF FF FF 1F if all channels are used).
  - connInterval: Connection interval (unit 1.25ms).
  - latency: Latency (unit connection interval).
  - supervisionTimeout: Supervision timeout (unit 10ms).
  - eventCounter: Current connection event counter.
  - centralSca: Sleep clock accuracy (0 to 7).
  - role: Role (0 for central and 1 for peripheral).
  - centralPhy: TX/RX PHY.
  - seqNum: Sequence number; bits 1-0: latest received SN and NESN; bits 5-4: latest transmitted SN and NESN.
  - hopAlgo2: Hop algorithm (0/1 for hop algorithm #1 or #2).
  - unmappedChannelIndex: Unmapped channel index (used only for hop algorithm #1).
  - hopIncrement: Hop increment.
  - ucNbReports: Number of connection intervals to monitor (0: continuous, 1-255: automatic stop).
  - uiEventCounterAdvance: Event counter delay to perform the search (0 to start ASAP).
  - timeout: Timeout in connection intervals (0 if no timeout).
  - timingDiffSlot: Slot difference of the LL timing.
  - timingDiffOffset: Slot offset difference of the LL timing.

  - mode: Search mode setting.

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorSearchStop

```
bleResult_t Gap_HandoverAnchorSearchStop(uint16_t connHandle);
```

Description: Stops anchor search for the given connection handle.

Parameters:

- connHandle: Connection handle for which to stop the anchor search.

Returns: gBleSuccess_c or error.

Gap_HandoverTimeSyncTransmit

```
bleResult_t Gap_HandoverTimeSyncTransmit(gapHandoverTimeSyncTransmitParams_t *pTransmitParams);
```

Description: Starts time sync advertising with given parameters.

Parameters:

- pTransmitParams: Pointer to structure containing time sync transmit parameters. The structure contains:
  - enable: Enables or disables the transmit operation (0 = disable, 1 = enable).
  - advChannel: BLE channel to use for advertising (0 to 39).
  - deviceAddress: Packet identifier, can be the BD address or any value (6 bytes).
  - phys: PHY setting (0 = 1M, 1 = 2M, 2 = LR S8, 3 = LR S2).
  - txPowerLevel: TX Power level (0 to 31).
  - txIntervalSlots625: Packet interval in slots (4 to 255).

Returns: gBleSuccess_c or error.

Gap_HandoverTimeSyncReceive

```
bleResult_t Gap_HandoverTimeSyncReceive(gapHandoverTimeSyncReceiveParams_t *pReceiveParams);
```

Description: Starts time sync scanning with the given parameters.

Parameters:

- pReceiveParams: Pointer to structure containing time sync receive parameters. The structure contains:
  - enable: Enables or disables the receive operation (0 = disable, 1 = enable).
  - scanChannel: BLE channel to use for scanning (0 to 39).
  - deviceAddress: Packet identifier, can be the BD address or any value (6 bytes).
  - phys: PHY setting (0 = 1M, 1 = 2M, 2 = LR S8, 3 = LR S2).
  - stopWhenFound: One-shot setting (0 = continue scanning, 1 = stop when packet is found).

Returns: gBleSuccess_c or error.

Gap_HandoverGetConnParams

```
bleResult_t Gap_HandoverGetConnParams(uint16_t connHandle, gapConnectionCallback_t␣
↪connectionCallback, uint8_t anchorNotification);
```

Description: Connects to a peer device initialized through the Connection Handover procedure.

Parameters:

- connHandle: Handover Connection handle.
- connectionCallback: Pointer to the connection event handler function.
- anchorNotification: Anchor search setting after handover, 0 - stop anchor notification, 1 - keep anchor notification.

Returns: gBleSuccess_c or error.

Gap_HandoverDisconnect

```
bleResult_t Gap_HandoverDisconnect(deviceId_t deviceId);
```

Description: Disconnects a peer device for which the connection was handed over to another device.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverInit

```
bleResult_t Gap_HandoverInit(void);
```

Description: Initializes the connection handover feature. Must be called before Ble_Initialize().

Returns: gBleInvalidState_c if Ble_Initialize() was already called, gBleSuccess_c otherwise.

Gap_HandoverSetSkd

```
bleResult_t Gap_HandoverSetSkd(uint8_t nvmIndex, uint8_t *pSkd);
```

Description: Sets the SKD of the connection that is to be handed over. Used only if gAppSecureMode_d is enabled in the application. Requires Handover Data to be set.

Parameters:

- nvmIndex: Bonding Data NVM index.
- pSkd: Pointer to 16 octets SKD.

Returns: gBleInvalidState_c if Ble_Initialize() was already called, gBleSuccess_c otherwise.

Gap_HandoverGetCsLlContext

```
bleResult_t Gap_HandoverGetCsLlContext(deviceId_t deviceId);
```

Description: Copies the LL CS-related context data for the given peer device id.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverSetCsLlContext

```
bleResult_t Gap_HandoverSetCsLlContext(deviceId_t deviceId, uint16_t mask, uint8_t contextLength,␣
↪uint8_t *pContextData);
```

Description: Sets the CS-related context data into the LL, for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- mask: Context data bitmask (used by the LL).
- contextLength: Length of the context data.
- pContextData: Pointer to context data.

Returns: gBleSuccess_c or error.

Gap_HandoverUpdateConnParams

```
bleResult_t Gap_HandoverUpdateConnParams(gapHandoverUpdateConnParams_t *pConnParams);
```

Description: Updates the channel map and/or PHY used in the current anchor monitoring process.

Parameters:

- pConnParams: Pointer to connection parameters to be updated.

Returns: gBleSuccess_c or error.

Gap_HandoverApplyConnectionUpdateProcedure

```
bleResult_t Gap_
↪HandoverApplyConnectionUpdateProcedure(gapHandoverApplyConnectionUpdateProcedure_t␣
↪*pConnParams);
```

Description: Applies the new connection parameters in case of Connection Update Procedure on the active controller.

Parameters:

- pConnParams: Pointer to connection parameters to be applied.

Returns: gBleSuccess_c or error.

Connection Handover Events

The following events can be received through the GAP generic callback during the handover operations:

gHandoverGetComplete_c (0x31)

Description: Event generated when handover data get operation is complete.

Event Data: handoverGetData_t containing:

- status: Command status.
- pData: Pointer to connection handover data.

gHandoverSetComplete_c (0x32) **Description**: Event generated when handover data set operation is complete.

Event Data: handoverSetData_t containing:

- status: Command status.
- pData: Pointer to connection handover data.

gHandoverGetCsLlContextComplete_c (0x33) **Description**: Event generated when handover CS LL context data get operation is complete.

Event Data: handoverGetCsLlContext_t containing:

- status: Command status.
- responseMask: LL context bitmap indicating the Channel Sounding context.
- llContextLength: Context data length.
- llContext: LL Context data.

gHandoverSetCsLlContextComplete_c (0x34) **Description**: Event generated when handover CS LL context data set operation is complete.

Event Data: Status of the operation.

gHandoverGetTime_c (0x35) **Description**: Event generated when Handover Get Time command is complete.

Event Data: handoverGetTime_t containing:

- status: Command status.
- slot: LL timing slot counter (unit 625us).
- us_offset: LL timing slot offset (0 to 624, unit us).

gHandoverSuspendTransmitComplete_c (0x36) **Description**: Event generated when Handover Suspend Transmit command is complete.

Event Data: handoverSuspendTransmitCompleteEvent_t containing:

- connectionHandle: Connection identifier.
- noOfPendingAclPackets: Number of pending ACL packets.
- sizeOfPendingAclPackets: Pending ACL packet data size.
- sizeOfDataTxInOldestPacket: Size of data transmitted and acked in the oldest ACL packet.
- sizeOfDataNAckInOldestPacket: Size of data transmitted but not acked in the oldest ACL packet.

gHandoverResumeTransmitComplete_c (0x37) **Description**: Event generated when Handover Resume Transmit command is complete.

Event Data: handoverResumeTransmitCompleteEvent_t containing:

- connectionHandle: Connection identifier.

gHandoverAnchorNotificationStateChanged_c (0x38) **Description**: Event generated when Handover Anchor Notification command is complete.

Event Data: handoverAnchorNotificationStateChanged_t containing:

- connectionHandle: Connection identifier.

gHandoverAnchorSearchStarted_c (0x39) **Description**: Event generated when Handover Anchor Search Start command is complete.

Event Data: handoverAnchorSearchStart_t containing:

- status: Command status.
- connectionHandle: Connection identifier.

gHandoverAnchorSearchStopped_c (0x3A) **Description**: Event generated when Handover Anchor Search Stop command is complete.

Event Data: handoverAnchorSearchStop_t containing:

- status: Command status.
- connectionHandle: Connection identifier.

gHandoverTimeSyncTransmitStateChanged_c (0x3B) **Description**: Event generated when Handover Time Sync Transmit command is complete.

Event Data: Status of the operation.

gHandoverTimeSyncReceiveComplete_c (0x3C) **Description**: Event generated when Handover Time Sync Receive command is complete.

Event Data: Status of the operation.

gHandoverAnchorMonitorEvent_c (0x3D) **Description**: Event received from Controller - Handover Anchor Monitor.

Event Data: handoverAnchorMonitorEvent_t containing:

- connectionHandle: Connection identifier.
- connEvent: Current connection event counter.
- rssiRemote: RSSI of the packet from the remote device (+127 if not available).
- lqiRemote: LQI (Link Quality Indicator) of the packet from the remote device.
- statusRemote: Status of the packet from the remote device.
- rssiActive: RSSI of the packet from the active device (+127 if not available).
- lqiActive: LQI (Link Quality Indicator) of the packet from the active device.
- statusActive: Status of the packet from the active device.
- anchorClock625Us: Slot of the anchor point timing of the connection event.
- anchorDelay: Slot offset of the anchor point timing of the connection event.
- chIdx: BLE channel index.
- ucNbReports: Number of remaining reports to receive.

gHandoverTimeSyncEvent_c (0x3E) **Description**: Event received from Controller - Handover Time Sync.

Event Data: handoverTimeSyncEvent_t containing:

- txClkSlot: Transmitter packet start time in slot (625 us).
- txUs: Transmitter packet start time offset inside the slot (0 to 624 us).
- rxClkSlot: Receiver packet start time in slot (625 us).
- rxUs: Receiver packet start time offset inside the slot (0 to 624 us).
- rssi: RSSI value.

gHandoverConnParamUpdateEvent_c (0x3F) **Description**: Event received from Controller - Handover Connection Parameters Update.

Event Data: handoverConnParamUpdateEvent_t containing:

- status: Status indicating the event trigger source.
- connectionHandle: Connection identifier.
- ulTxAccCode: Access address.
- aCrcInitVal: CRC initialization value (3 bytes).
- uiConnInterval: Connection interval (unit 1.25ms).
- uiSuperTO: Supervision timeout (unit 10ms).
- uiConnLatency: Latency (unit connection interval).
- aChMapBm: Channel map (5 bytes).
- ucChannelSelection: Hop algorithm (0/1 for algorithm #1 or #2).
- ucHop: Hop increment.
- ucUnMapChIdx: Unmapped channel index.
- ucCentralSca: Sleep clock accuracy.
- ucRole: Role (0 for central and 1 for peripheral).
- aucRemoteMasRxPHY: TX/RX PHY.
- seqNum: Sequence number.
- uiConnEvent: Current connection event counter.
- ulAnchorClk: Slot of the anchor point timing.
- uiAnchorDelay: Slot offset of the anchor point timing.
- ulRxInstant: Last successful access address reception instant.

gHandoverFreeComplete_c (0x46) **Description**: Event generated when handover data free operation is complete.

Event Data: Status of the operation.

gHandoverUpdateConnParamsComplete_c (0x47) **Description**: Event generated when Handover Update Connection Parameters command is complete.

Event Data: handoverUpdateConnParams_t containing:

- status: Command status.

- connectionHandle: Connection identifier.

gHandoverLlPendingData_c (0x4A) **Description**: Event indicating ACL Data pending to be transmitted by the LL.

Event Data: handoverLlPendingDataIndication_t containing:

- dataSize: Pending HCI ACL data packet size.
- pData: Pointer to message containing the HCI ACL data packet (should be freed by the application).

gHandoverAnchorMonitorPacketEvent_c (0x44) **Description**: Event received from Controller - Handover Anchor Monitor Packet.

Event Data: handoverAnchorMonitorPacketEvent_t containing:

- packetCounter: Packet counter, incremented for each event.
- connectionHandle: Connection identifier.
- statusPacket: Status of the packet.
- phy: PHY (0/1/2/3 for 1M/2M/LR S8/LR S2).
- chIdx: BLE channel index.
- rssiPacket: RSSI of the packet (+127 if not available).
- lqiPacket: LQI (Link Quality Indicator) of the packet.
- connEvent: Current connection event counter.
- anchorClock625Us: Slot value of packet start time (in 625us unit).
- anchorDelay: Slot offset value of packet start time (in 1us unit).
- ucNbConnIntervals: Number of remaining connection intervals to monitor.
- pduSize: PDU length.
- pPdu: Pointer to PDU data (must be freed by the application).

gHandoverAnchorMonitorPacketContinueEvent_c (0x45) **Description**: Event received from Controller - Handover Anchor Monitor Packet Continue.

Event Data: handoverAnchorMonitorPacketContinueEvent_t containing:

- packetCounter: Packet counter, incremented for each event.
- connectionHandle: Connection identifier.
- pduSize: PDU length.
- pPdu: Pointer to PDU data (must be freed by the application).

gHandoverConnectionUpdateProcedureEvent_c (0x4F) **Description**: Event reporting new connection parameters indicated during the Connection Update procedure.

Event Data: handoverConnectionUpdateProcedureEvent_t containing:

- connectionHandle: Connection identifier.
- winSize: Transmit window size value (in 1.25 ms units).
- winOffset: Transmit window offset value (in 1.25 ms units).
- interval: Connection interval value (in 1.25 ms units).

- latency: Connection peripheral latency value.
- timeout: Connection supervision timeout value (in 10 ms units).
- instant: Connection event counter value when new parameters will be applied.
- currentEventCounter: Current connection event counter.

gHandoverApplyConnectionUpdateProcedureComplete_c (0x50) **Description**: Event reporting completion of the Connection Update procedure application.

Event Data: handoverApplyConnectionUpdateProcedure_t containing:

- connectionHandle: Connection identifier.

Application common modules The following application common modules are meant to offer a simplified API for generic Connection Handover procedures and allow for quick integration into existing applications.

Handover application common module

Introduction The Connection Handover application common module allows for quick integration of the Connection Handover feature in a Bluetooth LE application.

Integration All the APIs referenced in this section are available in the Connection Handover application common module.

Connection handover requires a communication interface with the other devices involved in the connection handover process.

Perform the following steps to integrate Connection Handover in an application:

1. Add app_handover.c and app_handover.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the Connection Handover application common module by calling AppHandover_Init().
3. Call AppHandover_ProcessA2ACommand() to handle connection handover commands received.
4. Call AppHandover_GenericCallback() in the application generic events callback to handle the following events:
   - gHandoverAnchorSearchStarted_c
   - gHandoverAnchorSearchStopped_c
   - gHandoverAnchorMonitorEvent_c
   - gHandoverSuspendTransmitComplete_c
   - gHandoverGetComplete_c
   - gGetConnParamsComplete_c
   - gHandoverTimeSyncEvent_c
   - gHandoverTimeSyncTransmitStateChanged_c
   - gHandoverTimeSyncReceiveComplete_c
   - gLlSkdReportEvent_c

- gInternalError_c
- gHandoverAnchorMonitorPacketEvent_c
- gHandoverAnchorMonitorPacketContinueEvent_c
- gHandoverFreeComplete_c
- gHandoverLlPendingData_c
- gHandoverConnectionUpdateProcedureEvent_c
- gHandoverApplyConnectionUpdateProcedureComplete_c

5. Call AppHandover_ConnectionCallback in the application connection events callback to handle the following events:
   - gConnEvtHandoverConnected_c
   - gHandoverDisconnected_c
6. To trigger connection handover call AppHandover_SetPeerDevice() followed by AppHandover_StartTimeSync()

Connection Handover application common module APIs

AppHandover_Init

```
bleResult_t AppHandover_Init(appHandoverEventCb_t pfAppEventCb, gapConnectionCallback_t␣
↪pfConnectionCallback, appHandoverA2AInterfaceCb_t pfAppA2AInterfaceCb);
```

Description: Initializes the handover application common module.

Parameters:

- pfAppEventCb: Application callback for handover events.
- pfConnectionCallback: Connection callback for connection events.
- pfAppA2AInterfaceCb: Communication callback for handover data.

Returns: bleResult_t

AppHandover_TimeSyncTransmit

```
bleResult_t AppHandover_TimeSyncTransmit(bleHandoverTimeSyncEnable_t enable);
```

Description: Enables/disables transmission of time synchronization information.

Parameters:

- enable: Enables or disables the transmission of time synchronization parameters.

Returns: bleResult_t

AppHandover_TimeSyncReceive

```
bleResult_t AppHandover_TimeSyncReceive(bleHandoverTimeSyncEnable_t enable);
```

Description: Enables/disables reception of time synchronization information.

Parameters:

- enable: Enables or disables the reception of time synchronization parameters.

Returns: bleResult_t

AppHandover_StartTimeSync

```
bleResult_t AppHandover_StartTimeSync(bool_t bTimeSyncForHandover);
```

Description: Triggers handover time synchronization.

Parameters:

- bTimeSyncForHandover: TRUE if handover is following, FALSE if RSSI sniffing is following.

Returns: bleResult_t

AppHandover_TimeSyncTransmitSetParams

```
void AppHandover_TimeSyncTransmitSetParams(gapHandoverTimeSyncTransmitParams_t *pParams);
```

Description: Sets parameters for the transmission of time synchronization information.

Parameters:

- pParams: Tx params.

Returns: None

AppHandover_TimeSyncReceiveSetParams

```
void AppHandover_TimeSyncReceiveSetParams(gapHandoverTimeSyncReceiveParams_t *pParams);
```

Description: Sets parameters for the reception of time synchronization information.

Parameters:

- pParams: Rx params.

Returns: None

AppHandover_SetPeerDevice

```
void AppHandover_SetPeerDevice(deviceId_t deviceId);
```

Description: Sets the peer device for connection handover.

Parameters:

- deviceId: Peer device identifier.

Returns: None

AppHandover_ProcessA2ACommand

```
void AppHandover_ProcessA2ACommand(uint8_t cmdId, uint32_t cmdLen, uint8_t *pCmdData);
```

Description: Processes handover commands.

Parameters:

- cmdId: Command identifier.
- cmdLen: Command length.
- pCmdData: Command data.

Returns: None

AppHandover_SetPeerSkd

```
bleResult_t AppHandover_SetPeerSkd(uint8_t *pSkd, uint8_t nvmIndex);
```

Description: Sets the peer LL SKD for connection handover with advanced secure mode.

Parameters:

- nvmIndex: Peer device NVM index.
- pSkd:

A2A application common module

Introduction The A2A (Anchor-to-Anchor) application common module implements a sample communication interface between devices involved in the connection handover process.

Integration Perform the following steps to integrate A2A application common module in an application:

1. Add app_a2a_interface.c and app_a2a_interface.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the Connection Handover application common module by calling A2A_Init().
3. Use A2A_SendCommand() as the communication callback for the Connection Handover application common module.
4. Use A2A_SendSetBondingDataCommand() in the application to send the bonding data from the connected anchor to the target anchor after pairing a new device is successful.

A2A application common module APIs

A2A_Init

```
bleResult_t A2A_Init(serial_handle_t pSerialHandle, appA2ADataIndicationCb_t pfDataIndCb);
```

Description: Performs initialization of the Anchor to Anchor serial communication.

Parameters:

- pSerialHandle: Serial interface handle to be used for communication.
- pfConnectionCallback: Application callback for received packets.

Returns: bleResult_t

A2A_SendCommand

```
void A2A_SendCommand(uint8_t opGroup, uint8_t opCode, uint8_t *pPayload, uint16_t len);
```

Description: Send packet over the Anchor to Anchor interface.

Parameters:

- opGroup: Group identifier.
- opCode: Command identifier.
- pPayload: Pointer to command data.
- len: Command length.

Returns: None

A2B application common module

Introduction The A2B (Alice to Bob) application common module takes advantage of the EdgeLock to EdgeLock Key Exchange feature of the EdgeLock Secure Enclave integrated in the KW45 and KW47 to derive a special internal key (E2E key) that may be used to import/export encrypted key material between two devices. This may be used to allow for secure transfer of the key material between the anchors involved in the Connection Handover process. The LTK and IRK included in the bonding data are transferred as encrypted blobs that may only be decrypted by the EdgeLock Secure Enclave of the destination device.

Integration Perform the following steps to integrate A2B application common module in an application:

1. Add app_a2b.c and app_a2b.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the A2B application common module by calling A2B_Init().
3. One of the device involved in the Connection Handover process must be configured with the gA2BInitiator_d macro set to 1, the other must be configured with the gA2BInitiator_d macro set to 0. The initiator device triggers the EdgeLock-to-EdgeLock (E2E) key derivation and local IRK synchronization. Therefore, it should be started last.
4. After initialization the E2E key will be derived and the gSecLibFunctions.pfSecLib_ExportA2BBlob() and gSecLibFunctions.pfSecLib_ImportA2BBlob() functions can be used to export and import encrypted key material between the two devices.

A2B application common module APIs

A2B_Init

```
bleResult_t A2B_Init(appA2BEventCb_t pfAppEventCb, appA2BA2ACommInterfaceCb_t␣
↪pfAppA2AInterfaceCb);
```

Description: Trigger the A2B feature initialization by generating the public key used for the Edgelock to Edgelock key derivation.

Parameters:

- pfAppEventCb: Application callback functions for A2B events.
- pfAppA2AInterfaceCb: Communication callback for A2B data.

Returns: bleResult_t - Result of the operation.

A2B_ProcessA2ACommand

```
void A2B_ProcessA2ACommand(uint8_t cmdId, uint32_t cmdLen, uint8_t *pCmdData);
```

Description: Process A2B commands received through the A2A interface.

Parameters:

- cmdId: Command identifier.
- cmdLen: Command length.
- pCmdData: Command data.

Returns: None

A2B_FreeE2EKey

```
bleResult_t A2B_FreeE2EKey(void);
```

Description: Free the EdgeLock to EdgeLock key.

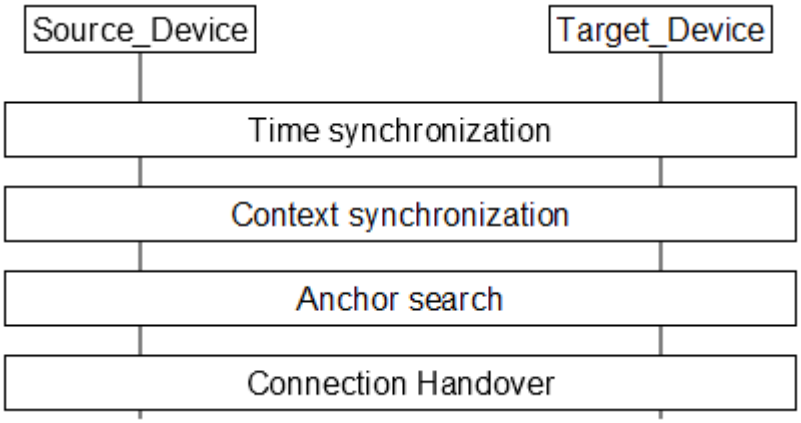
Parameters: None

Returns: bleResult_t - Result of the operation.

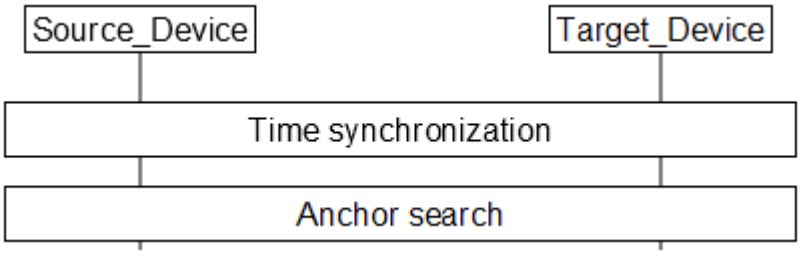
Procedures This section describes the available Connection Handover procedures. A procedure is executed between a device in a connection, referred as the Source device, and another device involved in the Connection Handover or Anchor/Packet Monitoring process, referred to as the Target device.

Procedures - Top level view The Connection Handover and Anchor/Packet Monitoring process comprises several procedures as illustrated below.

Connection Handover



Anchor/Packet monitoring



Individual procedures This section describes each individual procedure in detail.

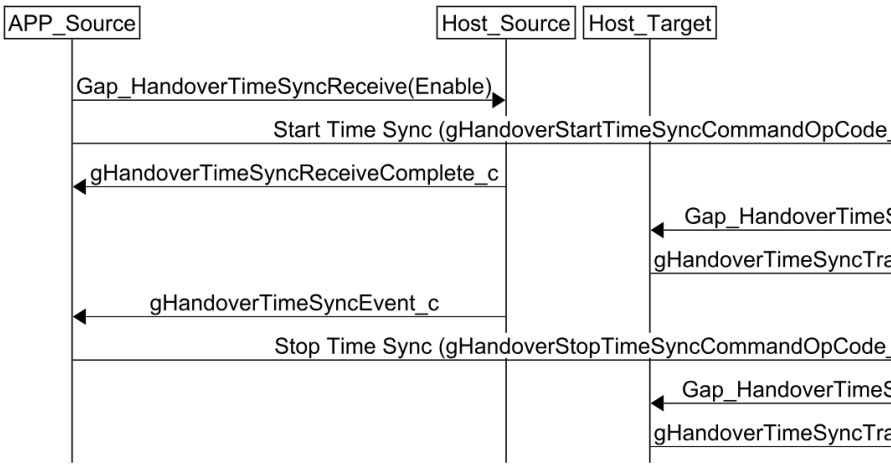
Time synchronization The Time Synchronization procedure enables precise timing alignment between the Source and Target devices for the Connection Handover and Anchor/Packet monitoring processes. The timing information obtained is further required for the Anchor Search procedure.

To start the time synchronization procedure, one of the devices must call GAP_HandoverTimeSyncReceive() and the other device must call GAP_HandoverTimeSyncTransmit(). Either device, Source or Target, can initiate the procedure. The Connection Handover application common module initiates the procedure on the Source device by calling the Gap_HandoverTimeSyncReceive() function.

Time Synchronization steps

Time synchronization steps for two devices where the connected device starts the procedure:

1. Source device application calls GAP_HandoverTimeSyncReceive() with the stopWhenFound parameter set to 'gTimeSyncStopWhenFound_c'.
2. Target device application calls GAP_HandoverTimeSyncTransmit()
3. Source device receives the 'gHandoverTimeSyncReceiveComplete_c' event indicating that Source Link Layer is waiting for the time synchronization event.
4. Target device receives the 'gHandoverTimeSyncTransmitStateChanged_c' event indication that time synchronization transmission is enabled.
5. The Source device receives the 'gHandoverTimeSyncEvent_c' event containing the timing information.
6. The Source device computes the Anchor Search 'timingDiffSlot' and 'timingDiffOffset' using the timing information received in the gHandoverTimeSyncEvent_c event as shown below: 'timingDiffSlot = txClkSlot - rxClkSlot;' 'timingDiffOffset = txUs - rxUs;'



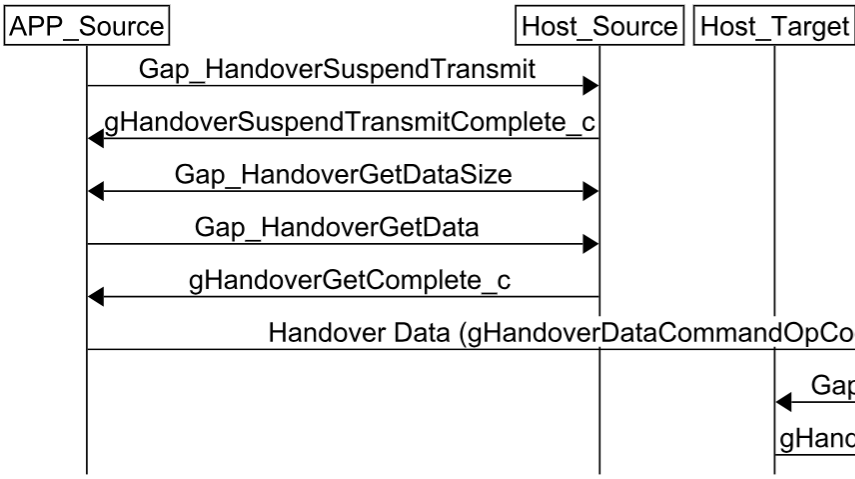
Time Synchronization signaling chart

Context Synchronization In order to transfer an existing connection from a Source device to a Target device, the Host and Link Layer connection context must be synchronized. The Handover Data includes all the required connection information, which must be retrieved from the Source device and transferred to the Target device. Potential issues include instances where the Host or Link Layer context is updated during the Connection Handover process. To avoid such issues, the Bluetooth LE communication between the Source device and the connected device must be suspended until the Connection Handover is complete or aborted. In case the Connection Handover process is aborted, the original connection can be resumed. This procedure is required only for the Connection Handover process and not for the Anchor/Packet Monitoring process.

Context Synchronization steps

1. Source device calls Gap_HandoverSuspendTransmit() to suspend the Bluetooth LE communication with the connected device.
2. After the gHandoverSuspendTransmitComplete_c event is received, the Handover Data may be retrieved using Gap_HandoverGetData(). The application must provided memory for the Handover Data. To obtain the required data size, use the function Gap_HandoverGetDataSize().
3. Wait for the gHandoverGetComplete_c event to confirm the Handover Data retrieval.
4. Transfer the Handover Data from the Source device to the Target device.
5. On the Target device, call Gap_HandoverSetData() to set the received Handover Data.

6. Wait for the gHandoverSetComplete_c event to confirm the Handover Data has been successfully set.

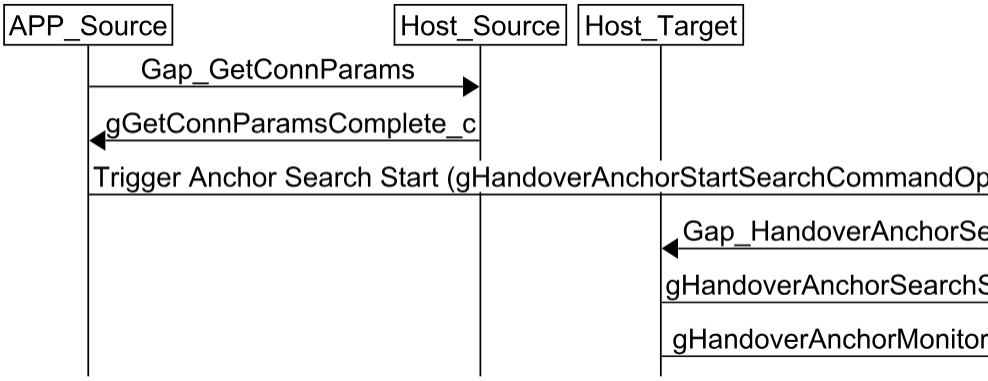


Context Synchronization signaling chart

Anchor Search The Anchor Search procedure takes the connection parameters and timing information obtained in the Time Synchronization procedure to locate the anchor point of the connection between the source device and the connected device.

Anchor Search steps

1. Source device calls Gap_GetConnParams() to retrieve current connection parameters. The connection parameters will be reported through the gGetConnParamsComplete_c event.
2. Send the connection parameters and the timing information obtained from Time Synchronization to the Target device.
3. On the Target device, call Gap_HandoverAnchorSearch() with the received connection parameters and timing information. Set the 'mode' parameter according to the current process:
   - 'gSuspendTxMode_c' for Connection Handover.
   - 'gRssiSniffingMode_c' for Anchor Monitoring.
   - 'gPacketMode_c' for Packet Monitoring.
4. Wait for the gHandoverAnchorSearchStarted_c event to confirm the Anchor Search procedure has started.
5. If Anchor Search is successful the requested events will be reported to the application through the gHandoverAnchorMonitorEvent_c or gHandoverAnchorMonitorPacketEvent_c event.



Anchor Search signaling chart

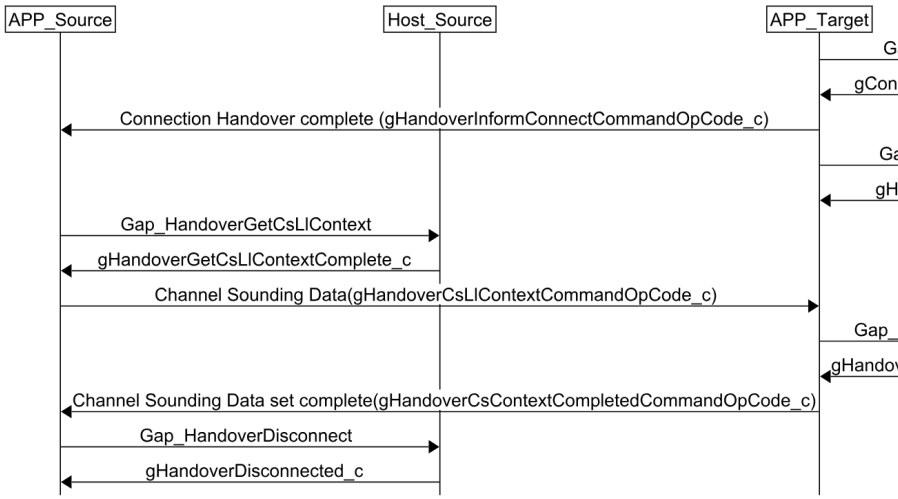
Connection Handover The Connection Handover procedure may be performed after a successful Anchor Search procedure and consists in the Target device taking over the connection from the Source device.

Connection Handover steps

1. Target device calls Gap_HandoverConnect() to initiate the connection handover process.
2. Wait for the gConnEvtHandoverConnected_c event on the Target device. Once received, the connection has been successfully transferred.
3. Notify the Source device that the Target device has successfully connected to the Device.
4. If the application does not include Channel Sounding, call Gap_HandoverDisconnect on the Source device to terminate its connection.
5. The Handover Data is no longer required and may be freed by calling Gap_HandoverFreeData() on the Target device.
6. Application context may be transferred at this point through application specific means. If Channel Sounding is enabled in the application, the Channel Sounding Context must be transferred.

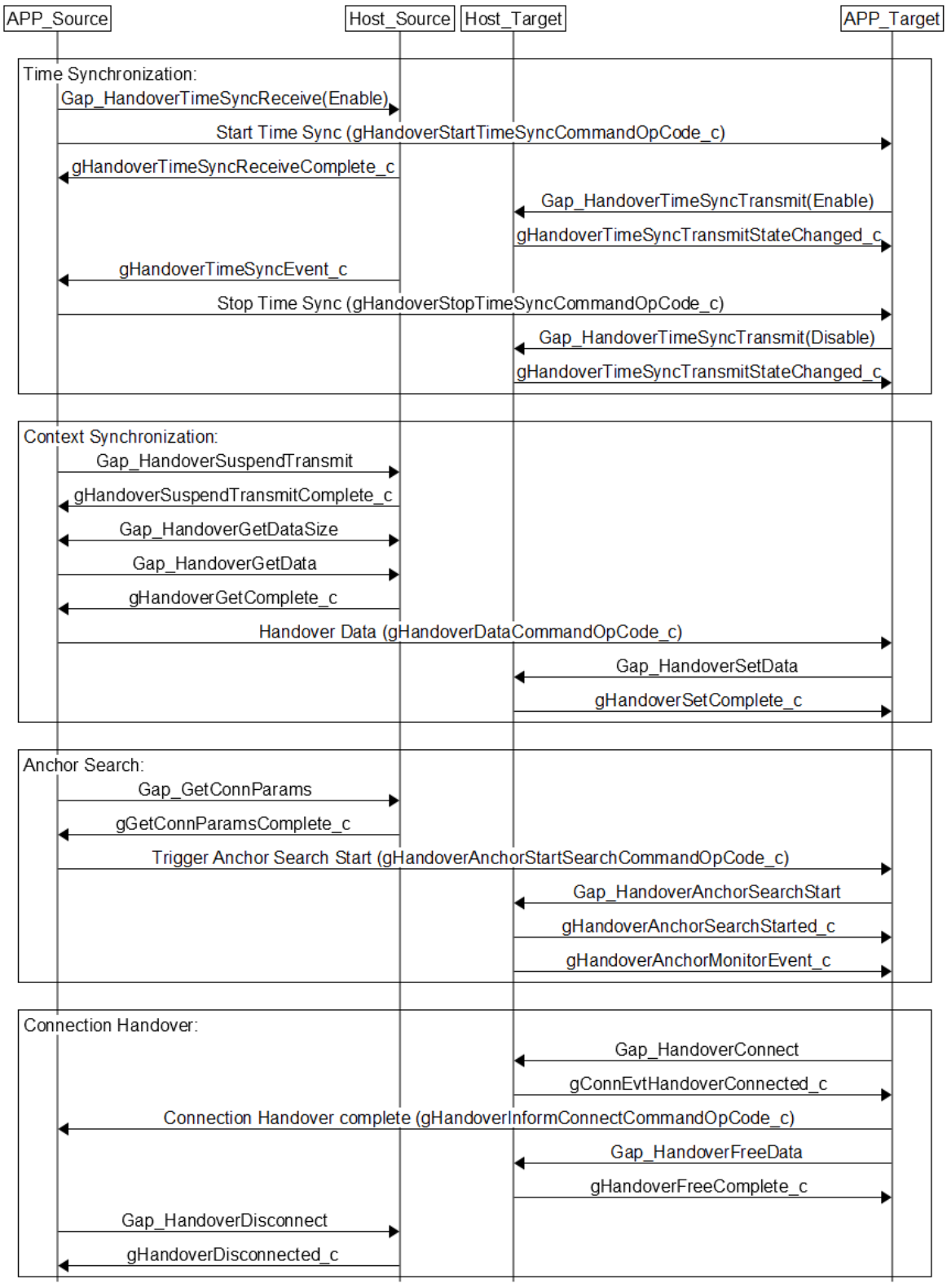
Additional steps for Channel Sounding applications:

1. After step 3 above, retrieve the Channel Sounding context on the Source device by calling Gap_HandoverGetCsLlContext() and wait for thegHandoverGetCsLlContextComplete event.
2. Send the Channel Sounding context to the Target device.
3. Set the context in the Target device by calling Gap_HandoverSetCsLlContext() and wait for the gHandoverSetCsLlContextComplete_c event.
4. Notify the Source device that the Channel Sounding context has been successfully set.
5. Call Gap_HandoverDisconnect() on the Source device to terminate its connection. When complete, the gHandoverDisconnected_c event will be sent.

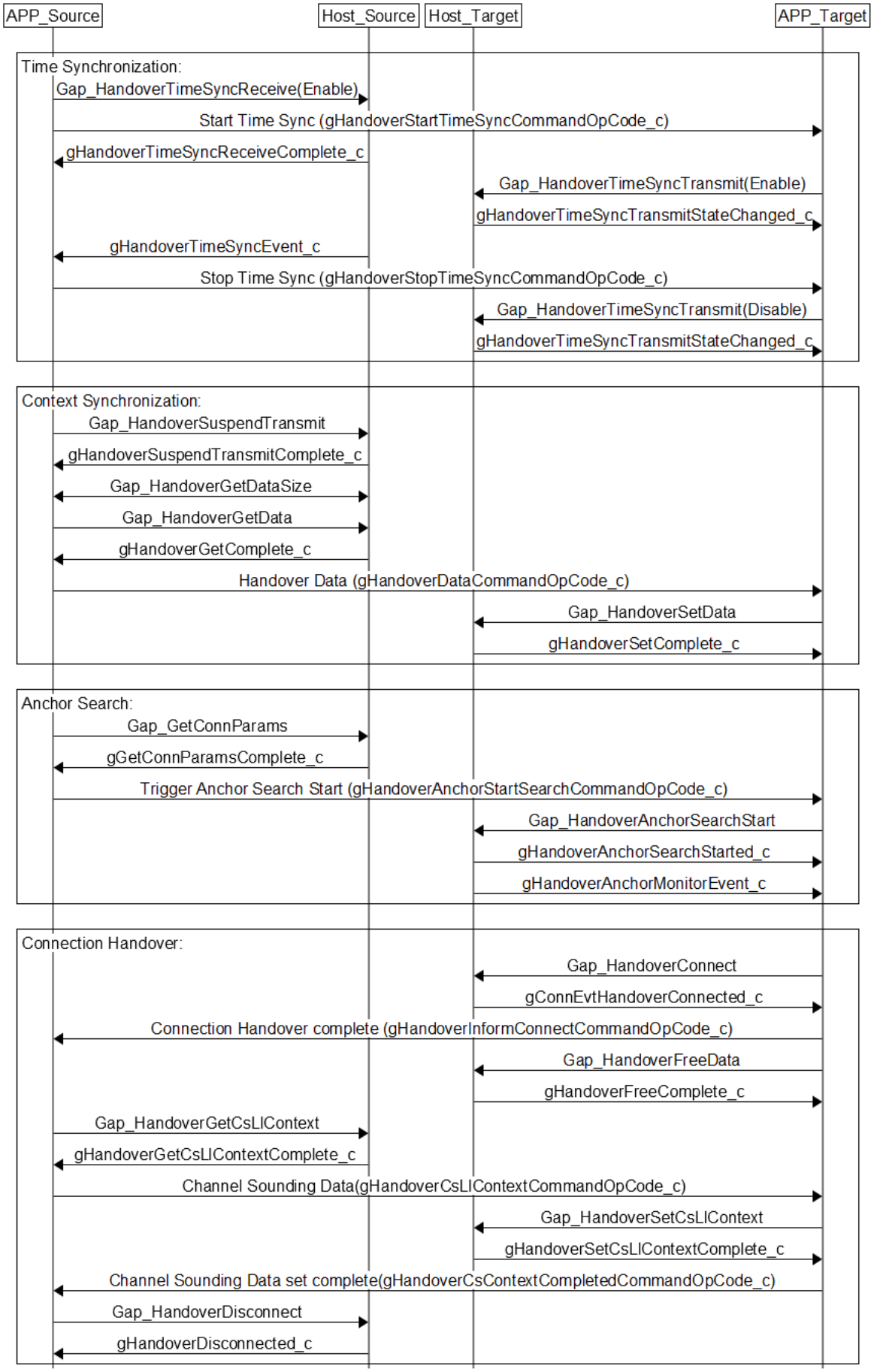


Connection Handover signaling chart

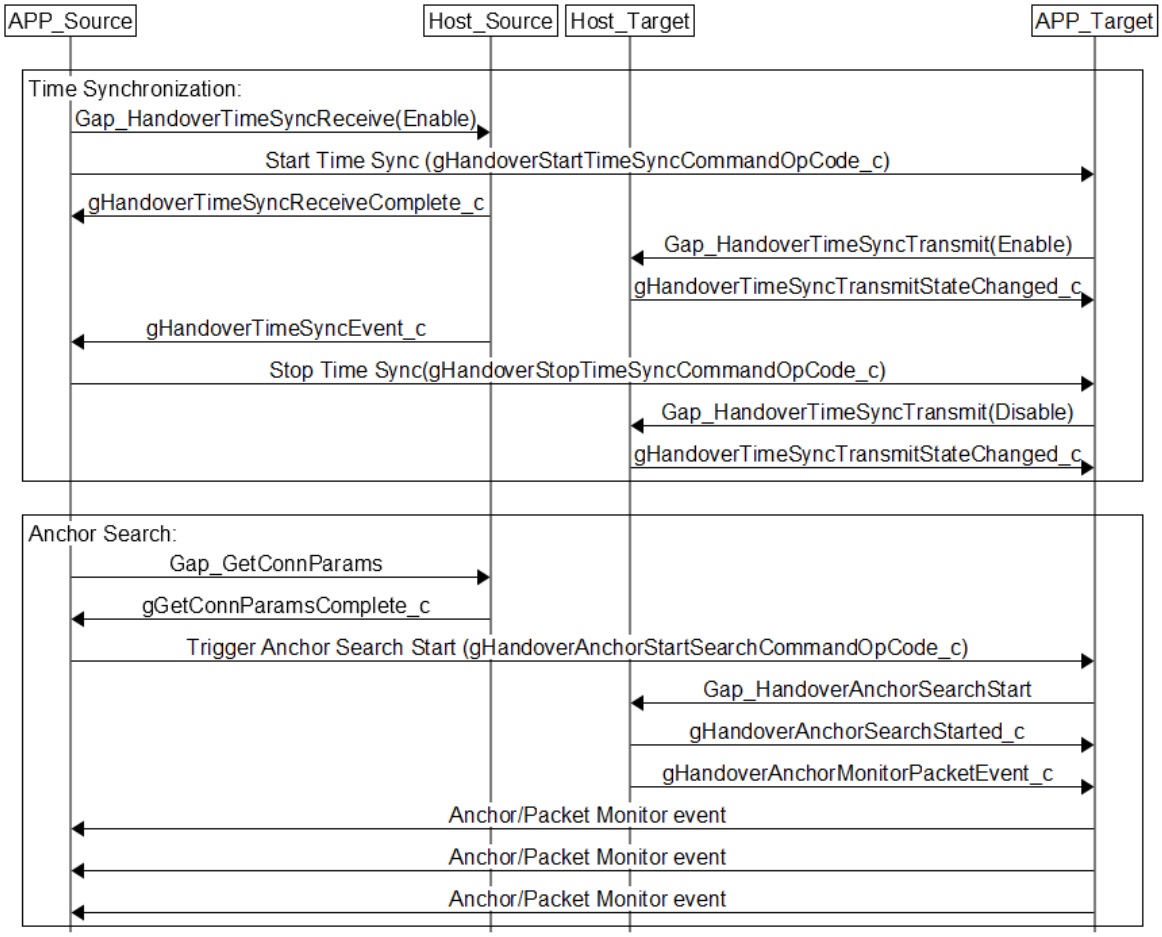
Procedures - Detail view **Connection Handover**



Connection Handover with Channel Sounding


APP_Source
Host_Source
Host_Target
APP_Target
Time Synchronization:
Gap_HandoverTimeSyncReceive(Enable)
Start Time Sync (gHandoverStartTimeSyncCommandOpCode_c)
gHandoverTimeSyncReceiveComplete_c
Gap_HandoverTimeSyncTransmit(Enable)
gHandoverTimeSyncTransmitStateChanged_c
gHandoverTimeSyncEvent_c
Stop Time Sync (gHandoverStopTimeSyncCommandOpCode_c)
Gap_HandoverTimeSyncTransmit(Disable)
gHandoverTimeSyncTransmitStateChanged_c
Context Synchronization:
Gap_HandoverSuspendTransmit
gHandoverSuspendTransmitComplete_c
Gap_HandoverGetDataSize
Gap_HandoverGetData
gHandoverGetComplete_c
Handover Data (gHandoverDataCommandOpCode_c)
Gap_HandoverSetData
gHandoverSetComplete_c
Anchor Search:
Gap_GetConnParams
gGetConnParamsComplete_c
Trigger Anchor Search Start (gHandoverAnchorStartSearchCommandOpCode_c)
Gap_HandoverAnchorSearchStart
gHandoverAnchorSearchStarted_c
gHandoverAnchorMonitorEvent_c
Connection Handover:
Gap_HandoverConnect
gConnEvtHandoverConnected_c
Connection Handover complete (gHandoverInformConnectCommandOpCode_c)
Gap_HandoverFreeData
gHandoverFreeComplete_c
Gap_HandoverGetCsLIContext
gHandoverGetCsLIContextComplete_c
Channel Sounding Data(gHandoverCsLIContextCommandOpCode_c)
Gap_HandoverSetCsLIContext
gHandoverSetCsLIContextComplete_c
Channel Sounding Data set complete(gHandoverCsContextCompletedCommandOpCode_c)
Gap_HandoverDisconnect
gHandoverDisconnected_c

Anchor/Packet Monitoring



Bluetooth Low Energy Host Stack API Reference Manual

Groups

GAP

enum gapRole_t

GAP Role of a BLE device

Values:

enumerator gGapCentral_c

Central scans and connects to Peripherals.

enumerator gGapPeripheral_c

Peripheral advertises and connects to Centrals.

enumerator gGapObserver_c

Observer only scans and makes no connections.

enumerator gGapBroadcaster_c

Broadcaster only advertises and makes no connections.

enum gapKeypressNotification_tag

Values:

enumerator gKnPasskeyEntryStarted_c

Start of the Passkey Entry.

enumerator gKnPasskeyDigitStarted_c

Digit entered.

enumerator gKnPasskeyDigitErased_c

Digit erased.

enumerator gKnPasskeyCleared_c

Passkey cleared.

enumerator gKnPasskeyEntryCompleted_c

Passkey Entry completed.

enum gapScanMode_t

Scan Mode options; used as parameter for Gap_SetScanMode.

Values:

enumerator gDefaultScan_c

Reports all scanned devices to the application.

enumerator gLimitedDiscovery_c

Reports only devices in Limited Discoverable Mode, i.e., containing the Flags AD with the LE Limited Discoverable Flag set.

enumerator gGeneralDiscovery_c

Reports only devices in General Discoverable Mode, i.e., containing the Flags AD with the LE General Discoverable Flag set.

enumerator gAutoConnect_c

Automatically connects with devices with known addresses and does not report any scanned device to the application.

enum gapAdvertisingChannelMapFlags_t

Advertising Channel Map flags - setting a bit activates advertising on the respective channel.

Values:

enumerator gAdvChanMapFlag37_c

Bit for channel 37.

enumerator gAdvChanMapFlag38_c

Bit for channel 38.

enumerator gAdvChanMapFlag39_c

Bit for channel 39.

enum gapAdvertisingFilterPolicy_t

Advertising Filter Policy values

Values:

enumerator gProcessAll_c

Default value: accept all connect and scan requests.

enumerator gProcessConnAllScanWL_c

Accept all connect requests, but scan requests only from devices in Filter Accept List.

enumerator gProcessScanAllConnWL_c

Accept all scan requests, but connect requests only from devices in Filter Accept List.

enumerator gProcessFilterAcceptListOnly_c

Accept connect and scan requests only from devices in Filter Accept List.

enum gapPeriodicAdvSyncMode_tag

Periodic Advertisement Sync Transfer parameters

Values:

enumerator gapPeriodicSyncNoSyncMode_c

No attempt is made to synchronize to the periodic advertising and no gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host

enumerator gapPeriodicSyncNoReports_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be disabled.

enumerator gapPeriodicSyncReportsEnabled_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be enabled with duplicate filtering disabled

enumerator gapPeriodicSyncReportsEnabledWithDF_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be enabled with duplicate filtering enabled

enum gapFilterDuplicates_t

Values:

enumerator gGapDuplicateFilteringDisable_c

Duplicate filtering disabled

enumerator gGapDuplicateFilteringEnable_c

Duplicate filtering enabled

enumerator gGapDuplicateFilteringPeriodicEnable_c

Duplicate filtering enabled, reset for each scan period

enum gapCreateSyncReqFilterPolicy_tag

Values:

enumerator gUseCommandParameters_c

Use the SID, peerAddressType, and peerAddress parameters to determine which advertiser to listen to.

enumerator gUsePeriodicAdvList_c

Use the Periodic Advertiser List to determine which advertiser to listen to

enum gapPrivateKeyType_t

Values:

enumerator gUseGeneratedKey_c

The private key generated by LE_Read_Local_P-256_Public_Key command, used for DH key generation

enumerator gUseDebugKey_c

The private debug key, used for DH key generation

enum gapAdType_t

AD Type values as defined by Bluetooth SIG used when defining *gapAdStructure_t* structures for advertising or scan response data.

Values:

enumerator gAdFlags_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete16bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete16bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete32bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete32bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete128bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete128bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdShortenedLocalName_c

Defined by the Bluetooth SIG.

enumerator gAdCompleteLocalName_c
Defined by the Bluetooth SIG.

enumerator gAdTxPowerLevel_c
Defined by the Bluetooth SIG.

enumerator gAdClassOfDevice_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingHashC192_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingRandomizerR192_c
Defined by the Bluetooth SIG.

enumerator gAdSecurityManagerTkValue_c
Defined by the Bluetooth SIG.

enumerator gAdSecurityManagerOobFlags_c
Defined by the Bluetooth SIG.

enumerator gAdPeripheralConnectionIntervalRange_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList16bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList32bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList128bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData16bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData32bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData128bit_c
Defined by the Bluetooth SIG.

enumerator gAdPublicTargetAddress_c
Defined by the Bluetooth SIG.

enumerator gAdRandomTargetAddress_c
Defined by the Bluetooth SIG.

enumerator gAdAppearance_c
Defined by the Bluetooth SIG.

enumerator gAdAdvertisingInterval_c
Defined by the Bluetooth SIG.

enumerator gAdLeDeviceAddress_c
Defined by the Bluetooth SIG.

enumerator gAdLeRole_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingHashC256_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingRandomizerR256_c
Defined by the Bluetooth SIG.

enumerator gAd3dInformationData_c
Defined by the Bluetooth SIG.

enumerator gAdUniformResourceIdentifier_c
Defined by the Bluetooth SIG.

enumerator gAdLeSupportedFeatures_c
Defined by the Bluetooth SIG.

enumerator gAdChannelMapUpdateIndication_c
Defined by the Bluetooth SIG.

enumerator gAdAdvertisingIntervalLong_c
Defined by the Bluetooth SIG.

enumerator gAdEncryptedAdvertisingData_c
Defined by the Bluetooth SIG.

enumerator gAdManufacturerSpecificData_c
Defined by the Bluetooth SIG.

enum gapDecisionInstructionsRelevantField_t
Decision Instruction Relevant Field Type.

Values:

enumerator gDIRF_ResolvableTag_c

enumerator gDIRF_AdvMode_c

enumerator gDIRF_RSSI_c

enumerator gDIRF_PathLoss_c

enumerator gDIRF_AdvAddress_c

enumerator gDIRF_ArbitraryDataOfExactly_1Byte_c

enumerator gDIRF_ArbitraryDataOfExactly_2Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_3Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_4Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_5Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_6Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_7Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_8Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_1Byte_c

enumerator gDIRF_ArbitraryDataOfAtLeast_2Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_3Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_4Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_5Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_6Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_7Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_8Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_1Byte_c

enumerator gDIRF_ArbitraryDataOfAtMost_2Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_3Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_4Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_5Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_6Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_7Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_8Bytes_c

enum gapDecisionInstructionsTestPassCriteria_t

Values:

enumerator gDITPC_Never_c

enumerator gDITPC_CheckPasses_c

enumerator gDITPC_CheckFails_c

enumerator gDITPC_RelevantFieldPresent_c

enumerator gDITPC_RelevantFieldNotPresent_c

enumerator gDITPC_CheckPassesOrRelevantFieldNotPresent_c

enumerator gDITPC_CheckFailsOrRelevantFieldNotPresent_c

enumerator gDITPC_Always_c

enum gapDecisionInstructionsTestGroup_t

Values:

enumerator gDITG_SameTestGroup_c

enumerator gDITG_NewTestGroup_c

enum gapDecisionInstructionsAdvAChecks_t

Values:

enumerator gDIAAC_AdvAinFilterAcceptList_c

enumerator gDIAAC_AdvAmatchAddress1_c

enumerator gDIAAC_AdvAmatchAddress1orAddress2_c

enum gapRadioPowerLevelReadType_t

Enumeration used by the Gap_ReadRadioPowerLevel function.

Values:

enumerator gTxPowerCurrentLevelInConnection_c
Reading the instantaneous TX power level in a connection.

enumerator gTxPowerMaximumLevelInConnection_c
Reading the maximum TX power level achieved during a connection.

enumerator gTxPowerLevelForAdvertising_c
Reading the TX power on the advertising channels.

enumerator gRssi_c
Reading the Received Signal Strength Indication in a connection.

enum gapControllerTestCmd_t
Enumeration for Controller Test commands.
Values:

enumerator gControllerTestCmdStartRx_c
Start Receiver Test.

enumerator gControllerTestCmdStartTx_c
Start Transmitter Test.

enumerator gControllerTestCmdEnd_c
End Test.

enum gapControllerTestTxType_tag
Values:

enumerator gControllerTestTxPrbs9_c
PRBS9 sequence ‘11111111100000111101’...

enumerator gControllerTestTxF0_c
Repeated ‘11110000’

enumerator gControllerTestTxAA_c
Repeated ‘10101010’

enumerator gControllerTestTxPrbs15_c
PRBS15 sequence

enumerator gControllerTestTxFF_c
Repeated ‘11111111’

enumerator gControllerTestTx00_c
Repeated ‘00000000’

enumerator gControllerTestTx0F_c
Repeated ‘00001111’

enumerator gControllerTestTx55_c

Repeated '01010101'

enum gapSleepClockAccuracy_t

Enumeration for Sleep Clock Accuracy command.

Values:

enumerator gSwitchToMoreAccurateClock_c

enumerator gSwitchToLessAccurateClock_c

enum gapAdvertisingEventType_t

Advertising event type enumeration, as contained in the *gapAdvertisingEvent_t*.

Values:

enumerator gAdvertisingStateChanged_c

Event received when advertising has been successfully enabled or disabled.

enumerator gAdvertisingCommandFailed_c

Event received when advertising could not be enabled or disabled. Reason contained in gapAdvertisingEvent_t.eventData.failReason.

enumerator gExtAdvertisingStateChanged_c

Event received when extended advertising has been successfully enabled or disabled.

enumerator gAdvertisingSetTerminated_c

Event received when advertising in a given advertising set has stopped.

enumerator gExtScanNotification_c

Event indicates that a SCAN_REQ PDU or an AUX_SCAN_REQ PDU has been received by the extended advertiser.

enumerator gPerAdvSubeventDataRequest_c

Event received when doing PAWR.

enumerator gPerAdvResponse_c

Periodic Advertising Response.

enum gapScanningEventType_t

Scanning event type enumeration, as contained in the *gapScanningEvent_t*.

Values:

enumerator gScanStateChanged_c

Event received when scanning had been successfully enabled or disabled, or a Scan duration time-out has occurred.

enumerator gScanCommandFailed_c

Event received when scanning could not be enabled or disabled. Reason contained in gapScanningEvent_t.eventData.failReason.

enumerator gDeviceScanned_c

Event received when an advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.scannedDevice.

enumerator gExtDeviceScanned_c

Event received when an advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.extScannedDevice.

enumerator gPeriodicDeviceScanned_c

Event received when an Periodic advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.periodicScannedDevice.

enumerator gPeriodicDeviceScannedV2_c

Event received when an Periodic advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.periodicScannedDeviceV2.

enumerator gPeriodicAdvSyncEstablished_c

Event received when a sync with a periodic advertiser was established.

enumerator gPeriodicAdvSyncLost_c

Event received when a sync with a periodic advertiser have been lost.

enumerator gPeriodicAdvSyncTerminated_c

Event received when a sync with a periodic advertiser have been terminated.

enumerator gConnectionlessIqReportReceived_c

Event received when the Controller has reported IQ information from the CTE of a received advertising packet

enumerator gMonAdvReportEventReceived_c

Event received when Monitored Advertisers is enabled

enum bleMonAdvCondition_t

Values:

enumerator gBleMonAdvConditionRssiLowThreshold_c

Monitored Advertisers RSSI value below the RSSI low threshold.

enumerator gBleMonAdvConditionRssiHighThreshold_c

Monitored Advertisers RSSI value greater than or equal to the RSSI high threshold.

enum gapConnectionEventType_t

Connection event type enumeration, as contained in the *gapConnectionEvent_t*.

Values:

enumerator gConnEvtConnected_c

A connection has been established. Data in gapConnectionEvent_t.eventData.connectedEvent.

enumerator gConnEvtPairingRequest_c

A pairing request has been received from the peer Central. Data in gapConnectionEvent_t.eventData.pairingEvent.

enumerator gConnEvtPeripheralSecurityRequest_c

A Peripheral Security Request has been received from the peer Peripheral. Data in gapConnectionEvent_t.eventData.peripheralSecurityRequestEvent.

enumerator gConnEvtPairingResponse_c

A pairing response has been received from the peer Peripheral. Data in gapConnectionEvent_t.eventData.pairingEvent.

enumerator gConnEvtAuthenticationRejected_c

A link encryption or pairing request has been rejected by the peer device. Data in gapConnectionEvent_t.eventData.authenticationRejectedEvent.

enumerator gConnEvtPasskeyRequest_c

Peer has requested a passkey (maximum 6 digit PIN) for the pairing procedure. Device should respond with Gap_EnterPasskey.

enumerator gConnEvtOobRequest_c

Out-of-Band data must be provided for the pairing procedure. Central or Peripheral should respond with Gap_ProvideOob.

enumerator gConnEvtPasskeyDisplay_c

The pairing procedure requires this Peripheral to display the passkey for the Central's user.

enumerator gConnEvtKeyExchangeRequest_c

The pairing procedure requires the SMP keys to be distributed to the peer. Data in gapConnectionEvent_t.eventData.keyExchangeRequestEvent.

enumerator gConnEvtKeysReceived_c

SMP keys distributed by the peer during pairing have been received. Data in gapConnectionEvent_t.eventData.keysReceivedEvent.

enumerator gConnEvtLongTermKeyRequest_c

The bonded peer Central has requested link encryption and the LTK must be provided. Peripheral should respond with Gap_ProvideLongTermKey. Data in gapConnectionEvent_t.eventData.longTermKeyRequestEvent.

enumerator gConnEvtEncryptionChanged_c

Link's encryption state has changed, e.g., during pairing or after a reconnection with a bonded peer. Data in gapConnectionEvent_t.eventData.encryptionChangedEvent.

enumerator gConnEvtPairingComplete_c

Pairing procedure is complete, either successfully or with failure. Data in gapConnectionEvent_t.eventData.pairingCompleteEvent.

enumerator gConnEvtDisconnected_c

A connection has been terminated. Data in gapConnectionEvent_t.eventData.disconnectedEvent.

enumerator gConnEvtRssiRead_c

RSSI for an active connection has been read. Data in gapConnectionEvent_t.eventData.rssi_dBm.

enumerator gConnEvtTxPowerLevelRead_c

TX power level for an active connection has been read. Data in gapConnectionEvent_t.eventData.txPowerLevel_dBm.

enumerator gConnEvtPowerReadFailure_c

Power reading could not be performed. Data in gapConnectionEvent_t.eventData.failReason.

enumerator gConnEvtParameterUpdateRequest_c

A connection parameter update request has been received. Data in gapConnectionEvent_t.eventData.connectionUpdateRequest.

enumerator gConnEvtParameterUpdateComplete_c

The connection has new parameters. Data in gapConnectionEvent_t.eventData.connectionUpdateComplete.

enumerator gConnEvtLeDataLengthChanged_c

The new TX/RX Data Length parameters. Data in gapConnectionEvent_t.eventData.rssi_dBm.leDataLengthChanged.

enumerator gConnEvtLeScOobDataRequest_c

Event sent to request LE SC OOB Data (r, Cr and Addr) received from a peer.

enumerator gConnEvtLeScDisplayNumericValue_c

Event sent to display and confirm a Numeric Comparison Value when using the LE SC Numeric Comparison pairing method.

enumerator gConnEvtLeScKeypressNotification_c

Remote Keypress Notification received during Passkey Entry Pairing Method.

enumerator gConnEvtChannelMapRead_c

Channel Map was read for a connection. Data is contained in gapConnectionEvent_t.eventData.channelMap

enumerator gConnEvtChannelMapReadFailure_c

Channel Map reading could not be performed. Data in gapConnectionEvent_t.eventData.failReason.

enumerator gConnEvtChanSelectionAlgorithm2_c

LE Channel Selection Algorithm #2 is used on the data channel connection.

enumerator gConnEvtPairingNoLtk_c

No LTK was found for the Central peer. Pairing shall be performed again.

enumerator gConnEvtPairingAlreadyStarted_c

Pairing process was already started

enumerator gConnEvtIqReportReceived_c

Controller has reported IQ information received from a connected peer. Data in gapConnectionEvent_t.eventData.connIqReport.

enumerator gConnEvtCteRequestFailed_c

CTE Request to a connected peer has failed. Data in gapConnectionEvent_t.eventData.cteRequestFailedEvent.

enumerator gConnEvtCteReceiveParamsSetupComplete_c

Connection CTE receive parameters have been successfully set.

enumerator gConnEvtCteTransmitParamsSetupComplete_c

Connection CTE transmit parameters have been successfully set.

enumerator gConnEvtCteReqStateChanged_c

Controller started or stopped initiating the CTE Request procedure.

enumerator gConnEvtCteRspStateChanged_c

Controller enabled or disabled sending CTE Responses for a connection.

enumerator gConnEvtPathLossThreshold_c

Received a Path Loss Threshold event. Data in gapConnectionEvent_t.eventData.pathLossThreshold.

enumerator gConnEvtTransmitPowerReporting_c

Received a Transmit Power report. Data in gapConnectionEvent_t.eventData.transmitPowerReporting.

enumerator gConnEvtEnhancedReadTransmitPowerLevel_c

Local information has been read from Controller. Data in gapConnectionEvent_t.eventData.transmitPowerInfo.

enumerator gConnEvtPathLossReportingParamsSetupComplete_c

Path Loss Reporting parameters have been successfully set.

enumerator gConnEvtPathLossReportingStateChanged_c

Path Loss Reporting has been enabled or disabled.

enumerator gConnEvtTransmitPowerReportingStateChanged_c

Transmit Power Reporting has been enabled or disabled for local and/or remote Controllers.

enumerator gConnEvtEattConnectionRequest_c

Received Enhanced Connection Request for EATT channels.

enumerator gConnEvtEattConnectionComplete_c

Received Enhanced Connection Response for EATT channels.

enumerator gConnEvtEattChannelReconfigureResponse_c

Received Enhanced Channel Reconfigure Response for EATT channels.

enumerator gConnEvtEattBearerStatusNotification_c

Enhanced Bearer status updated

enumerator gConnEvtHandoverConnected_c

A connection has been established through the Handover feature. Data in gapConnectionEvent_t.eventData.handoverConnectedEvent.

enumerator gHandoverDisconnected_c

A connection has been terminated as a result of connection handover.

enumerator gConnEvtLeSetDataLengthFailure_c

The Set Data Length command has failed.

enumerator gConnEvtSmError_c

Security Manager error occured.

enum gapEattBearerStatus_t

Enumeration for Bearer Status Notification command status.

Values:

enumerator gEnhancedBearerActive_c

enumerator gEnhancedBearerSuspendedNoLocalCredits_c

enumerator gEnhancedBearerNoPeerCredits_c

enumerator gEnhancedBearerDisconnected_c

enumerator gEnhancedBearerStatusEnd_c

enum gapCarSupport_t

Central Address Resolution characteristic state

Values:

enumerator CAR_Unknown

The Central Address Resolution characteristic was not read

enumerator CAR_Unavailable

The device tried to read the Central Address Resolution characteristic, but it's unavailable

enumerator CAR_Unsupported

The device has read the Central Address Resolution characteristic, and the it's value is FALSE

enumerator CAR_Supported

The device has read the Central Address Resolution characteristic, and the it's value is TRUE

enum gapPeriodicAdvListOperation_t

Values:

enumerator gAddDevice_c

enumerator gRemoveDevice_c

enumerator gRemoveAllDevices_c

enum gapAppearance_tag

Appearance characteristic enumeration, also used in advertising.

Values:

enumerator gUnknown_c

enumerator gGenericPhone_c

enumerator gGenericComputer_c

enumerator gGenericWatch_c

enumerator gSportsWatch_c

enumerator gGenericClock_c

enumerator gGenericDisplay_c

enumerator gGenericRemoteControl_c

enumerator gGenericEyeglasses_c

enumerator gGenericTag_c

enumerator gGenericKeyring_c

enumerator gGenericMediaPlayer_c

enumerator gGenericBarcodeScanner_c

enumerator gGenericThermometer_c

enumerator gThermometerEar_c

enumerator gGenericHeartrateSensor_c

enumerator gHeartRateSensorHeartRateBelt_c

enumerator gGenericBloodPressure_c

enumerator gBloodPressureArm_c

enumerator gBloodPressureWrist_c

enumerator gHumanInterfaceDevice_c

enumerator gKeyboard_c

enumerator gMouse_c

enumerator gJoystick_c

enumerator gGamepad_c

enumerator gDigitizerTablet_c

enumerator gCardReader_c

enumerator gDigitalPen_c

enumerator gBarcodeScanner_c

enumerator gGenericGlucoseMeter_c

enumerator gGenericRunningWalkingSensor_c

enumerator gRunningWalkingSensorInShoe_c

enumerator gRunningWalkingSensorOnShoe_c

enumerator gRunningWalkingSensorOnHip_c

enumerator gGenericCycling_c

enumerator gCyclingComputer_c

enumerator gCyclingSpeedSensor_c

enumerator gCyclingCadenceSensor_c

enumerator gCyclingPowerSensor_c

enumerator gCyclingSpeedandCadenceSensor_c

enumerator gGenericPulseOximeter_c

enumerator gFingertip_c

enumerator gWristWorn_c

enumerator gGenericWeightScale_c

enumerator gGenericOutdoorSportsActivity_c

enumerator gLocationDisplayDevice_c

enumerator gLocationandNavigationDisplayDevice_c

enumerator gLocationPod_c

enumerator gLocationAndNavigationPod_c

typedef uint8_t gapIoCapabilities_t

I/O Capabilities as defined by the SMP

typedef uint8_t gapSmpKeyFlags_t

Flags indicating the Keys to be exchanged by the SMP during the key exchange phase of pairing.

typedef uint8_t gapSecurityMode_t

LE Security Mode

typedef uint8_t gapSecurityLevel_t

LE Security Level

typedef uint8_t gapSecurityModeAndLevel_t

Security Mode-and-Level definitions

typedef uint8_t gapKeypressNotification_t

Keypress Notification Types

typedef enum *gapKeypressNotification_tag* gapKeypressNotification_tag

typedef uint8_t gapAuthenticationRejectReason_t

Reason for rejecting the pairing request.

typedef struct *gapExtAdvertisingParameters_tag* gapExtAdvertisingParameters_t

Extended Advertising Parameters; for defaults see gGapDefaultExtAdvertisingParameters_d.

typedef struct *gapPeriodicAdvParameters_tag* gapPeriodicAdvParameters_t

Periodic Advertising Parameters [v2]; for defaults see gGapDefaultPeriodicAdvParameters_d.

typedef enum *gapPeriodicAdvSyncMode_tag* gapPeriodicAdvSyncMode_t

Periodic Advertisement Sync Transfer parameters

typedef struct *gapPeriodicAdvSyncTransfer_tag* gapPeriodicAdvSyncTransfer_t

typedef struct *gapPeriodicAdvSetInfoTransfer_tag* gapPeriodicAdvSetInfoTransfer_t

typedef struct *gapSetPeriodicAdvSyncTransferParams_tag*
gapSetPeriodicAdvSyncTransferParams_t

typedef uint8_t gapCreateSyncReqFilterPolicy_t

Create Sync Request Filter Policy values

typedef struct *gapCreateSyncReqOptions_tag* gapCreateSyncReqOptions_t

Create Sync Request Options

typedef struct *gapPeriodicAdvSyncReq_tag* gapPeriodicAdvSyncReq_t

Periodic Advertising Sync Request parameters

typedef uint8_t gapAdTypeFlags_t

Values of the AD Flags advertising data structure.

typedef uint8_t gapDecisionInstructionsAdvMode_t

typedef struct *gapDecisionInstructionsData_tag* gapDecisionInstructionsData_t

typedef *gapAdvertisingData_t* gapScanResponseData_t

Scan Response Data structure : a list of several *gapAdStructure_t* structures.

typedef uint8_t gapControllerTestTxType_t

Enumeration for Controller Transmitter Test payload types.

typedef *bleResult_t* gapDisconnectionReason_t

Disconnection reason alias - reasons are contained in HCI error codes.

typedef void (*gapAdvertisingCallback_t)(*gapAdvertisingEvent_t* *pAdvertisingEvent)

Advertising Callback prototype.

typedef void (*gapScanningCallback_t)(*gapScanningEvent_t* *pScanningEvent)

Scanning Callback prototype.

typedef void (*gapConnectionCallback_t)(*deviceId_t* deviceId, *gapConnectionEvent_t* *pConnectionEvent)

Connection Callback prototype.

typedef enum *gapAppearance_tag* gapAppearance_t

Appearance characteristic enumeration, also used in advertising.

typedef struct *gapHostVersion_tag* gapHostVersion_t

bleResult_t Gap_SetDataRelatedAddressChanges(uint8_t advertisingHandle, uint8_t changeReasons)

Command specifies circumstances when the Controller shall refresh any Resolvable Private Address used by the advertising set identified by the Advertising_Handle parameter.

Parameters

- advertisingHandle – **[in]** Identify an advertising set.
- changeReasons – **[in]** Change reasons. Any combination of gAdvDataChange_c and gScanRspDataChange_c. 0U to disable the feature.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range

bleResult_t Gap_RegisterDeviceSecurityRequirements(const *gapDeviceSecurityRequirements_t* *pSecurity)

Registers the device security requirements. This function includes a central security for all services and, optionally, additional stronger security settings for services as required by the profile and/or application.

Remark

pSecurity or any other contained security structure pointers that are NULL are ignored, i.e., defaulted to No Security (Security Mode 1 Level 1, No Authorization, Minimum encryption key size). This function executes synchronously.

Parameters

- pSecurity – **[in]** A pointer to the application-allocated *gapDeviceSecurityRequirements_t* structure.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range

bleResult_t Gap_SetAdvertisingParameters(const *gapAdvertisingParameters_t* *pAdvertisingParameters)

Sets up the Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to *gapAdvertisingParameters_t* structure.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_SetAdvertisingData(const *gapAdvertisingData_t* *pAdvertisingData, const *gapScanResponseData_t* *pScanResponseData)

Sets up the Advertising and Scan Response Data.

Remark

Any of the parameters may be NULL, in which case they are ignored. Therefore, this function can be used to set any of the parameters individually or both at once. The standard advertising packet payload is 37 bytes. Some of the payload may be occupied by the Advertiser Address which takes up 6 bytes and for some advertising PDU types also by the Initiator Address which takes another 6 bytes. This leaves 25-31 bytes to the application to include advertising structures (Length [1Byte], AD Type [1 Byte], AD Data[Length-1 Bytes])

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure or NULL.
- pScanResponseData – **[in]** Pointer to gapScanResponseData_t structure or NULL.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gGapAdvDataTooLong_c – The advertising data length is too big
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_StartAdvertising(*gapAdvertisingCallback_t* advertisingCallback,
gapConnectionCallback_t connectionCallback)

Commands the controller to start advertising.

Remark

The advertisingCallback confirms or denies whether the advertising has started. The connectionCallback is only used if a connection gets established during advertising.

Remark

GAP Peripheral-only API function.

Parameters

- advertisingCallback – **[in]** Callback used by the application to receive advertising events. Can be NULL.
- connectionCallback – **[in]** Callback used by the application to receive connection events. Can be NULL.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidState_c – Advertising is already started or pending to be started
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_StopAdvertising(void)

Commands the controller to stop advertising.

Remark

GAP Peripheral-only API function.

Return values

- gBleSuccess_c –
- gBleInvalidState_c – Advertising is already stopped or pending to be stopped
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_Authorize(*deviceId_t* deviceId, uint16_t handle, *gattDbAccessType_t* access)

Authorizes a peer for an attribute in the database.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer being authorized.
- handle – **[in]** The attribute handle.
- access – **[in]** The type of access granted (gAccessRead_c or gAccessWrite_c).

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory
- gBleInvalidParameter_c – The deviceId parameter is not valid or the access parameter is not gAccessRead_c or gAccessWrite_c
- gBleUnexpectedError_c – Saving to NVM failed (module not initialized or corrupted NVM)
- gBleOverflow_c – the number of handles to be read or written from the NVM is bigger than gcGapMaxAuthorizationHandles_c
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_SaveCccd(*deviceId_t* deviceId, uint16_t handle, *gattCccdFlags_t* cccd)

Save the CCCD value for a specific Client and CCCD handle.

Remark

The GATT Server layer saves the CCCD value automatically when it is written by the Client. This API should only be used to save the CCCD in other situations, e.g., when for some reason the application decides to disable notifications/indications for a specific Client.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD as defined in the GATT Database.
- cccd – **[in]** The bit mask representing the CCCD value to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleUnexpectedError_c – Saving to NVM failed (module not initialized or corrupted NVM)
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory
- gDevDbCccdLimitReached_c – No more room in NVM

bleResult_t Gap_CheckNotificationStatus(*deviceId_t* deviceId, uint16_t handle, bool_t *pOutIsActive)

Retrieves the notification status for a given Client and a given CCCD handle.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD.
- pOutIsActive – **[out]** The address to store the status into.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gDevDbCccdNotFound_c – The handle is not CCCD
- gBleUnavailable_c – The bond data entry for this device is corrupted

bleResult_t Gap_CheckIndicationStatus(*deviceId_t* deviceId, uint16_t handle, bool_t *pOutIsActive)

Retrieves the indication status for a given Client and a given CCCD handle.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD.
- pOutIsActive – **[out]** The address to store the status into.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gDevDbCccdNotFound_c – The handle is not CCCD.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_GetBondedDevicesIdentityInformation(*gapIdentityInformation_t* *aOutIdentityAddresses, uint8_t maxDevices, uint8_t *pOutActualCount)

Retrieves a list of the identity information of bonded devices, if any.

Remark

This API may be useful when creating a filter accept list or a resolving list.

Remark

This function executes synchronously.

Parameters

- aOutIdentityAddresses – **[out]** Array of identities to be filled.
- maxDevices – **[in]** Maximum number of identities to be obtained.
- pOutActualCount – **[out]** The actual number of identities written.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value, is outside the accepted range, or the NVM entry is invalid

bleResult_t Gap_Pair(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Initiates pairing with a peer device.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** The peer to pair with.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – Cannot allocate memory for the Host task

bleResult_t Gap_SendPeripheralSecurityRequest(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Informs the peer Central about the local security requirements.

Remark

The procedure has the same parameters as the pairing request, but, because it is initiated by the Peripheral, it has no pairing effect. It only informs the Central about the requirements.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer to pair with.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

- gBleOutOfMemory_c – Cannot allocate memory for the Host task

bleResult_t Gap_EncryptLink(*deviceId_t* deviceId)

Encrypts the link with a bonded peer.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gGapDeviceNotBonded_c – Trying to execute an API that is only available for bonded devices.

bleResult_t Gap_AcceptPairingRequest(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Accepts the pairing request from a peer.

Remark

This should be called in response to a gPairingRequest_c event.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The peer requesting authentication.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectPairing(*deviceId_t* deviceId, *gapAuthenticationRejectReason_t* reason)

Rejects the peer's authentication request.

Parameters

- deviceId – **[in]** The GAP peer who requested authentication.
- reason – **[in]** Reason why the current device rejects the authentication.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_EnterPasskey(*deviceId_t* deviceId, uint32_t passkey)

Enters the passkey requested by the peer during the pairing process.

Parameters

- deviceId – **[in]** The GAP peer that requested a passkey entry.
- passkey – **[in]** The peer's secret passkey.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ProvideOob(*deviceId_t* deviceId, const uint8_t *aOob)

Provides the Out-Of-Band data for the SMP Pairing process.

Parameters

- deviceId – **[in]** The pairing device.
- aOob – **[in]** Pointer to OOB data (array of gcSmpOobSize_d size).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectPasskeyRequest(*deviceId_t* deviceId)

Rejects the passkey request from a peer.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** The GAP peer that requested a passkey entry.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SendSmpKeys(*deviceId_t* deviceId, const *gapSmpKeys_t* *pKeys)

Sends the SMP keys during the SMP Key Exchange procedure.

Parameters

- deviceId – **[in]** The GAP peer who initiated the procedure.
- pKeys – **[in]** The SMP keys of the local device.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectKeyExchangeRequest(*deviceId_t* deviceId)

Rejects the Key Exchange procedure with a paired peer.

Parameters

- deviceId – **[in]** The GAP peer who requested the Key Exchange procedure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScRegeneratePublicKey(void)

Regenerates the private/public key pair used for LE Secure Connections pairing.

Remark

The application should listen for the gLeScPublicKeyRegenerated_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScValidateNumericValue(*deviceId_t* deviceId, bool_t valid)

Validates the numeric value during the Numeric Comparison LE Secure Connections pairing.

Parameters

- deviceId – **[in]** Device ID of the peer.
- valid – **[in]** TRUE if user has indicated that numeric values are matched, FALSE otherwise.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScGetLocalOobData(void)

Retrieves local OOB data used for LE Secure Connections pairing.

Remark

The application should listen for the gLeScLocalOobData_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScSetPeerOobData(*deviceId_t* deviceId, const *gapLeScOobData_t* *pPeerOobData)

Sets peer OOB data used for LE Secure Connections pairing.

Remark

This function should be called in response to the gConnEvtLeScOobDataRequest_c event.

Parameters

- deviceId – **[in]** Device ID of the peer.
- pPeerOobData – **[in]** OOB data received from the peer.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LeScSendKeypressNotification(*deviceId_t* deviceId, *gapKeypressNotification_t* keypressNotification)

Sends a Keypress Notification to the peer.

Remark

This function shall only be called during the passkey entry process and only if both peers support Keypress Notifications.

Parameters

- deviceId – **[in]** Device ID of the peer.
- keypressNotification – **[in]** Value of the Keypress Notification.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ProvideLongTermKey(*deviceId_t* deviceId, const uint8_t *aLtk, uint8_t ltkSize)

Provides the Long Term Key (LTK) to the controller for encryption setup.

Remark

The application should provide the same LTK used during bonding with the respective peer.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer who requested encryption.
- aLtk – **[in]** The Long Term Key.
- ltkSize – **[in]** The Long Term Key size.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_DenyLongTermKey(*deviceId_t* deviceId)

Rejects an LTK request originating from the controller.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer who requested encryption.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LoadEncryptionInformation(*deviceId_t* deviceId, uint8_t *aOutLtk, uint8_t *pOutLtkSize)

Loads the encryption key for a bonded device.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the peer.
- aOutLtk – **[out]** Array of size gcMaxLtkSize_d to be filled with the LTK.
- pOutLtkSize – **[out]** The LTK size.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_SetLocalPasskey(uint32_t passkey)

Sets the SMP passkey for this device.

Remark

This is the PIN that the peer's user must enter during pairing.

Remark

This function executes synchronously.

Remark

GAP Peripheral-only API function.

Parameters

- passkey – **[in]** The SMP passkey.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

- gSmInvalidDeviceId_c – The device ID is not valid or it has been disconnected in the meantime.
- gSmInvalidCommandCode_c – An invalid internal state machine code was requested.
- gSmPairingAlreadyStarted_c – Pairing process was already started.
- gSmInvalidInternalOperation_c – A memory corruption or invalid operation may have occurred.
- gSmSmpTimeoutOccurred_c – An SMP timeout has occurred for the peer device.
- gSmInvalidCommandParameter_c – One of the parameters of the SM command is not valid.
- gSmCommandNotSupported_c – The Security Manager does not have the required features or version to support this command
- gSmUnexpectedCommand_c – This command is not or cannot be handled in the current context of the SM.
- gSmSmpPacketReceivedAfterTimeoutOccurred_c – A SMP packet has been received from a peer device for which a pairing procedure has timed out.
- gSmUnexpectedPairingTerminationReason_c – The upper layer tried to cancel the pairing procedure with an unexpected pairing failure reason for the current phase of the pairing procedure.

bleResult_t Gap_SetScanMode(*gapScanMode_t* scanMode, *gapAutoConnectParams_t* *pAutoConnectParams, *gapConnectionCallback_t* connCallback)

Sets internal scan filters and actions.

Remark

This function can be called before Gap_StartScanning. If this function is never called, then the default value of gDefaultScan_c is considered and all scanned devices are reported to the application without any additional filtering or action.

Remark

This function executes synchronously.

Remark

GAP Central-only API function.

Parameters

- scanMode – **[in]** The scan mode to be activated. Default is gDefaultScan_c.
- pAutoConnectParams – **[in]** Pointer to the *gapAutoConnectParams_t* structures if scanMode is set to gAutoConnect_c. The memory used must be persistent and should not change during the next scan periods or until another Gap_SetScanMode is called.
- connCallback – **[in]** Auto-Connect callback. Must be set if scanMode is set to gAutoConnect_c.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetDecisionInstructions(uint8_t numTest, const *gapDecisionInstructionsData_t* *pDecisionInstructions)

Sets up the Decision Instructions.

Remark

GAP Central-only API function.

Remark

Application should wait for gDecisionInstructionsSetupComplete_c generic event or for gInternalError_c generic event with errorSource = gSetDecisionInstructions_c.

Parameters

- numTest – **[in]** The number of tests in the decision instructions.
- pDecisionInstructions – **[in]** Pointer to an array of gapDecisionInstructionsData_t structures. containing decision instruction for each test.

Return values

- gBleFeatureNotSupported_c – in case gLeExtendedAdv_c or gLeDecisionBasedAdvertisingFiltering_c are not supported.
- gBleInvalidParameter_c – in case at least 1 parameter is incorrect.
- gBleOutOfMemory_c – in case the memory allocation for the app to host message fails.
- gBleSuccess_c – otherwise.

bleResult_t Gap_InitMonitoringAdvertisers(void)

Initialize function table for Monitored Advertisers functionality.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Feature is not supported.

bleResult_t Gap_AddDeviceToMonAdvList(*bleAddressType_t* addressType, *bleDeviceAddress_t* address, int8_t rssiLowThreshold, int8_t rssiHighThreshold, uint8_t timeout)

This function adds a device to the Monitored Advertisers List.

Parameters

- addressType – **[in]** Address type of the device.
- address – **[in]** Address of the device.
- rssiLowThreshold – **[in]** RSSI low threshold value.

- rssiHighThreshold – **[in]** RSSI high threshold value.
- timeout – **[in]** Timeout value.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveDeviceFromMonAdvList(*bleAddressType_t* addressType, *bleDeviceAddress_t* address)

This function removes a device from the Monitored Advertisers List.

Parameters

- addressType – **[in]** Address type of the device.
- address – **[in]** Address of the device.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ClearMonAdvList(void)

This function clears the Monitored Advertisers List.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableMonAdv(bool_t enable)

This function enables or disables monitoring of advertisers.

Parameters

- enable – **[in]** Enable or disable monitoring.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadMonAdvListSize(void)

This function reads the size of the Monitored Advertisers List.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StartScanning(const *gapScanningParameters_t* *pScanningParameters, *gapScanningCallback_t* scanningCallback, *gapFilterDuplicates_t* enableFilterDuplicates, uint16_t duration, uint16_t period)

Optionally sets the scanning parameters and begins scanning.

Remark

Use this API to both set the scanning parameters and start scanning. If pScanningParameters is NULL, scanning is started with the existing settings.

Remark

GAP Central-only API function.

Parameters

- pScanningParameters – **[in]** The scanning parameters; may be NULL.
- scanningCallback – **[in]** The scanning callback.
- enableFilterDuplicates – **[in]** Enable or disable duplicate advertising report filtering
- duration – **[in]** Scan duration expressed in units of 10 ms. Set 0 for continuous scan until explicitly disabled. Used only for BLE5.0, otherwise ignored.
- period – **[in]** Time interval expressed in units of 1.28 seconds from when the Controller started its last Scan_Duration until it begins the subsequent Scan_Duration. Set 0 to disable periodic scanning. Used only for BLE5.0, otherwise ignored.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The requested API cannot be called in the current state, scanning was already started.

bleResult_t Gap_StopScanning(void)

Commands the controller to stop scanning.

Remark

GAP Central-only API function.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The requested API cannot be called in the current state, scanning is in the process of stopping.

bleResult_t Gap_Connect(const *gapConnectionRequestParameters_t* *pParameters, *gapConnectionCallback_t* connCallback)

Connects to a scanned device.

Remark

GAP Central-only API function.

Parameters

- pParameters – **[in]** Create Connection command parameters.
- connCallback – **[in]** Callback used to receive connection events.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ConnectFromPawr(const *gapConnectionFromPawrParameters_t* *pParameters, *gapConnectionCallback_t* connCallback)

Connects to a device which is doing periodic advertising with responses.

Remark

GAP Central-only API function.

Parameters

- pParameters – **[in]** Create Connection command parameters.
- connCallback – **[in]** Callback used to receive connection events.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_Disconnect(*deviceId_t* deviceId)

Initiates disconnection from a connected peer device.

Parameters

- deviceId – **[in]** The connected peer to disconnect from.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SaveCustomPeerInformation(*deviceId_t* deviceId, const uint8_t *pInfo, uint16_t offset, uint16_t infoSize)

Saves custom peer information in raw data format.

Remark

This function can be called by the application to save custom information about the peer device, e.g., Service Discovery data (to avoid doing it again on reconnection).

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pInfo – **[in]** Pointer to the beginning of the data.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_LoadCustomPeerInformation(*deviceId_t* deviceId, uint8_t *pOutInfo, uint16_t offset, uint16_t infoSize)

Loads the custom peer information in raw data format.

Remark

This function can be called by the application to load custom information about the peer device, e.g., Service Discovery data (to avoid doing it again on reconnection).

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pOutInfo – **[out]** Pointer to the beginning of the allocated memory.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_LoadCustomBondedDeviceInformation(uint8_t nvmIndex, uint8_t *pOutInfo, uint16_t offset, uint16_t infoSize)

Loads the custom bonded device information in raw data format. Unlike Gap_LoadCustomPeerInformation, it does not require an active connection.

Remark

This function can be called by the application to load custom information about the bonded device.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pOutInfo – **[out]** Pointer to the beginning of the allocated memory.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_CheckIfBonded(*deviceId_t* deviceId, bool_t *pOutIsBonded, uint8_t *pOutNvmIndex)

Returns whether or not a connected peer device is bonded and the NVM index.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pOutIsBonded – **[out]** Boolean to be filled with the bonded flag.
- pOutNvmIndex – **[out]** If bonded, to be filled optionally with the NVM index.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_CheckIfConnected(*bleAddressType_t* addressType, const *bleDeviceAddress_t* aAddress, bool_t addrResolved, bool_t *pOutIsConnected)

Checks if a connection exists between the local device and another device that has the provided address information. This function executes synchronously.

Parameters

- addressType – **[in]** Peer address type
- aAddress – **[in]** Peer address
- addrResolved – **[in]** Set to TRUE if the address contained in the addressType and aAddress fields is the identity address of a resolved RPA.
- pOutIsConnected – **[out]** TRUE if the device is found

Return values

- bleResult_t –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_CheckNvmIndex(uint8_t nvmIndex, bool_t *pOutIsFree)

Returns whether or not the given NVM index is free.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** NVM index.
- pOutIsFree – **[out]** TRUE if free, FALSE if occupied.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ReadFilterAcceptListSize(void)

Retrieves the size of the Filter Accept List.

Remark

Response is received in the gFilterAcceptListSizeRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ClearFilterAcceptList(void)

Removes all addresses from the Filter Accept List, if any.

Remark

Confirmation is received in the gFilterAcceptListCleared_c generic event.

Return values

gBleSuccess_c –

bleResult_t Gap_AddDeviceToFilterAcceptList(*bleAddressType_t* addressType, const *bleDeviceAddress_t* address)

Adds a device address to the Filter Accept List.

Parameters

- address – **[in]** The address of the filter accept listed device.
- addressType – **[in]** The device address type (public or random).

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveDeviceFromFilterAcceptList(*bleAddressType_t* addressType, const *bleDeviceAddress_t* address)

Removes a device address from the Filter Accept List.

Parameters

- address – **[in]** The address of the filter accept listed device.
- addressType – **[in]** The device address type (public or random).

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadPublicDeviceAddress(void)

Reads the device's public address from the controller.

Remark

The application should listen for the gPublicAddressRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_CreateRandomDeviceAddress(const uint8_t *aIrk, const uint8_t *aRandomPart)

Requests the controller to create a random address.

Remark

The application should listen for the gRandomAddressReady_c generic event. Note that this does not set the random address in the Controller. To set the random address, call *Gap_SetRandomAddress()* with the generated address contained in the event data.

Parameters

- aIrk – **[in]** The Identity Resolving Key to be used for a private resolvable address or NULL for a private non-resolvable address (fully random).
- aRandomPart – **[in]** If aIrk is not NULL, this is a 3-byte array containing the Random Part of a Private Resolvable Address, in LSB to MSB order; the most significant two bits of the most significant byte (aRandomPart[3] & 0xC0) are ignored. This may be NULL, in which case the Random Part is randomly generated internally.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SaveDeviceName(*deviceId_t* deviceId, const uchar_t *pName, uint8_t cNameSize)

Store the name of a bonded device.

Remark

This function copies cNameSize characters from the pName array and adds the NULL character to terminate the string.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID for the active peer which name is saved.
- pName – **[in]** Array of characters holding the name.
- cNameSize – **[in]** Number of characters to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_GetBondedDevicesCount(uint8_t *pOutBondedDevicesCount)

Retrieves the number of bonded devices.

Remark

This function executes synchronously.

Parameters

- pOutBondedDevicesCount – **[out]** Pointer to integer to be written.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetBondedDeviceName(uint8_t nvmIndex, uchar_t *pOutName, uint8_t maxNameSize)

Retrieves the name of a bonded device.

Remark

nvmIndex is an integer ranging from 0 to N-1, where N is the number of bonded devices and can be obtained by calling Gap_GetBondedDevicesCount(&N).

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pOutName – **[out]** Destination array to copy the name into.
- maxNameSize – **[in]** Maximum number of characters to be copied, including the terminating NULL character.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_SetBondedDeviceName(uint8_t nvmIndex, const uchar_t *pName, uint8_t cNameSize)

Store the name of a bonded device.

Remark

This function copies cNameSize characters from the pName array and adds the NULL character to terminate the string.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pName – **[in]** Array of characters holding the name.
- cNameSize – **[in]** Number of characters to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_RemoveBond(uint8_t nvmIndex)

Removes the bond with a device.

Remark

This API requires that there are no active connections at call time. The *Gap_CheckIfBonded()* API can be called to obtain the nvmIndex of a bonded peer.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The bond is not active.

bleResult_t Gap_RemoveAllBonds(void)

Removes all bonds with other devices.

Remark

This API requires that there are no active connections at call time.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidState_c – At least one bond is still active.

bleResult_t Gap_ReadRadioPowerLevel(*gapRadioPowerLevelReadType_t* txReadType, *deviceId_t* deviceId)

Reads the power level of the controller's radio. The response is contained in the gConnEvtTxPowerLevelRead_c connection event when reading connection TX power level, the

gAdvTxPowerLevelRead_c generic event when reading the advertising TX power level, or the gConnEvtRssiRead_c connection event when reading the RSSI.

Parameters

- txReadType – **[in]** Advertising or connection Tx power
- deviceId – **[in]** Peer identifier (for connections only, otherwise ignored)

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetTxPowerLevel(uint8_t powerLevel, *bleTransmitPowerChannelType_t* channelType)

Sets the Tx power level on the controller's radio.

Remark

The application should listen for the gTxPowerLevelSetComplete_c generic event.

Remark

This function executes synchronously.

Remark

For QN908x platform this command is not supported by the controller.

Remark

Instead, use RF_SetTxPowerLevel API to set the desired TX power level.

Parameters

- powerLevel – **[in]** Power level as specified in the controller interface.
- channelType – **[in]** The advertising or connection channel type.

Return values

gBleSuccess_c –

bleResult_t Gap_VerifyPrivateResolvableAddress(uint8_t nvmIndex, const *bleDeviceAddress_t* aAddress)

Verifies a Private Resolvable Address with a bonded device's IRK.

Remark

nvmIndex is an integer ranging from 0 to N-1, where N is the number of bonded devices and can be obtained by calling Gap_GetBondedDevicesCount(&N); the application should listen to the gPrivateResolvableAddressVerified_c event.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area whose IRK must be checked.
- aAddress – **[in]** The Private Resolvable Address to be verified.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetRandomAddress(const *bleDeviceAddress_t* aAddress)

Sets a random address into the Controller.

Remark

The application should listen for the gRandomAddressSet_c generic event.

Parameters

- aAddress – **[in]** The Private Resolvable, Private Non-Resolvable, or Static Random Address.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadControllerLocalRPA(const *bleIdentityAddress_t* *pIdAddress)

Reads the device's Local Private Address for a specific peer device from the controller.

Remark

The application should listen for the gControllerLocalRPARead_c generic event.

Parameters

- pIdAddress – **[in]** pointer to the peer identity address the local private address should be retrieved for.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The Controller privacy is not enabled.

bleResult_t Gap_SetDefaultPairingParameters(const *gapPairingParameters_t* *pPairingParameters)

Sets the default pairing parameters to be used by automatic pairing procedures.

Remark

When these parameters are set, the Security Manager automatically responds to a Pairing Request or a Peripheral Security Request using these parameters. If NULL is provided, it returns to the default state where all security requests are sent to the application.

Remark

This function executes synchronously.

Parameters

- pPairingParameters – **[in]** Pairing parameters as required by the SMP or NULL.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_UpdateConnectionParameters(*deviceId_t* deviceId, uint16_t intervalMin, uint16_t intervalMax, uint16_t peripheralLatency, uint16_t timeoutMultiplier, uint16_t minCeLength, uint16_t maxCeLength)

Request a set of new connection parameters.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- intervalMin – **[in]** The minimum value for the connection event interval
- intervalMax – **[in]** The maximum value for the connection event interval
- peripheralLatency – **[in]** The peripheral latency parameter
- timeoutMultiplier – **[in]** The connection timeout parameter
- minCeLength – **[in]** The minimum value for the connection event length
- maxCeLength – **[in]** The maximum value for the connection event length

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

Pre

A connection must be in place

bleResult_t Gap_EnableUpdateConnectionParameters(*deviceId_t* deviceId, bool_t enable)

Update the connection parameters.

Remark

The LE central Host may accept the requested parameters or reject the request

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- enable – **[in]** Allow/disallow the parameters update

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – The device id is not valid.

Pre

A connection must be in place

bleResult_t Gap_UpdateLeDataLength(*deviceId_t* deviceId, uint16_t txOctets, uint16_t txTime)

Update the Tx Data Length.

Remark

The response is contained in the gConnEvtLeDataLengthChanged_c connection event.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- txOctets – **[in]** Maximum transmission number of payload octets
- txTime – **[in]** Maximum transmission time

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – The device id is not valid.

Pre

A connection must be in place

bleResult_t Gap_EnableHostPrivacy(bool_t enable, const uint8_t *aIrk)

Enables or disables Host Privacy (automatic regeneration of a Private Address).

Remark

The application should listen for the gHostPrivacyStateChanged_c generic event.

Parameters

- enable – **[in]** TRUE to enable, FALSE to disable.
- aIrk – **[in]** Local IRK to be used for Resolvable Private Address generation or NULL for Non-Resolvable Private Address generation. Ignored if enable is FALSE.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableControllerPrivacy(bool_t enable, const uint8_t *aOwnIrk, uint8_t peerIdCount, const *gapIdentityInformation_t* *aPeerIdentities)

Enables or disables Controller Privacy (Enhanced Privacy feature).

Remark

The application should listen for the gControllerPrivacyStateChanged_c generic event.

Parameters

- enable – **[in]** TRUE to enable, FALSE to disable.
- aOwnIrk – **[in]** Local IRK. Ignored if enable is FALSE, otherwise shall not be NULL.
- peerIdCount – **[in]** Size of aPeerIdentities array. Shall not be zero or greater than gcGapControllerResolvingListSize_c. Ignored if enable is FALSE.
- aPeerIdentities – **[in]** Array of peer identity addresses and IRKs. Ignored if enable is FALSE, otherwise shall not be NULL.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleFeatureNotSupported_c – Cannot use Device Privacy mode if controller does not support LE Set Privacy Mode Command.

bleResult_t Gap_SetPrivacyMode(uint8_t nvmIndex, *blePrivacyMode_t* privacyMode)

Sets the privacy mode to an existing bond.

Remark

The change has no effect (other than the change in NVM) unless controller privacy is enabled for the bonded identities.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- privacyMode – **[in]** Controller privacy mode: Network or Device

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Cannot use Device Privacy mode if controller does not support LE Set Privacy Mode Command.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleOutOfMemory_c – There is no more room to allocate memory for a bonding entry.

bleResult_t Gap_ControllerTest(*gapControllerTestCmd_t* testCmd, uint8_t radioChannel, uint8_t txDataLength, *gapControllerTestTxType_t* txPayloadType)

Commands a Controller Test procedure.

Remark

The application should listen for the gControllerTestEvent_c generic event.

Remark

This API function is available only in the full-featured host library.

Parameters

- testCmd – Command type - “start TX test”, “start RX test” or “end test”.
- radioChannel – Radio channel index. Valid range: 0-39. Frequency will be F[MHz] = 2402 + 2 * index. Effective range: 2402-2480 MHz. Ignored if command is “end test”.
- txDataLength – Size of packet payload for TX tests. Ignored if command is “start RX test” or “end test”.
- txPayloadType – Type of packet payload for TX tests. Ignored if command is “start RX test” or “end test”.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ModifySleepClockAccuracy(*gapSleepClockAccuracy_t* action)

Requests the Controller to change its sleep clock accuracy for testing purposes.

Remark

The application should listen for the generic event.

Parameters

- action – Specifies whether the sleep clock should be changed to one that is more accurate or one that is less accurate.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LeReadPhy(*deviceId_t* deviceId)

Read the Tx and Rx Phy on the connection with a device.

Remark

The application should listen for the gLePhyEvent_c generic event. This API is available only in the Bluetooth 5.0 Host Stack.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetPhy(bool_t defaultMode, *deviceId_t* deviceId, uint8_t allPhys, uint8_t txPhys, uint8_t rxPhys, uint16_t phyOptions)

Set the Tx and Rx Phy preferences on the connection with a device or all subsequent connections.

Remark

The application should listen for the gLePhyEvent_c generic event. This API is available only in the Bluetooth 5.0 Host Stack.

Parameters

- defaultMode – **[in]** Use the provided values for all subsequent connections
- deviceId – **[in]** Device ID of the peer Ignored if defaultMode is TRUE.
- allPhys – **[in]** Host preferences on Tx and Rx Phy, as defined by gapLeAllPhyFlags_t
- txPhys – **[in]** Host preferences on Tx Phy, as defined by gapLePhyFlags_t, ignored for gLeTxPhyNoPreference_c
- rxPhys – **[in]** Host preferences on Rx Phy, as defined by gapLePhyFlags_t, ignored for gLeRxPhyNoPreference_c

- phyOptions – **[in]** Host preferences on Coded Phy, as defined by gapLePhyOptionsFlags_t Ignored if defaultMode is TRUE.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_ControllerEnhancedNotification(uint16_t eventType, *deviceId_t* deviceId)

Configures enhanced notifications on advertising, scanning and connection events on the controller.

Remark

The application should listen for the gControllerNotificationEvent_c generic event.

Remark

This function executes synchronously.

Parameters

- eventType – **[in]** Event type selection as specified by bleNotificationEventType_t.
- deviceId – **[in]** Device ID of the peer, used only for connection events.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_BleAdvIndexChange(*bleAdvIndexType_t* advIndexType, uint8_t aUserDefinedChannels[3])

This function configures the advertising index type.

Parameters

- advIndexType – **[in]** Advertising index type
- aUserDefinedChannels – **[in]** User defined channels array

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LoadKeys(uint8_t nvmIndex, *gapSmpKeys_t* *pOutKeys, *gapSmpKeyFlags_t* *pOutKeyFlags, bool_t *pOutLeSc, bool_t *pOutAuth)

Retrieves the keys from an existing bond with a device.

Remark

This API requires that the aAddress in the pOutKeys shall not be NULL.

Remark

The application will check pOutKeyFlags to see which information is valid in pOutKeys.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- pOutKeys – **[out]** Pointer to fill the keys distributed during pairing.
- pOutKeyFlags – **[out]** Pointer to indicate which keys were distributed during pairing.
- pOutLeSc – **[out]** Pointer to mark if LE Secure Connections was used during pairing.
- pOutAuth – **[out]** Pointer to mark if the device was authenticated for MITM during pairing.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SaveKeys(uint8_t nvmIndex, const *gapSmpKeys_t* *pKeys, bool_t leSc, bool_t auth)

Saves the keys to a new or existing bond based on OOB information.

Remark

This API requires that the aAddress in the pKeys shall not be NULL.

Remark

If any of the keys are passed as NULL, they will not be saved.

Remark

The application listen for gBondCreatedEvent_c to confirm the bond was created.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- pKeys – **[in]** Pointer to the keys distributed during pairing.
- leSc – **[in]** Indicates if LE Secure Connections was used during pairing.
- auth – **[in]** Indicates if the device was authenticated for MITM during pairing.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetChannelMap(const *bleChannelMap_t* channelMap)

Set the channel map in the Controller and trigger an LL channel map update. Specifies the channel map for data, secondary advertising and periodic physical channels.

Remark

The application should listen for the gChannelMapSet_c generic event.

Remark

This function executes synchronously.

Remark

API supported on GAP Central devices or Broadcaster with Extended Advertising support.

Parameters

- channelMap – **[in]** Array with the channels using 0 for bad channels and 1 for unknown.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

bleResult_t Gap_ReadChannelMap(*deviceId_t* deviceId)

Reads the channel map of a connection.

Remark

The application should listen for the gConnEvtChannelMapRead_c connection event.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetExtAdvertisingParameters(*gapExtAdvertisingParameters_t* *pAdvertisingParameters)

Sets up the Extended Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to gapExtAdvertisingParameters_t structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gGapAnotherProcedureInProgress_c – Another set extended advertisment parameters command is in progress.

bleResult_t Gap_SetExtAdvertisingData(uint8_t handle, *gapAdvertisingData_t* *pAdvertisingData, *gapScanResponseData_t* *pScanResponseData)

Sets up the Extended Advertising and Extended Scan Response Data.

Remark

Any of the parameters may be NULL, in which case they are ignored. Therefore, this function can be used to set any of the parameters individually or both at once.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set
- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure or NULL.
- pScanResponseData – **[in]** Pointer to gapScanResponseData_t structure or NULL.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not configured previously using Gap_SetExtAdvertisingParameters.
- gGapAdvDataTooLong_c – The advertising data is too long.

bleResult_t Gap_SetExtAdvertisingDecisionData(uint8_t handle, const *gapAdvertisingDecisionData_t* *pAdvertisingDecisionData)

Sets up the Extended Advertising Decision Data.

Remark

pDecisionData or pKey in the *gapAdvertisingDecisionData_t* structure may be NULL but not both.

Remark

GAP Peripheral-only API function.

Remark

Application should wait for gExtAdvertisingDecisionDataSetupComplete_c or generic event or for gInternalError_c generic event with errorSource = gSetExtAdvDecisionData_c.

Parameters

- handle – **[in]** The ID of the advertising set
- pAdvertisingDecisionData – **[in]** Pointer to *gapAdvertisingDecisionData_t* structure.

Return values

- gBleFeatureNotSupported_c – in case gLeExtendedAdv_c or gLeDecisionBasedAdvertisingFiltering_c are not supported.

- gBleInvalidParameter_c – in case at least 1 parameter is incorrect.
- gBleOutOfMemory_c – in case the memory allocation for the app to host message fails.
- gBleSuccess_c – otherwise.

bleResult_t Gap_StartExtAdvertising(*gapAdvertisingCallback_t* advertisingCallback, *gapConnectionCallback_t* connectionCallback, uint8_t handle, uint16_t duration, uint8_t maxExtAdvEvents)

Commands the controller to start the extended advertising.

Remark

The advertisingCallback confirms or denies whether the advertising has started. The connectionCallback is only used if a connection gets established during advertising.

Remark

GAP Peripheral-only API function.

Parameters

- advertisingCallback – **[in]** Callback used by the application to receive advertising events. Can be NULL.
- connectionCallback – **[in]** Callback used by the application to receive connection events. Can be NULL.
- handle – **[in]** The ID of the advertising set
- duration – **[in]** The duration of the advertising
- maxExtAdvEvents – **[in]** The maximum number of advertising events

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The advertising is already started or pending to be started.

bleResult_t Gap_StopExtAdvertising(uint8_t handle)

Commands the controller to stop extended advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set is not started.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveAdvSet(uint8_t handle)

Commands the controller to remove the specified advertising set and all its data.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvParameters(*gapPeriodicAdvParameters_t* *pAdvertisingParameters)

Sets up the Periodic Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to gapPeriodicAdvParameters_t structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvSubeventData(uint8_t advHandle, const *gapPeriodicAdvertisingSubeventData_t* *pData)

Sets up the Periodic Advertising Subevent Data.

Remark

GAP Peripheral-only API function.

Parameters

- advHandle – **[in]** Advertising handle.
- gapPeriodicAdvertisingSubeventData_t – Pointer to *gapPeriodicAdvertisingSubeventData_t* structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvResponseData(uint16_t syncHandle, const *gapPeriodicAdvertisingResponseData_t* *pData)

Sets up the Periodic Advertising Response Data.

Parameters

- syncHandle – **[in]** Sync handle.
- gapPeriodicAdvertisingResponseData_t – Pointer to *gapPeriodicAdvertisingResponseData_t* structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicSyncSubevent(uint16_t syncHandle, const *gapPeriodicSyncSubeventParameters_t* *pParams)

Instructs the controller to sync with a subset of the subevents within a PAWR train identified by syncHandle.

Parameters

- syncHandle – **[in]** Sync handle.
- gapPeriodicSyncSubeventParameters_t – Pointer to *gapPeriodicSyncSubeventParameters_t* structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvertisingData(uint8_t handle, *gapAdvertisingData_t* *pAdvertisingData, bool_t bUpdateDID)

Sets up the Periodic Advertising Data.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set
- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure.
- bUpdateDid – **[in]** If TRUE, use operation mode gHciExtDataUnchanged_c.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gGapAdvDataTooLong_c – The advertising data is too long.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StartPeriodicAdvertising(uint8_t handle, bool_t bIncludeADI)

Commands the controller to start periodic advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set
- bIncludeADI – **[in]** If TRUE, include ADI field in AUX_SYNC_IND PDUs

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StopPeriodicAdvertising(uint8_t handle)

Commands the controller to stop periodic advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_UpdatePeriodicAdvList(*gapPeriodicAdvListOperation_t* operation, *bleAddressType_t* addrType, uint8_t *pAddr, uint8_t SID)

Manage the periodic advertising list.

Remark

GAP Central-only API function.

Parameters

- operation – **[in]** The list operation: add/remove a device, or clear all.
- addrType – **[in]** The address type of the periodic advertiser.
- pAddr – **[in]** Pointer to the advertiser's address.
- SID – **[in]** The ID of the advertising set.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvCreateSync(*gapPeriodicAdvSyncReq_t* *pReq)

Start tracking periodic advertisements. Scanning is required to be ON for this request to be processed, so the scanning callback will receive the periodic advertising events.

Remark

GAP Central-only API function.

Parameters

- pReq – **[in]** Pointer to the Sync Request parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvTerminateSync(uint16_t syncHandle)

Stop tracking periodic advertisements.

Remark

GAP Central-only API function.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvReceiveEnable(uint16_t syncHandle, bool_t enableDuplicateFiltering)

Enable Periodic Advertisement Sync Transfer Controller feature.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser
- enableDuplicateFiltering – **[in]** Used to enable or disable duplicate filtering

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvReceiveDisable(uint16_t syncHandle)

Disable Periodic Advertisement Sync Transfer Controller feature.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvSyncTransfer(*gapPeriodicAdvSyncTransfer_t* *pParam)

Instruct the Controller to send synchronization information about the periodic advertising train identified by the sync handle to a connected device.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvSetInfoTransfer(*gapPeriodicAdvSetInfoTransfer_t* *pParam)

Instruct the Controller to send synchronization information about the periodic advertising to a connected device.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvSyncTransferParams(*gapSetPeriodicAdvSyncTransferParams_t* *pParam)

Specify how the Controller will process periodic advertising synchronization information received from the device identified by the device id.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetDefaultPeriodicAdvSyncTransferParams(*gapSetPeriodicAdvSyncTransferParams_t* *pParam)

Specify the initial value for the mode, skip, timeout, and Constant Tone Extension type (set by the Gap_SetPeriodicAdvSyncTransferParams command) to be used for all subsequent connections over the LE transport.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ResumeLeScStateMachine(computeDhKeyParam_t *pData)

Resume the pairing process. At this point the ECDH key must be computed. This function should be called only for secured LE connections. In any other cases the user should make his own code for handling the case when the ECDH computation is completed.

Parameters

- pData – **[in]** Pointer to the data used to resume the host state machine. The data is allocated by the stack when it requests an ECDH multiplication. It is also freed by the stack at the end of the multiplication.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetConnectionlessCteTransmitParameters(*gapConnectionlessCteTransmitParams_t* *pTransmitParams)

Set Connectionless CTE Transmit Parameters for an advertising set.

Parameters

- pTransmitParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionlessCteTransmit(uint8_t handle, *bleCteTransmitEnable_t* enable)

Enable or disable Connectionless CTE Transmit for an advertising set.

Parameters

- handle – **[in]** Advertising set handle.
- enable – **[in]** Enable or disable CTE Transmit.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionlessIqSampling(uint16_t syncHandle, *gapConnectionlessIqSamplingParams_t* *pIqSamplingParams)

Enable or disable Connectionless IQ sampling for an advertising train.

Parameters

- syncHandle – **[in]** Used to identify advertising train.
- pIqSamplingParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetConnectionCteReceiveParameters(*deviceId_t* deviceId, *gapConnectionCteReceiveParams_t* *pReceiveParams)

Set CTE Receive Parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pReceiveParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetConnectionCteTransmitParameters(*deviceId_t* deviceId, *gapConnectionCteTransmitParams_t* *pTransmitParams)

Set CTE Transmit Parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pTransmitParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionCteRequest(*deviceId_t* deviceId, *gapConnectionCteReqEnableParams_t* *pCteReqEnableParams)

Enable or disable CTE Request procedure for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pCteReqEnableParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionCteResponse(*deviceId_t* deviceId, *bleCteRspEnable_t* enable)

Enable or disable sending CTE Responses for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- enable – **[in]** Enable or disable sending CTE Responses.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadAntennaInformation(void)

Read Antenna Information.

Remark

Antenna information contained in gAntennaInformationRead_c generic event.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnhancedReadTransmitPowerLevel(*deviceId_t* deviceId, *blePowerControlPhyType_t* phy)

Read local current and maximum tx power levels for a certain connection and PHY.

Parameters

- deviceId – **[in]** Peer device ID.
- phy – **[in]** PHY.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadRemoteTransmitPowerLevel(*deviceId_t* deviceId, *blePowerControlPhyType_t* phy)

Read remote tx power for a certain connection and PHY.

Parameters

- deviceId – **[in]** Peer device ID.
- phy – **[in]** PHY.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPathLossReportingParameters(*deviceId_t* deviceId, *gapPathLossReportingParams_t* *pPathLossReportingParams)

Set path loss threshold reporting parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pPathLossReportingParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnablePathLossReporting(*deviceId_t* deviceId, *blePathLossReportingEnable_t* enable)

Enable or disable path loss threshold reporting for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- enable – **[in]** Enable or disable path loss threshold reporting.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableTransmitPowerReporting(*deviceId_t* deviceId, *bleTxPowerReportingEnable_t* localEnable, *bleTxPowerReportingEnable_t* remoteEnable)

Enable or disable tx power reporting for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- localEnable – **[in]** Enable or disable local tx power reports.
- removeEnable – **[in]** Enable or disable remote tx power reports.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GenerateDhKeyV2(ecdhPublicKey_t *pRemoteP256PublicKey, *gapPrivateKeyType_t* keyType)

Initiates generation of a Diffie-Hellman key in the Controller for use over the LE transport. Version 2 of LE Generate DHKey command.

Remark

Generated key is contained in gHciLeGenerateDhKeyCompleteEvent_c event.

Parameters

- pRemoteP256PublicKey – **[in]** Pointer to the remote P-256 public key used as input for DH key generation.
- keyType – **[in]** The private key used of DH key generation.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattConnectionRequest(*deviceId_t* deviceId, uint16_t mtu, uint8_t cBearers, uint16_t initialCredits, bool_t autoCreditsMgmt)

Open up to 5 Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- mtu – **[in]** MTU
- cBearers – **[in]** Number of bearers
- initialCredits – **[in]** Initial credits
- autoCreditsMgmt – **[in]** EATT will automatically send credits in chunks of initialCredits if TRUE.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattConnectionAccept(*deviceId_t* deviceId, bool_t accept, uint16_t localMtu, uint16_t initialCredits, bool_t autoCreditsMgmt)

Accept or reject a received EATT Connection Request.

Parameters

- deviceId – **[in]** Peer device id
- accept – **[in]** TRUE - accept received EattConnectionRequest FALSE - reject incoming EattConnectionRequest
- localMtu – **[in]** Local MTU
- Initial (initialCredits,) – **[in]** credits
- autoCreditsMgmt – **[in]** EATT will automatically send credits in chunks of initialCredits if TRUE.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattReconfigureRequest(*deviceId_t* deviceId, uint16_t mtu, uint16_t mps, uint8_t cBearers, *bearerId_t* *pBearers)

Reconfigure the MTU of up to 5 Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- mtu – **[in]** New MTU value to be configured.
- mps – **[in]** New MPS value to be configured. Set to 0 to use current maximum mps value of the channels being reconfigured
- cBearers – **[in]** Number of bearers
- *pBearers – **[in]** Initial credits

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattSendCredits(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t credits)

Send L2cap credits for Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- bearerId – **[in]** Enhanced ATT bearer id.
- credits – **[in]** Number of credits.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattDisconnect(*deviceId_t* deviceId, *bearerId_t* bearerId)

Disconnect the specified bearer.

Parameters

- deviceId – **[in]** Peer device id
- bearerId – **[in]** Enhanced ATT bearer id to be disconnected.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GetDeviceIdFromConnHandle(uint16_t connHandle, *deviceId_t* *pDeviceId)

Returns the deviceId associated with the received connection handle.

Parameters

- connHandle – **[in]** Connection identifier
- pDeviceId – **[out]** Corresponding device id

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetConnectionHandleFromDeviceId(*deviceId_t* deviceId, uint16_t *pConnHandle)

Returns the deviceId associated with the received connection handle.

Parameters

- deviceId – **[in]** Connection identifier
- pConnHandle – **[out]** Corresponding connection handle

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetHostVersion(*gapHostVersion_t* *pOutHostVersion)

Retrieves Host Version information.

Parameters

- pOutHostVersion – **[out]** Pointer to the memory location where the Host Version information should be stored.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ReadRemoteVersionInformation(*deviceId_t* deviceId)

Reads the version information of a peer device.

Parameters

- deviceId – **[in]** Peer device identifier.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GetConnParamsMonitoring(*deviceId_t* deviceId, uint8_t mode)

Get the Connection parameters for the given deviceId in the given mode.

Parameters

- deviceId – **[in]** Peer device Id.
- mode – **[in]** Connection parameters event report mode: bit 0: update the local sequence number for central / peripheral differentiation and send the LE_Handover_Connection_Parameters_Update_Event after remote sequence number change. bit 1: Send LE_Handover_Connection_Parameters_Update_Event when the connection parameters are updated following an LL procedure

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeChannelOverride(*bleChannelOverrideMode_t* mode, uint8_t channelListLength, uint8_t *pChannelList)

This function sets the channels to be used for advertising, scanning or initiation.

If the mode is 0x00 (advertising), the specified channels are used by the following advertising commands. The channels used by the ongoing advertising will not change until the advertising is disabled, even if a new channel override command with mode 0x00 is received. This allows different advertising channels for different advertising sets. The mode 0x00 (advertising) can be issued when an advertising is ongoing. The setting will be used by the next advertising enable command.

Remark

In the channel list, the MSB of each byte indicates if the channel is BLE channel index or generic channel index:

- BLE channel index: 0 to 39
- Generic channel index: 0x80+0 to 0x80+127

If the mode is 0x01 or 0x02 (scan or initiation), the specified channels are used by the following scan and initiation. The command with mode 0x01 or 0x02 should be issued when the scan or initiation is not ongoing.

When a generic channel is used, the whitening is initialized by the lower 6 bits of the generic channel index. When a BLE channel is used, the whitening is initialized by the BLE channel index. So the BLE channel 37/38/39 are not strictly the same as the generic channels 42/66/120: they operate on the same frequency but their channel whitening is different.

Parameters

- mode – **[in]** Override the advertising, scanning or initiation channels.
- channelListLength – **[in]** Length of channel list, must be between 1 and 6.
- pChannelList – **[in]** List of channels (see remarks for more details).

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetSchedulerPriority(uint16_t priorityHandle)

This function sets the priority for one connection in case of several connections by calling the corresponding HCI command.

Parameters

- priorityHandle – **[in]** Parameter that sets the priority for connections

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LePeriodicAdvUpdateSync(uint16_t syncHandle, uint16_t skip, uint16_t syncTimeout)

This function updates the synchronization parameters for periodic advertising.

Parameters

- syncHandle – **[in]** Handle identifying the periodic advertising sync.
- skip – **[in]** Number of periodic advertising packets to skip.
- syncTimeout – **[in]** Synchronization timeout value.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetHostFeature(uint8_t bitNumber, bool_t enable)

This function is used by the Host to set or clear a bit controlled by the Host in the Link Layer FeatureSet.

Parameters

- bitNumber – **[in]** Bit position in the FeatureSet
- enable – **[in]** Host feature is enabled/disabled

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EncryptAdvertisingData(const *gapAdvertisingData_t* *pAdvertisingData, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)

This function is used to encrypt a series of AD structures into an encrypted AD structure.

Parameters

- pAdvertisingData – **[in]** Pointer to AD(s) to be encrypted.
- pKey – **[in]** Pointer to 16-byte key to be used for encryption (must be big endian).
- pIV – **[in]** Pointer to 8-byte initialization vector (must be little endian).
- pOutput – **[out]** Pointer for location of output encrypted data. Must be large enough to accommodate the size of the total data to be encrypted plus the size of the Randomizer and MIC fields.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – At least one of the provided parameters is invalid.

- gBleOutOfMemory_c – Could not allocate memory for the buffers used in CCM.
- gBleRngError_c – Could not generate the randomizer.
- gBleSecLibError_c – CCM call failed.

bleResult_t Gap_DecryptAdvertisingData(uint8_t *pData, uint16_t dataLength, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)

This function is used to decrypt advertising data.

Parameters

- pData – **[in]** Pointer to encrypted data (AD payload).
- dataLength – **[in]** Length of the encrypted data (includes randomizer and MIC fields).
- pKey – **[in]** Pointer to 16-byte key to be used for decryption (must be big endian).
- pIV – **[in]** Pointer to 8-byte initialization vector (must be little endian).
- pOutput – **[out]** Pointer to the location of decrypted data.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – At least one of the provided parameters is invalid.
- gBleOutOfMemory_c – Could not allocate memory for the buffers used in CCM.
- gBleSecLibError_c – CCM call failed.

void Gap_SetConnectionCallback(*gapConnectionCallback_t* pfConnectionCallback)

Set the GAP connection callback.

Parameters

- pfConnectionCallback – **[in]** The function called when one of the events defined in gapConnectionEventType_t occurs.

Returns

none

void Gap_SetScanningCallback(*gapScanningCallback_t* pfScanningCallback)

Set the GAP scanning callback.

Parameters

- pfScanningCallback – **[in]** The function called when one of the events defined in gapScanningEventType_t occurs.

Returns

none

Gap_AddSecurityModesAndLevels(modeLevelA, modeLevelB)

Macro used to combine two security mode-levels.

Remark

This macro is useful when two different security requirements must be satisfied at the same time, such as a device central security requirement and a service-specific security requirement.

Parameters

- modeLevelB (modeLevelA,) – **[in]** The two security mode-levels.

Returns

The resulting security mode-level.

Gap_CancelInitiatingConnection()

Macro used to cancel a connection initiated by Gap_Connect(...).

Remark

This macro can only be used for a connection that has not yet been established, such as the "gConnEvtConnected_c" has not been received. For example, call this when a connection request has timed out. Application should listen for gCreateConnectionCanceled_c generic event.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadAdvertisingTxPowerLevel()

Macro used to read the radio transmitter power when advertising.

Remark

The result is contained in the gAdvTxPowerLevelRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadRssi(deviceId)

Macro used to read the RSSI of a radio connection.

Remark

The result is contained in the gConnEvtRssiRead_c connection event. The RSSI value is a signed byte, and the unit is dBm. If the RSSI cannot be read, the gConnEvtPowerReadFailure_c connection event is generated.

Parameters

- deviceId – **[in]** Device ID identifying the radio connection.

Return values

- gBleSuccess_c –

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadTxPowerLevelInConnection(deviceId)

Macro used to read the radio transmitting power level of a radio connection.

Remark

The result is contained in the gConnEvtTxPowerLevelRead_c connection event. If the TX Power cannot be read, the gConnEvtPowerReadFailure_c connection event is generated.

Parameters

- deviceId – **[in]** Device ID identifying the radio connection.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_GetConnParams(deviceId)

Get the Connection parameters for the given deviceId.

Parameters

- deviceId – **[in]** Peer device Id.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

gCancelOngoingInitiatingConnection_d

Use this value as a parameter to the Gap_Disconnect(deviceId) function to cancel any ongoing connection initiation, for example if the connection has timed out.

gMode_2_Mask_d

Mask to check if a Security Mode-and-Level is Mode 2

getSecurityLevel(modeLevel)

Extracts the security level (see gapSecurityLevel_t) from the combined security mode-level (gapSecurityModeAndLevel_t).

getSecurityMode(modeLevel)

Extracts the security mode (see gapSecurityMode_t) from the combined security mode-level (gapSecurityModeAndLevel_t).

isMode_2(modeLevel)

isMode_1(modeLevel)

isSameMode(modeLevelA, modeLevelB)

addSameSecurityModes(modeLevelA, modeLevelB)

addMode1AndMode2(mode1, mode2)

addDifferentSecurityModes(modeLevelA, modeLevelB)

gDefaultEncryptionKeySize_d
: The default (minimum) value for the LTK size

gMaxEncryptionKeySize_d
: The maximum value for the LTK size

gGapDefaultDeviceSecurity_d
: The default value for the Device Security (no requirements)

gGapDefaultSecurityRequirements_d
: The default value for a Security Requirement

gGapAdvertisingIntervalRangeMinimum_c
: Minimum advertising interval (20 ms)

gGapAdvertisingIntervalDefault_c
: Default advertising interval (1.28 s)

gGapAdvertisingIntervalRangeMaximum_c
: Maximum advertising interval (10.24 s)

gGapExtAdvertisingIntervalRangeMinimum_c
: Minimum extended advertising interval (20 ms)

gGapExtAdvertisingIntervalDefault_c
: Default extended advertising interval (1.28 s)

gGapExtAdvertisingIntervalRangeMaximum_c
: Maximum extended advertising interval (10485.76 s)

gGapPeriodicAdvIntervalRangeMinimum_c
: Minimum periodic advertising interval (7.5 ms)

gGapPeriodicAdvIntervalDefault_c
: Default periodic advertising interval (2.56 s)

gGapPeriodicAdvIntervalRangeMaximum_c
: Maximum periodic advertising interval (81.91875 s)

gGapAdvertisingChannelMapDefault_c
: Default Advertising Channel Map - all 3 channels are enabled

gGapDefaultAdvertisingParameters_d
: Default value for Advertising Parameters struct

gGapDefaultExtAdvertisingParameters_d
: Default value for Extended Advertising Parameters struct

gGapDefaultPeriodicAdvParameters_d
: Default value for Periodic Advertising Parameters struct

gGapScanIntervalMin_d
: Minimum scan interval (2.5 ms)

gGapScanIntervalDefault_d
: Default scan interval (10 ms)

gGapScanIntervalMax_d
: Maximum scan interval (10.24 s)

gGapScanWindowMin_d
: Minimum scan window (2.5 ms)

gGapScanWindowDefault_d
: Default scan window (10 ms)

gGapScanWindowMax_d
: Maximum scan window (10.24 s)

gGapRssiMin_d
: Minimum valid value for RSSI (dB)

gGapRssiMax_d
: Maximum valid value for RSSI (dB)

gGapRssiNotAvailable_d
: A special invalid value for the RSSI indicating that the measurement is not available.

gGapScanContinuously_d
: Default value for Scanning duration - Scan continuously until explicitly disable

gGapScanPeriodicDisabled_d
: Default value for Scanning period - Periodic scanning disabled

gGapDefaultScanningParameters_d
: Default value for Scanning Parameters struct

gGapConnIntervalMin_d
: Minimum connection interval (7.5 ms)

gGapConnIntervalMax_d
: Maximum connection interval (4 s)

gGapConnLatencyMin_d
: Minimum connection latency value (0 - no connection event may be ignored)

gGapConnLatencyMax_d

Maximum connection latency value (499 connection events may be ignored)

gGapConnSuperTimeoutMin_d

Minimum supervision timeout (100 ms)

gGapConnSuperTimeoutMax_d

Maximum supervision timeout (32 s)

gGapConnEventLengthMin_d

Minimum value of the connection event length (0 ms)

gGapConnEventLengthMax_d

Maximum value of the connection event length (~41 s)

gGapDefaultConnectionLatency_d

Default connection latency: 0

gGapDefaultSupervisionTimeout_d

Default supervision timeout: 10s

gGapDefaultMinConnectionInterval_d

Default minimum connection interval: 100ms

gGapDefaultMaxConnectionInterval_d

Default maximum connection interval: 200ms

gGapDefaultConnectionRequestParameters_d

The default value for the Connection Request Parameters structure

gGapChSelAlgorithmNo2

“Channel Selection Algorithm #2 is used” value in LE Channel Selection Algorithm Event

gBlePeriodicAdvOngoingSyncCancelHandle

Sync handle value for which to call the create sync cancel command instead of terminate sync

gGapInvalidSyncHandle

Sync handle used to differentiate extended advertising reports from periodic advertising reports

gNone_c

Values of the AD Flags advertising data structure. No information.

gLeLimitedDiscoverableMode_c

This device is in Limited Discoverable mode.

gLeGeneralDiscoverableMode_c

This device is in General Discoverable mode.

gBrEdrNotSupported_c

This device supports only Bluetooth Low Energy; no support for Classic Bluetooth.

gSimultaneousLeBrEdrCapableController_c

This device's Controller also supports Classic Bluetooth.

gSimultaneousLeBrEdrCapableHost_c

This device's Host also supports Classic Bluetooth.

gNoKeys_c

Flags indicating the Keys to be exchanged by the SMP during the key exchange phase of pairing. No key can be distributed.

gLtk_c

Long Term Key.

gIrk_c

Identity Resolving Key.

gCsrk_c

Connection Signature Resolving Key.

gSecurityMode_1_c

LE Security Mode values for gapSecurityMode_t Mode 1 - Encryption required (except for Level 1).

gSecurityMode_2_c

Mode 2 - Data Signing required.

gSecurityLevel_NoSecurity_c

LE Security Level values for gapSecurityLevel_t No security (combined only with Mode 1).

gSecurityLevel_NoMitmProtection_c

Unauthenticated (no MITM protection).

gSecurityLevel_WithMitmProtection_c

Authenticated (MITM protection by PIN or OOB).

gSecurityLevel_LeSecureConnections_c

Authenticated with LE Secure Connections.

gSecurityMode_1_Level_1_c

Security Mode-and-Level definitions values for gapSecurityModeAndLevel_t Mode 1 Level 1 - No Security.

gSecurityMode_1_Level_2_c

Mode 1 Level 2 - Encryption without authentication.

gSecurityMode_1_Level_3_c

Mode 1 Level 3 - Encryption with authentication.

gSecurityMode_1_Level_4_c

Mode 1 Level 4 - Encryption with LE Secure Connections pairing.

gSecurityMode_2_Level_1_c

Mode 2 Level 1 - Data Signing without authentication.

gSecurityMode_2_Level_2_c

Mode 2 Level 2 - Data Signing with authentication.

gOobNotAvailable_c

Reason for rejecting the pairing request used by gapAuthenticationRejectReason_t. These values are equal to the corresponding reasons from SMP. This device does not have the required OOB for authenticated pairing.

gIncompatibleIoCapabilities_c

The combination of I/O capabilities does not allow pairing with the desired level of security.

gPairingNotSupported_c

This device does not support pairing.

gLowEncryptionKeySize_c

The peer's encryption key size is too low for this device's required security level.

gUnspecifiedReason_c

The host has rejected the pairing for an unknown reason.

gRepeatedAttempts_c

This device is the target of repeated unsuccessful pairing attempts and does not allow further pairing attempts at the moment.

gLinkEncryptionFailed_c

Link could not be encrypted. This reason may not be used by Gap_RejectPairing!

gIoDisplayOnly_c

I/O Capabilities as defined by the SMP used by gapIoCapabilities_t May display a PIN, no input.

gIoDisplayYesNo_c

May display a PIN and has a binary input (e.g., YES and NO buttons).

gIoKeyboardOnly_c

Has keyboard input, no display.

gIoNone_c

No input and no display.

gIoKeyboardDisplay_c
: Has keyboard input and display.

gGapCteMinLength_c
: AoA/AoD Min CTE length in 8 us units.

gGapCteMaxLength_c
: Max CTE length in 8 us units.

gGapCteMinCount_c
: Min number of CTEs to transmit in each periodic advertising interval.

gGapCteMaxCount_c
: Max number of CTEs to transmit in each periodic advertising interval.

gGapMinSwitchingPatternLength_c
: Min number of antennae in the pattern.

gGapMaxSwitchingPatternLength_c
: Max number of antennae in the pattern.

gGapEattMaxBearers
: Maximum number of bearer Ids in one GAP request/response.

gbmpDIAM_NonConnectableNonScannableUndirected_c

gbmpDIAM_ConnectableUndirected_c

gbmpDIAM_ScannableUndirected_c

struct gapSmpKeys_t
: *#include <gap_types.h>* Structure containing the SMP information exchanged during pairing.

Public Members

uint8_t cLtkSize
: Encryption Key Size. If aLtk is NULL, this is ignored.

uint8_t *aLtk
: Long Term (Encryption) Key. NULL if LTK is not distributed, else size is given by cLtkSize.

uint8_t *aIrk
: Identity Resolving Key. NULL if aIrk is not distributed.

uint8_t *aCsrk
: Connection Signature Resolving Key. NULL if aCsrk is not distributed.

uint8_t cRandSize

Size of RAND; usually equal to gcMaxRandSize_d. If aLtk is NULL, this is ignored.

uint8_t *aRand

RAND value used to identify the LTK. If aLtk is NULL, this is ignored.

uint16_t ediv

EDIV value used to identify the LTK. If aLtk is NULL, this is ignored.

bleAddressType_t addressType

Public or Random address. If aAddress is NULL, this is ignored.

uint8_t *aAddress

Device Address. NULL if address is not distributed. If aIrk is NULL, this is ignored.

struct gapSecurityRequirements_t

#include <gap_types.h> Security Requirements structure for a Device, a Service or a Characteristic

Public Members

gapSecurityModeAndLevel_t securityModeLevel

Security mode and level.

bool_t authorization

Authorization required.

uint16_t minimumEncryptionKeySize

Minimum encryption key (LTK) size. Ignored if gSecurityMode_1_Level_4_c is required (set to gMaxEncryptionKeySize_d automatically)

struct gapServiceSecurityRequirements_t

#include <gap_types.h> Service Security Requirements

Public Members

uint16_t serviceHandle

Handle of the Service declaration in the GATT Database.

gapSecurityRequirements_t requirements

Requirements for all attributes in this service.

struct gapDeviceSecurityRequirements_t

#include <gap_types.h> Device Security - Security Requirements + Service Security Requirements

Public Members

gapSecurityRequirements_t *pSecurityRequirements

Security requirements added to all services.

uint8_t cNumServices

Number of service-specific requirements; must be less than or equal to gcGapMaxServiceSpecificSecurityRequirements_c.

gapServiceSecurityRequirements_t *aServiceSecurityRequirements

Array of service-specific requirements.

struct gapHandleList_t

#include <gap_types.h> List of Attribute Handles for authorization lists.

Public Members

uint8_t cNumHandles

Number of handles in this list.

uint16_t aHandles[(1U)]

List of handles.

struct gapConnectionSecurityInformation_t

#include <gap_types.h> Connection Security Information structure

Public Members

bool_t authenticated

TRUE if pairing was performed with MITM protection.

gapHandleList_t authorizedToRead

List of handles the peer has been authorized to read.

gapHandleList_t authorizedToWrite

List of handles the peer has been authorized to write.

struct gapPairingParameters_t

#include <gap_types.h> Pairing parameters structure for the Gap_Pair and Gap_AcceptPairingRequest APIs

Public Members

bool_t withBonding

TRUE if this device is able to and wants to bond after pairing, FALSE otherwise.

gapSecurityModeAndLevel_t securityModeAndLevel

The desired security mode-level.

uint8_t maxEncryptionKeySize

Maximum LTK size supported by the device.

gapIoCapabilities_t localIoCapabilities

I/O capabilities used to determine the pairing method.

bool_t oobAvailable

TRUE if this device has Out-of-Band data that can be used for authenticated pairing. FALSE otherwise.

gapSmpKeyFlags_t centralKeys

Indicates the SMP keys to be distributed by the Central.

gapSmpKeyFlags_t peripheralKeys

Indicates the SMP keys to be distributed by the Peripheral.

bool_t leSecureConnectionSupported

Indicates if device supports LE Secure Connections pairing. Conflict if this is FALSE and securityModeAndLevel is gSecurityMode_1_Level_4_c.

bool_t useKeypressNotifications

Indicates if device supports Keypress Notification PDUs during Passkey Entry pairing. Conflict if this is TRUE and localIoCapabilities is set to gIoNone_c.

struct gapPeripheralSecurityRequestParameters_t

#include <gap_types.h> Parameters of a Peripheral Security Request.

Public Members

bool_t bondAfterPairing

TRUE if the Peripheral supports bonding.

bool_t authenticationRequired

TRUE if the Peripheral requires authentication for MITM protection.

struct gapAdvertisingParameters_t

#include <gap_types.h> Advertising Parameters; for defaults see gGapDefaultAdvertising-Parameters_d.

Public Members

uint16_t minInterval

Minimum desired advertising interval. Default: 1.28 s.

uint16_t maxInterval

Maximum desired advertising interval. Default: 1.28 s.

bleAdvertisingType_t advertisingType

Advertising type. Default: connectable undirected.

bleAddressType_t ownAddressType

Indicates whether the advertising address is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleAddressType_t peerAddressType

Address type of the peer; only used in directed advertising and Enhanced Privacy.

bleDeviceAddress_t peerAddress

Address of the peer; same as above.

gapAdvertisingChannelMapFlags_t channelMap

Bit mask indicating which of the three advertising channels are used. Default: all three.

gapAdvertisingFilterPolicy_t filterPolicy

Indicates whether the connect and scan requests are filtered using the Filter Accept List. Default: does not use Filter Accept List (process all).

struct gapExtAdvertisingParameters_tag

#include <gap_types.h> Extended Advertising Parameters; for defaults see gGapDefaultExtAdvertisingParameters_d.

Public Members

uint8_t SID

ID of the advertising set chosen by application. Shall be lower than gBleExtAdvMaxSetId_c

uint8_t handle

ID of the advertising set handled by controller. Shall be lower than gMaxAdvSets_c

uint32_t minInterval

Minimum desired advertising interval. Shall be at least equal or higher than gGapExtAdvertisingIntervalRangeMinimum_c

uint32_t maxInterval

Maximum desired advertising interval. Shall be higher than gGapExtAdvertisingIntervalRangeMinimum_c and higher than minInterval

bleAddressType_t ownAddressType

Indicates whether the advertising address is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleDeviceAddress_t ownRandomAddr

The random address used for advertising on the current handle

bleAddressType_t peerAddressType

Address type of the peer; only used in directed advertising and Enhanced Privacy.

bleDeviceAddress_t peerAddress

Address of the peer; same as above.

gapAdvertisingChannelMapFlags_t channelMap

Bit mask indicating which of the three advertising channels are used for primary advertising

gapAdvertisingFilterPolicy_t filterPolicy

Indicates whether the connect and scan requests are filtered using the Filter Accept List

bleAdvRequestProperties_t extAdvProperties

Type of advertising event

int8_t txPower

The maximum power level at which the adv packets are to be transmitted. The Controller shall choose a power level lower than or equal to the one specified by the Host. Valid range: -127 to 20

gapLePhyMode_t primaryPHY

The PHY on which the advertising packets are transmitted (1M or Coded PHY). Used for sending ADV_EXT_IND

gapLePhyMode_t secondaryPHY

The PHY used for sending AUX_ADV_IND PDU. Used only for Extended Advertising Events

uint8_t secondaryAdvMaxSkip

Maximum number of advertising events that can be skipped before the AUX_ADV_IND can be sent

bool_t enableScanReqNotification

Indicates whether the Controller shall send notifications upon the receipt of a scan request PDU

gapLePhyOptionsFlags_t primaryAdvPhyOptions

Preferred or required coding when transmitting on the primary LE Coded PHY

gapLePhyOptionsFlags_t secondaryAdvPhyOptions

Preferred or required coding when transmitting on the secondary LE Coded PHY

struct gapPeriodicAdvParameters_tag

#include <gap_types.h> Periodic Advertising Parameters [v2]; for defaults see gGapDefaultPeriodicAdvParameters_d.

Public Members

uint8_t handle

ID of the advertising set handled by controller. Shall be lower than gMaxAdvSets_c

bool_t addTxPowerInAdv

Set this option to include the Tx power in advertising packet.

uint16_t minInterval

Minimum advertising interval for periodic advertising.

uint16_t maxInterval

Maximum advertising interval for periodic advertising. Should be different and higher than minInterval.

uint8_t numSubevents

Number of subevents.

uint8_t subeventInterval

Interval between subevents.

uint8_t responseSlotDelay

Time between the advertising packet in a subevent and the first response slot.

uint8_t responseSlotSpacing

Time between response slots.

uint8_t numResponseSlots

Number of response slots.

struct gapPeriodicAdvSyncTransfer_tag

#include <gap_types.h>

Public Members

deviceId_t deviceId

Connection identifier

uint16_t serviceData

Value provided by the host

uint16_t syncHandle

Identifier for the periodic advertising train

struct gapPeriodicAdvSetInfoTransfer_tag

#include <gap_types.h>

Public Members

deviceId_t deviceId

Connection identifier

uint16_t serviceData

Value provided by the host

uint16_t advHandle

Identifier for the advertising set

struct gapSetPeriodicAdvSyncTransferParams_tag

#include <gap_types.h>

Public Members

deviceId_t deviceId

Connection identifier

gapPeriodicAdvSyncMode_t mode

The action to be taken when periodic advertising synchronization information is received

uint16_t skip

The number of periodic advertising packets that can be skipped after a successful receive

uint16_t syncTimeout

Synchronization timeout for the periodic advertising train

bleSyncCteType_t CTEType

0: Do not sync to packets with an AoA Constant Tone Extension 1: Do not sync to packets with an AoD Constant Tone Extension with 1 us slots 2: Do not sync to packets with an AoD Constant Tone Extension with 2 us slots 4: Do not sync to packets without a Constant Tone Extension

struct gapScanningParameters_t

#include <gap_types.h> Scanning parameters; for defaults see gGapDefaultScanningParameters_d.

Public Members

bleScanType_t type
: Scanning type. Default: passive.

uint16_t interval
: Scanning interval. Default: 10 ms.

uint16_t window
: Scanning window. Default: 10 ms.

bleAddressType_t ownAddressType
: Indicates whether the address used in scan requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleScanningFilterPolicy_t filterPolicy
: Indicates whether the advertising packets are filtered using the Filter Accept List. Default: does not use Filter Accept List (scan all).

gapLePhyFlags_t scanningPHYs
: Indicates the PHYs on which the advertising packets should be received on the primary advertising channel.

struct gapCreateSyncReqOptions_tag
: *#include <gap_types.h>* Create Sync Request Options

struct gapPeriodicAdvSyncReq_tag
: *#include <gap_types.h>* Periodic Advertising Sync Request parameters

Public Members

gapCreateSyncReqOptions_t options
: Bit 0: Filter Policy. Bit 1: Reporting enabled/disabled. Bit 2: Duplicate filtering disabled/enabled.

uint8_t SID
: The SID advertised by the periodic advertiser in the ADI field.

bleAddressType_t peerAddressType
: Periodic advertiser's address type (Public or Random)

bleDeviceAddress_t peerAddress
: Periodic advertiser's address

uint16_t skipCount
: The number of consecutive periodic advertising packets that the receiver may skip after successfully receiving a periodic advertising packet.

uint16_t timeout

The maximum permitted time between successful receives. If this time is exceeded, synchronization is lost.

bleSyncCteType_t cteType

Specifies whether to only sync to certain types of CTE

struct gapConnectionRequestParameters_t

#include <gap_types.h> Connection request parameter structure to be used in the Gap_Connect function; for API-defined defaults, use gGapDefaultConnectionRequestParameters_d.

Public Members

uint16_t scanInterval

Scanning interval. Default: 10 ms.

uint16_t scanWindow

Scanning window. Default: 10 ms.

bleInitiatorFilterPolicy_t filterPolicy

Indicates whether the connection request is issued for a specific device or for all the devices in the Filter Accept List. Default: specific device.

bleAddressType_t ownAddressType

Indicates whether the address used in connection requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address.

bleAddressType_t peerAddressType

When connecting to a specific device (see filterPolicy), this indicates that device's address type. Default: public address.

bleDeviceAddress_t peerAddress

When connecting to a specific device (see filterPolicy), this indicates that device's address.

uint16_t connIntervalMin

The minimum desired connection interval. Default: 100 ms.

uint16_t connIntervalMax

The maximum desired connection interval. Default: 200 ms.

uint16_t connLatency

The desired connection latency (the maximum number of consecutive connection events the Peripheral is allowed to ignore). Default: 0.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped. Default: 10 s.

uint16_t connEventLengthMin

The minimum desired connection event length. Default: 0 ms.

uint16_t connEventLengthMax

The maximum desired connection event length. Default: maximum possible, ~41 s. (lets the Controller decide).

bool_t usePeerIdentityAddress

If Controller Privacy is enabled and this parameter is TRUE, the address defined in the peerAddressType and peerAddress is an identity address. Otherwise, it is a device address.

gapLePhyFlags_t initiatingPHYs

Indicates the PHY on which the advertising packets should be received on the primary advertising channel and the PHY for which connection parameters have been specified.

struct gapConnectionFromPawrParameters_t

#include <gap_types.h> Connection request parameter structure to be used in the Gap_ConnectFromPawr function

Public Members

uint16_t scanInterval

Scanning interval. Default: 10 ms.

uint16_t scanWindow

Scanning window. Default: 10 ms.

bleInitiatorFilterPolicy_t filterPolicy

Indicates whether the connection request is issued for a specific device or for all the devices in the Filter Accept List. Default: specific device.

bleAddressType_t ownAddressType

Indicates whether the address used in connection requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address.

bleAddressType_t peerAddressType

When connecting to a specific device (see filterPolicy), this indicates that device's address type. Default: public address.

bleDeviceAddress_t peerAddress

When connecting to a specific device (see filterPolicy), this indicates that device's address.

uint16_t connIntervalMin

The minimum desired connection interval. Default: 100 ms.

uint16_t connIntervalMax

The maximum desired connection interval. Default: 200 ms.

uint16_t connLatency

The desired connection latency (the maximum number of consecutive connection events the Peripheral is allowed to ignore). Default: 0.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped. Default: 10 s.

uint16_t connEventLengthMin

The minimum desired connection event length. Default: 0 ms.

uint16_t connEventLengthMax

The maximum desired connection event length. Default: maximum possible, ~41 s. (lets the Controller decide).

bool_t usePeerIdentityAddress

If Controller Privacy is enabled and this parameter is TRUE, the address defined in the peerAddressType and peerAddress is an identity address. Otherwise, it is a device address.

gapLePhyFlags_t initiatingPHYs

Indicates the PHY on which the advertising packets should be received on the primary advertising channel and the PHY for which connection parameters have been specified.

uint8_t advHandle

Advertising handle identifying the periodic advertising train.

uint8_t subevent

Subevent where the connection request is to be sent.

struct gapConnectionParameters_t

#include <gap_types.h> Connection parameters as received in the gConnEvtConnected_c connection event.

Public Members

uint16_t connInterval

Interval between connection events.

uint16_t connLatency

Number of consecutive connection events the Peripheral may ignore.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

bleCentralClockAccuracy_t centralClockAccuracy

Accuracy of central's clock, allowing for frame detection optimizations.

struct gapGenerateDHKeyV2Params_t

#include <gap_types.h> DH Key Generate V2

Public Members

ecdhPublicKey_t remoteP256PublicKey

The remote P-256 public key

gapPrivateKeyType_t keyType

The private key type used for generating the DH key

struct gapConnectionlessCteTransmitParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionlessCteTransmitParameters function.

Public Members

uint8_t handle

Advertising set handle.

uint8_t cteLength

Constant Tone Extension length in 8 us.

bleCteType_t cteType

Type of CTE.

uint8_t cteCount

Number of CTEs to transmit in each periodic advertising interval.

uint8_t switchingPatternLength

Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]

List of Antenna IDs in pattern.

struct gapConnectionlessIqSamplingParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_EnableConnectionlessIqSampling function.

Public Members

bleIqSamplingEnable_t iqSamplingEnable

Enable or disable IQ sampling.

bleSlotDurations_t slotDurations
Switching and sampling slot durations.

uint8_t maxSampledCtes
Maximum number of CTEs to sample and report in each periodic adv interval.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteTransmitParams_t
#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionCteTransmitParameters function.

Public Members

bleCteAllowedTypesMap_t cteTypes
Allowed CTE types.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteReceiveParams_t
#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionCteReceiveParameters function.

Public Members

bleIqSamplingEnable_t iqSamplingEnable
Enable or disable IQ sampling.

bleSlotDurations_t slotDurations
Switching and sampling slot durations.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteReqEnableParams_t
#include <gap_types.h> Parameter structure to be used in the Gap_EnableConnectionCteRequest function.

Public Members

bleCteReqEnable_t cteReqEnable

Enable or disable CTE Req procedure.

uint16_t cteReqInterval

Requested interval for initiating CTE Req procedure.

uint8_t requestedCteLength

Minimum length of CTEs requested in 8 us units.

bleCteType_t requestedCteType

Requested types of CTE.

struct gapPathLossReportingParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_SetPathLossReportingParameters function.

Public Members

uint8_t highThreshold

High threshold for the path loss. Units: dB.

uint8_t highHysteresis

Hysteresis value for the high threshold. Units: dB.

uint8_t lowThreshold

Low threshold for the path loss. Units: dB.

uint8_t lowHysteresis

Hysteresis value for the low threshold. Units: dB.

uint16_t minTimeSpent

Minimum time in number of connection events to be observed once the path crosses the threshold before an event is generated.

struct gapAdStructure_t

#include <gap_types.h> Definition of an AD Structure as contained in Advertising and Scan Response packets. An Advertising or Scan Response packet contains several AD Structures.

Public Members

uint8_t length

Total length of the [adType + aData] fields. Equal to 1 + lengthOf(aData).

gapAdType_t adType

AD Type of this AD Structure.

uint8_t *aData

Data contained in this AD Structure; length of this array is equal to (*gapAdStructure_t.length* - 1).

struct gapAdvertisingData_t

#include <gap_types.h> Advertising Data structure : a list of several *gapAdStructure_t* structures.

Public Members

uint8_t cNumAdStructures

Number of AD Structures.

gapAdStructure_t *aAdStructures

Array of AD Structures.

struct gapAdvertisingDecisionData_t

#include <gap_types.h> Extended Advertising Decision Data structure.

Public Members

uint8_t *pKey

Pointer to the key for the resolvable tag. When pKey != NULL Decision data should contain only arbitrary data or pDecisionData should be NULL.

uint8_t *pPrand

Used only when pKey != NULL. When pPrand != NULL it should point the 3 bytes value of the prand. Otherwise it will be generated by the host.

uint8_t *pDecisionData

Pointer to the decision data. When pDecisionData != NULL decision data may contain the resolvable tag and/or arbitrary data.

uint8_t dataLength

The length of the decision data pointed by pDecisionData. When pKey is NULL it should not exceed 8. Otherwise it should not exceed 2

bool_t resolvableTagPresent

When pKey is NULL it indicates whether decision data contains the resolvable tag. In this case the 6 <= dataLength <= 8

struct gapDecisionInstructionsData_tag

#include <gap_types.h>

struct gapSubeventDataStructure_t

#include <gap_types.h> Definition of a Subevent Data structure as contained in Periodic Advertising subevents.

Public Members

uint8_t subevent
: Subevent index for this data.

uint8_t responseSlotStart
: The first response slots to be used in this subevent.

uint8_t responseSlotCount
: The number of response slots to be used.

gapAdvertisingData_t *pAdvertisingData
: Advertising data.

struct gapPeriodicAdvertisingSubeventData_t
: *#include <gap_types.h>* Advertising Data structure : a list of several *gapAdStructure_t* structures.

Public Members

uint8_t cNumSubevents
: Number of subevents.

gapSubeventDataStructure_t *aSubeventDataStructures
: Data for each subevent.

struct gapPeriodicAdvertisingResponseData_t
: *#include <gap_types.h>* Periodic Advertising Response data

Public Members

uint16_t requestEvent
: The value of paEventCounter (see [Vol 6] Part B, Section 4.4.2.1) for the periodic advertising packet that the Host is responding to

uint8_t requestSubevent
: The subevent for the periodic advertising packet that the Host is responding to.

uint8_t responseSubevent
: Used to identify the subevent of the PAwR train.

uint8_t responseSlot
: Used to identify the response slot of the PAwR train.

gapAdvertisingData_t *pResponseData
: Response data.

struct gapPeriodicSyncSubeventParameters_t

#include <gap_types.h> Periodic Sync Subevent Parameters

Public Members

uint16_t perAdvProperties

Bit 6: Include TxPower in the advertising PDU

uint8_t numSubevents

Number of subevents.

uint8_t aSubevents[1]

List of subevents

struct gapExtScanNotification_t

#include <gap_types.h>

Public Members

uint8_t handle

Advertising Handle

bleAddressType_t scannerAddrType

Scanner device's address type.

bleDeviceAddress_t aScannerAddr

Scanner device's address.

bool_t scannerAddrResolved

Whether the address corresponds to Resolved Private Address.

struct gapAdvertisingSetTerminated_t

#include <gap_types.h>

Public Members

bleResult_t status

Status of advertising set termination

uint8_t handle

Advertising Handle

deviceId_t deviceId

Valid only if the advertising ended with a connection

uint8_t numCompletedExtAdvEvents
: Number of advertising events sent by Controller

struct gapPerAdvSubeventDataRequest_t
: *#include <gap_types.h>*

Public Members

uint8_t handle
: Used to identify a periodic advertising train.

uint8_t subeventStart
: The first subevent that data is requested for.

uint8_t subeventDataCount
: The number of subevents that data is requested for.

struct gapPerAdvResponse_t
: *#include <gap_types.h>*

struct gapAdvertisingEvent_t
: *#include <gap_types.h>* Advertising event structure: type + data.

Public Members

gapAdvertisingEventType_t eventType
: Event type.

union *gapAdvertisingEvent_t* eventData
: Event data, to be interpreted according to *gapAdvertisingEvent_t.eventType*.

struct gapScannedDevice_t
: *#include <gap_types.h>* Scanned device information structure, obtained from LE Advertising Reports.

Public Members

bleAddressType_t addressType
: Device's advertising address type.

bleDeviceAddress_t aAddress
: Device's advertising address.

int8_t rssi
: RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

uint8_t dataLength

Length of the advertising or scan response data.

uint8_t *data

Advertising or scan response data.

bleAdvertisingReportEventType_t advEventType

Advertising report type, indicating what type of event generated this data (advertising, scan response).

bool_t directRpaUsed

TRUE if directed advertising with Resolvable Private Address as Direct Address was detected while Enhanced Privacy is enabled.

bleDeviceAddress_t directRpa

Resolvable Private Address set as Direct Address for directed advertising. Valid only when directRpaUsed is TRUE.

bool_t advertisingAddressResolved

If this is TRUE, the address contained in the addressType and aAddress fields is the identity address of a resolved RPA from the Advertising Address field. Otherwise, the address from the respective fields is the public or random device address contained in the Advertising Address field.

struct gapExtScannedDevice_t

#include <gap_types.h>

Public Members

bleAddressType_t addressType

Device's advertising address type.

bleDeviceAddress_t aAddress

Device's advertising address.

uint8_t SID

Advertising set id

bool_t advertisingAddressResolved

If this is TRUE, the address contained in the addressType and aAddress fields is the identity address of a resolved RPA from the Advertising Address field. Otherwise, the address from the respective fields is the public or random device address contained in the Advertising Address field.

bleAdvReportEventProperties_t advEventProperties

Advertising report properties, indicating what type of event generated this data (advertising, scan response).

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

int8_t txPower

The Tx power level of the advertiser

uint8_t primaryPHY

Advertiser PHY for primary channel

uint8_t secondaryPHY

Advertiser PHY for secondary channel

uint16_t periodicAdvInterval

Interval of the periodic advertising. Zero if not periodic advertising.

bool_t directRpaUsed

TRUE if directed advertising with Resolvable Private Address as Direct Address was detected while Enhanced Privacy is enabled.

bleAddressType_t directRpaType

Address type for directed advertising. Valid only when directRpaUsed is TRUE.

bleDeviceAddress_t directRpa

Resolvable Private Address set as Direct Address for directed advertising. Valid only when directRpaUsed is TRUE.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData

Advertising or scan response data.

struct gapPeriodicScannedDevice_t

#include <gap_types.h>

Public Members

uint16_t syncHandle

Sync Handle

int8_t txPower

The Tx power level of the advertiser

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

bleCteType_t cteType

Type of Constant Tone Extension in the periodic advertising packets.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData

Advertising or scan response data.

struct gapPeriodicScannedDeviceV2_t

#include <gap_types.h>

Public Members

uint16_t syncHandle

Sync Handle

int8_t txPower

The Tx power level of the advertiser

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

bleCteType_t cteType

Type of Constant Tone Extension in the periodic advertising packets.

uint16_t periodicEventCounter

The value of paEventCounter (see [Vol 6] Part B, Section 4.4.2.1) for the reported periodic advertising packet.

uint8_t subevent

Subevent number.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData

Advertising or scan response data.

struct gapSyncEstbEventData_t

#include <gap_types.h>

Public Members

bleResult_t status

Status of the Sync Established Event

uint16_t syncHandle
: Sync Handle

uint8_t SID
: Value of the Advertising SID subfield in the ADI field of the PDU

bleAddressType_t peerAddressType
: Advertiser Address Type

bleDeviceAddress_t peerAddress
: Advertiser Address

gapLePhyMode_t PHY
: Advertiser PHY

uint16_t periodicAdvInterval
: Periodic advertising interval

bleAdvertiserClockAccuracy_t advertiserClockAccuracy
: Advertiser Clock Accuracy

uint8_t numSubevents
: Only relevant for v2 of the event, otherwise value is 0

uint8_t subeventInterval
: Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotDelay
: Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotSpacing
: Only relevant for v2 of the event, otherwise value is 0

struct gapSyncLostEventData_t
: *#include <gap_types.h>*

Public Members

uint16_t syncHandle
: Sync Handle

struct gapConnectionlessIqReport_t
: *#include <gap_types.h>*

Public Members

uint16_t syncHandle
Sync Handle.

uint8_t channelIndex
Index of the channel on which the packet was received.

int16_t rssi
RSSI of the packet.

uint8_t rssiAntennaId
Antenna ID.

bleCteType_t cteType
Type of CTE on the received packet (AoA, AoD 1 us, AoD 2 us).

bleSlotDurations_t slotDurations
Durations of switching/sampling slots.

bleIqReportPacketStatus_t packetStatus
Packet status, information about CRC.

uint16_t periodicEventCounter
The value of paEventCounter for the reported AUX_SYNC_IND PDU.

uint8_t sampleCount
Total number of sample pairs.

int8_t *aI_samples
List of sampleCount I_samples.

int8_t *aQ_samples
List of sampleCount Q_samples.

struct gapMonAdvReportReport_t
#include <gap_types.h>

Public Members

bleAddressType_t peerAddressType
Advertiser Address Type

bleDeviceAddress_t peerAddress
Advertiser Address

bleMonAdvCondition_t condition
Condition parameter

struct gapScanningEvent_t
#include <gap_types.h> Scanning event structure: type + data.

Public Members

gapScanningEventType_t eventType

Event type.

union *gapScanningEvent_t* eventData

Event data, to be interpreted according to *gapScanningEvent_t.eventType*.

struct gapConnectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtConnected_c event.

Public Members

gapConnectionParameters_t connParameters

Connection parameters established by the Controller.

bleAddressType_t peerAddressType

Connected device's address type.

bleDeviceAddress_t peerAddress

Connected device's address.

bool_t peerRpaResolved

If this is TRUE, the address defined by peerAddressType and peerAddress is the identity address of the peer, and the peer used an RPA that was resolved by the Controller and is contained in the peerRpa field. Otherwise, it is a device address. This parameter is irrelevant if Controller Privacy is not enabled.

bleDeviceAddress_t peerRpa

Peer Resolvable Private Address if Controller Privacy is active and peerRpaResolved is TRUE.

bool_t localRpaUsed

If this is TRUE, the Controller has used an RPA contained in the localRpa field. This parameter is irrelevant if Controller Privacy is not enabled.

bleDeviceAddress_t localRpa

Local Resolvable Private Address if Controller Privacy is active and localRpaUsed is TRUE.

bleLlConnectionRole_t connectionRole

Connection Role - central or peripheral.

uint8_t advHandle

Only relevant if the connection was from PAWR.

uint16_t syncHandle

Only relevant if the connection was from PAWR.

struct gapKeyExchangeRequestEvent_t

#include <gap_types.h> Event data structure for the gConnEvtKeyExchangeRequest_c event.

Public Members

gapSmpKeyFlags_t requestedKeys

Mask identifying the keys being requested.

uint8_t requestedLtkSize

Requested size of the encryption key.

struct gapKeysReceivedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtKeysReceived_c event.

Public Members

gapSmpKeys_t *pKeys

The SMP keys distributed by the peer.

struct gapAuthenticationRejectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtAuthenticationRejected_c event.

Public Members

gapAuthenticationRejectReason_t rejectReason

Peripheral's reason for rejecting the authentication.

struct gapPairingCompleteEvent_t

#include <gap_types.h> Event data structure for the gConnEvtPairingComplete_c event.

Public Members

bool_t pairingSuccessful

TRUE if pairing succeeded, FALSE otherwise.

union *gapPairingCompleteEvent_t* pairingCompleteData

Information of completion, selected upon the value of *gapPairingCompleteEvent_t.pairingSuccessful*.

struct gapLongTermKeyRequestEvent_t

#include <gap_types.h> Event data structure for the gConnEvtLongTermKeyRequest_c event.

Public Members

uint16_t ediv
: The Encryption Diversifier, as defined by the SMP.

uint8_t aRand[(8U)]
: The Random number, as defined by the SMP.

uint8_t randSize
: Usually equal to gcMaxRandSize_d.

struct gapEncryptionChangedEvent_t
: *#include <gap_types.h>* Event data structure for the gConnEvtEncryptionChanged_c event.

Public Members

bool_t newEncryptionState
: TRUE if link has been encrypted, FALSE if encryption was paused or removed.

struct gapDisconnectedEvent_t
: *#include <gap_types.h>* Event data structure for the gConnEvtDisconnected_c event.

Public Members

gapDisconnectionReason_t reason
: Reason for disconnection.

struct gapConnParamsUpdateReq_t
: *#include <gap_types.h>* Event data structure for the gConnEvtParameterUpdateRequest_c event.

Public Members

uint16_t intervalMin
: Minimum interval between connection events.

uint16_t intervalMax
: Maximum interval between connection events.

uint16_t peripheralLatency
: Number of consecutive connection events the Peripheral may ignore.

uint16_t timeoutMultiplier
: The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

struct gapConnParamsUpdateComplete_t

#include <gap_types.h> Event data structure for the gConnEvtParameterUpdateComplete_c event.

Public Members

uint16_t connInterval

Interval between connection events.

uint16_t connLatency

Number of consecutive connection events the Peripheral may ignore.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

struct gapConnLeDataLengthChanged_t

#include <gap_types.h> Event data structure for the gConnEvtLeDataLengthChanged_c event.

Public Members

uint16_t maxTxOctets

The maximum number of payload octets in a Link Layer Data Channel PDU to transmit on this connection.

uint16_t maxTxTime

The maximum time that the local Controller will take to send a Link Layer Data Channel PDU on this connection.

uint16_t maxRxOctets

The maximum number of payload octets in a Link Layer Data Channel PDU to receive on this connection.

uint16_t maxRxTime

The maximum time that the local Controller will take to receive a Link Layer Data Channel PDU on this connection.

struct gapConnIqReport_t

#include <gap_types.h> Event data structure for the gConnEvtIqReportReceived_c event.

Public Members

gapLePhyMode_t rxPhy

Identifies receiver PHY for this connection.

uint8_t dataChannelIndex
: Index of the data channel on which the Data Physical Channel PDU was received.

int16_t rssi
: RSSI of the packet.

uint8_t rssiAntennaId
: Antenna ID.

bleCteType_t cteType
: Type of CTE on the received packet (AoA, AoD 1 us, AoD 2 us).

bleSlotDurations_t slotDurations
: Durations of switching/sampling slots.

bleIqReportPacketStatus_t packetStatus
: Packet status, information about CRC.

uint16_t connEventCounter
: The value of the connEventCounter for the reported PDU.

uint8_t sampleCount
: Total number of sample pairs.

int8_t *aI_samples
: List of sampleCount I_samples

int8_t *aQ_samples
: List of sampleCount Q_samples

struct gapConnCteRequestFailed_t
: *#include <gap_types.h>* Event data structure for the gConnEvtCteRequestFailed_c event.

Public Members

bleResult_t status
: Status (0x00 - received response but without CTE, 0x01-0xFF - peer rejected request

struct gapPathLossThresholdEvent_t
: *#include <gap_types.h>* Event data structure for the gConnEvtPathLossThreshold_c event.

Public Members

uint8_t currentPathLoss
: Current path loss. Units: dB.

blePathLossThresholdZoneEntered_t zoneEntered

Low, middle or high.

struct gapTransmitPowerReporting_t

#include <gap_types.h> Event data structure for the gConnEvtTransmitPowerReporting_c event.

Public Members

bleTxPowerReportingReason_t reason

Reason for generating this event

- local tx power changed
- remote tx power changed
- a HCI_LE_Read_Remote_Transmit_Power_Level command completed

blePowerControlPhyType_t phy

The PHY for which the event is generated.

int8_t txPowerLevel

New transmit power level on this connection and PHY.

bleTxPowerLevelFlags_t flags

Tx power level flags.

int8_t delta

Change in tx power level. Units: dB.

struct gapTransmitPowerInfo_t

#include <gap_types.h> Event data structure for the gConnEvtEnhancedReadTransmitPowerLevel_c event.

Public Members

blePowerControlPhyType_t phy

The PHY for which the event is generated.

int8_t currTxPowerLevel

Current tx power level on this connection and PHY.

int8_t maxTxPowerLevel

Maximum tx power level on this connection and PHY.

struct gapEattConnectionRequest_t

#include <gap_types.h> Event data structure for the gConnEvtEattConnectionRequest_c event.

struct gapEattConnectionComplete_t

#include <gap_types.h> Event data structure for the gConnEvtEattConnectionComplete_c event.

struct gapEattReconfigureResponse_t

#include <gap_types.h> Event data structure for the gConnEvtEattChannelReconfigureResponse_c event.

struct gapEattBearerStatusNotification_t

#include <gap_types.h> Event data structure for the bearer status update event.

struct gapHandoverConnectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtHandoverConnected_c event.

Public Members

bleAddressType_t peerAddressType

Connected device's address type.

bleDeviceAddress_t peerAddress

Connected device's address.

bleLlConnectionRole_t connectionRole

Connection Role - master or slave.

struct gapHandoverDisconnectedEvent_t

#include <gap_types.h> Event data structure for the gHandoverDisconnected_c event.

Public Members

bleResult_t status

Status for the handover disconnect command

struct gapConnectionEvent_t

#include <gap_types.h> Connection event structure: type + data.

Public Members

gapConnectionEventType_t eventType

Event type

union *gapConnectionEvent_t* eventData

Event data, to be interpreted according to *gapConnectionEvent_t.eventType*.

struct gapIdentityInformation_t

#include <gap_types.h> Identity Information structure definition

Public Members

bleIdentityAddress_t identityAddress
Identity Address - Public or Random Static

uint8_t irk[(16U)]
Identity Resolving Key - must not be all-zero if Network Privacy is used

blePrivacyMode_t privacyMode
Privacy mode to be used for the entry on the resolving list

struct gapAutoConnectParams_t
#include <gap_types.h> Parameters for the Auto Connect Scan Mode.

Public Members

uint8_t cNumAddresses
Number of device addresses to automatically connect to.

bool_t writeInFilterAcceptList
If set to TRUE, the device addresses are written in the Filter Accept List before scanning is enabled.

gapConnectionRequestParameters_t *aAutoConnectData
The array of connection request parameters, of size equal to cNumAddresses.

struct gapHostVersion_tag
#include <gap_types.h>

union testParameters

Public Members

uint8_t resolvableTagKey[(16U)]

struct *gapDecisionInstructionsData_tag* arbitraryData

struct *gapDecisionInstructionsData_tag* rssi

struct *gapDecisionInstructionsData_tag* pathLoss

struct *gapDecisionInstructionsData_tag* advA

gapDecisionInstructionsAdvMode_t advMode

struct arbitraryData

struct rssi

struct pathLoss

struct advA

union eventData

Public Members

bleResult_t failReason

Event data for gAdvertisingCommandFailed_c event type: reason of failure to enable or disable advertising.

gapExtScanNotification_t scanNotification

Event data for gExtScanNotification_c event type: Scan Request Received Event.

gapAdvertisingSetTerminated_t advSetTerminated

Event received when advertising in a given advertising set has stopped.

gapPerAdvSubeventDataRequest_t subeventDataRequest

Event received when doing PAWR.

gapPerAdvResponse_t perAdvResponse

Periodic Advertising Response.

uint8_t advHandle

Event data for gExtAdvertisingStateChanged_c event type.

union eventData

Public Members

bleResult_t failReason

Event data for gScanCommandFailed_c or gPeriodicAdvSyncEstablished_c event type: reason of failure to enable/disable scanning or to establish sync.

gapScannedDevice_t scannedDevice

Event data for gDeviceScanned_c event type: scanned device information.

gapExtScannedDevice_t extScannedDevice

Event data for gExtDeviceScanned_c event type: extended scanned device information.

gapPeriodicScannedDevice_t periodicScannedDevice

Event data for gPeriodicDeviceScanned_c event type.

gapPeriodicScannedDeviceV2_t periodicScannedDeviceV2

Event data for gPeriodicDeviceScannedV2_c event type.

gapSyncEstbEventData_t syncEstb

Event data for gPeriodicAdvSyncEstablished_c event type: Sync handle information for the application.

gapSyncLostEventData_t syncLost

Event data for gPeriodicAdvSyncLost_c event type: Sync handle information for the application.

gapConnectionlessIqReport_t iqReport

Event data for gConnectionlessIqReportReceived_c event type: IQ information for the application.

gapMonAdvReportReport_t monAdvReport

Event data for gMonAdvReportEventReceived_c event type.

union pairingCompleteData

Public Members

bool_t withBonding

If pairingSuccessful is TRUE, this indicates whether the devices bonded.

bleResult_t failReason

If pairingSuccessful is FALSE, this contains the reason of failure.

union eventData

Public Members

gapConnectedEvent_t connectedEvent

Data for gConnEvtConnected_c: information about the connection parameters.

gapPairingParameters_t pairingEvent

Data for gConnEvtPairingRequest_c, gConnEvtPairingResponse_c: pairing parameters.

gapAuthenticationRejectedEvent_t authenticationRejectedEvent

Data for gConnEvtAuthenticationRejected_c: reason for rejection.

gapPeripheralSecurityRequestParameters_t peripheralSecurityRequestEvent

Data for gConnEvtPeripheralSecurityRequest_c: Peripheral's security requirements.

gapKeyExchangeRequestEvent_t keyExchangeRequestEvent

Data for gConnEvtKeyExchangeRequest_c: mask indicating the keys that were requested by the peer.

gapKeysReceivedEvent_t keysReceivedEvent

Data for gConnEvtKeysReceived_c: the keys received from the peer.

gapPairingCompleteEvent_t pairingCompleteEvent

Data for gConnEvtPairingComplete_c: fail reason or (if successful) bonding state.

gapLongTermKeyRequestEvent_t longTermKeyRequestEvent

Data for gConnEvtLongTermKeyRequest_c: encryption diversifier and random number.

gapEncryptionChangedEvent_t encryptionChangedEvent

Data for gConnEvtEncryptionChanged_c: new encryption state.

gapDisconnectedEvent_t disconnectedEvent

Data for gConnEvtDisconnected_c: reason for disconnection.

int8_t rssi_dBm

Data for gConnEvtRssiRead_c: value of the RSSI in dBm.

int8_t txPowerLevel_dBm

Data for gConnEvtTxPowerLevelRead_c: value of the TX power.

bleResult_t failReason

Data for gConnEvtPowerReadFailure_c: reason for power reading failure.

uint32_t passkeyForDisplay

gapConnParamsUpdateReq_t connectionUpdateRequest

Data for gConnEvtParameterUpdateRequest_c: connection parameters update.

gapConnParamsUpdateComplete_t connectionUpdateComplete

Data for gConnEvtParameterUpdateComplete_c: connection parameters update.

gapConnLeDataLengthChanged_t leDataLengthChanged

Data for gConnEvtLeDataLengthChanged_c: new data length parameters

gapKeypressNotification_t incomingKeypressNotification

uint32_t numericValueForDisplay

bleChannelMap_t channelMap

Data for gConnEvtChannelMapRead_c: channel map read from the Controller

gapConnIqReport_t connIqReport

Data for gConnEvtIqReportReceived_c: IQ information received from a peer

gapConnCteRequestFailed_t cteRequestFailedEvent

Data for gConnEvtCteRequestFailed_c: CTE Request to peer has failed

bleResult_t perAdvSyncTransferStatus

gapPathLossThresholdEvent_t pathLossThreshold

Data for gConnEvtPathLossThreshold_c: current path loss and zone entered.

gapTransmitPowerReporting_t transmitPowerReporting

Data for gConnEvtTransmitPowerReporting_c: changes in power levels.

gapTransmitPowerInfo_t transmitPowerInfo

Data for gConnEvtEnhancedReadTransmitPowerLevel_c: local power level information.

gapEattConnectionRequest_t eattConnectionRequest

Data for gConnEvtEattConnectionRequest_c: notification of L2Cap Enhanced Connect request for EATT.

gapEattConnectionComplete_t eattConnectionComplete

Data for gConnEvtEattConnectionComplete_c: result of *Gap_EattConnectionRequest()*.

gapEattReconfigureResponse_t eattReconfigureResponse

Data for gConnEvtEattChannelReconfigureResponse_c: result of *Gap_EattReconfigureRequest()*.

gapEattBearerStatusNotification_t eattBearerStatusNotification

Data for gConnEvtEattBearerStatusNotification_c: bearer status update.

gapHandoverConnectedEvent_t handoverConnectedEvent

Data for gConnEvtHandoverConnected_c: information about the connection parameters.

gapHandoverDisconnectedEvent_t handoverDisconnectedEvent

Data for gHandoverDisconnected_c: status of the Gap_HandoverDisconnect.

bleResult_t smError

Data for gConnEvtSmError_c: SM error status.

GATT

enum attErrorCode_t

ATT error codes

Values:

enumerator gAttErrCodeNoError_c

No error has occured.

enumerator gAttErrCodeInvalidHandle_c
The attribute handle given was not valid on this server.

enumerator gAttErrCodeReadNotPermitted_c
The attribute cannot be read.

enumerator gAttErrCodeWriteNotPermitted_c
The attribute cannot be written.

enumerator gAttErrCodeInvalidPdu_c
The attribute PDU was invalid.

enumerator gAttErrCodeInsufficientAuthentication_c
The attribute requires authentication before it can be read or written.

enumerator gAttErrCodeRequestNotSupported_c
ATT Server does not support the request received from the client.

enumerator gAttErrCodeInvalidOffset_c
Offset specified was past the end of the attribute.

enumerator gAttErrCodeInsufficientAuthorization_c
The attribute requires authorization before it can be read or written.

enumerator gAttErrCodePrepareQueueFull_c
Too many prepare writes have been queued.

enumerator gAttErrCodeAttributeNotFound_c
No attribute found within the given attribute handle range.

enumerator gAttErrCodeAttributeNotLong_c
The attribute cannot be read using the ATT_READ_BLOB_REQ PDU.

enumerator gAttErrCodeInsufficientEncryptionKeySize_c
The Encryption Key Size used for encrypting this link is too short.

enumerator gAttErrCodeInvalidAttributeValueLength_c
The attribute value length is invalid for the operation.

enumerator gAttErrCodeUnlikelyError_c
The attribute request could not be completed due to an unlikely error.

enumerator gAttErrCodeInsufficientEncryption_c
The attribute requires encryption before it can be read or written.

enumerator gAttErrCodeUnsupportedGroupType_c
The attribute type is not a supported grouping attribute.

enumerator gAttErrCodeInsufficientResources_c

Insufficient Resources to complete the request.

enumerator gAttErrCodeDatabaseOutOfSync_c

The server requests the client to rediscover the database.

enumerator gAttErrCodeValueNotAllowed_c

The attribute parameter value was not allowed.

enumerator gAttErrCodeWriteRequestRejected_c

Write request cannot be fulfilled for reasons other than permissions.

enumerator gAttErrCodeCccdImproperlyConfigured_c

CCCD is not configured according to the requirements of the profile or service.

enumerator gAttErrCodeProcedureAlreadyInProgress_c

Profile or service request cannot be serviced because another operation is still in progress.

enumerator gAttErrCodeOutOfRange_c

Attribute value is out of range as defined by a profile or service specification.

enum gattCachingClientState_c

GATT Caching Client states

Values:

enumerator gGattClientChangeUnaware_c

Gatt client is “change-unaware” regarding changes in the database definitions on the Server.

enumerator gGattClientStateChangePending_c

Gatt client is in the process of going from “change-unaware” to “change-aware”.

enumerator gGattClientChangeAware_c

Gatt client is “change-aware” regarding the database definitions on the Server.

enum gattClientHashUpdateType_t

Hash Update type

Values:

enumerator gattClientFirstConnection_c

Hash update for the first connection with a peer

enumerator gattClientReconnectBondedPeer_c

Hash update after reconnecting with a bonded peer

enumerator gattClientActiveConnectionUpdate

Hash update when a change occurs during an active connection

enumerator gattClientNoChange
Idle state - no change

enum gattProcedureType_t
GATT Client Procedure type
Values:

enumerator gGattProcExchangeMtu_c
MTU Exchange

enumerator gGattProcDiscoverAllPrimaryServices_c
Primary Service Discovery

enumerator gGattProcDiscoverPrimaryServicesByUuid_c
Discovery of Services by UUID

enumerator gGattProcFindIncludedServices_c
Discovery of Included Services within a Service range

enumerator gGattProcDiscoverAllCharacteristics_c
Characteristic Discovery within Service range

enumerator gGattProcDiscoverCharacteristicByUuid_c
Characteristic Discovery by UUID

enumerator gGattProcDiscoverAllCharacteristicDescriptors_c
Characteristic Descriptor Discovery

enumerator gGattProcReadCharacteristicValue_c
Characteristic Reading using Value handle

enumerator gGattProcReadUsingCharacteristicUuid_c
Characteristic Reading by UUID

enumerator gGattProcReadMultipleCharacteristicValues_c
Reading multiple Characteristics at once

enumerator gGattProcWriteCharacteristicValue_c
Characteristic Writing

enumerator gGattProcReadCharacteristicDescriptor_c
Reading Characteristic Descriptors

enumerator gGattProcWriteCharacteristicDescriptor_c
Writing Characteristic Descriptors

enumerator gGattProcUpdateDatabaseCopy_c
Inform the application to update its database copy

enumerator gGattProcSignalServiceDiscoveryComplete_c
Inform the application that service discovery has finished

enumerator gGattProcReadMultipleVarLengthCharValues_c
Read Multiple Variable Length Characteristic Values

enum gattProcedurePhase_t
GATT Procedure phase

Values:

enumerator gattProcPhaseInitiated
GATT procedure has been initiated.

enumerator gattProcPhaseRunning
GATT procedure is running.

typedef struct *gattService_tag* gattService_t
GATT Service structure definition

typedef uint8_t gattCccdFlags_t
Flags for the value of the Client Characteristic Configuration Descriptor.

bleResult_t Gatt_Init(void)
Initializes the GATT module.

Remark

If the GAP module is present, this function is called internally by *Ble_HostInitialize()*. Otherwise, the application must call this function once at device start-up.

Remark

This function executes synchronously.

Return values
gBleSuccess_c –

bleResult_t Gatt_GetMtu(*deviceId_t* deviceId, uint16_t *pOutMtu)
Retrieves the MTU used with a given connected device.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- pOutMtu – **[out]** Pointer to integer to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.

gCccdEmpty_c
: Nothing is enabled.

gCccdNotification_c
: Enables notifications.

gCccdIndication_c
: Enabled indications.

struct gattAttribute_t
: *#include <gatt_types.h>* GATT Attribute structure definition

Public Members

uint16_t handle
: Attribute handle.

bleUuidType_t uuidType
: Type of the UUID.

bleUuid_t uuid
: The attribute's UUID.

uint16_t valueLength
: Length of the attribute value array.

uint16_t maxValueLength
: Maximum length of the attribute value array; if this is set to 0, then the attribute's length is fixed and cannot be changed.

uint8_t *paValue
: Attribute value array.

struct gattCharacteristic_t
: *#include <gatt_types.h>* GATT Characteristic structure definition

Public Members

gattCharacteristicPropertiesBitFields_t properties
: Characteristic Properties as defined by GATT.

gattAttribute_t value
: Characteristic Value attribute.

uint8_t cNumDescriptors

Size of the Characteristic Descriptors array.

gattAttribute_t *aDescriptors

Characteristic Descriptors array.

struct gattService_tag

#include <gatt_types.h> GATT Service structure definition

Public Members

uint16_t startHandle

The handle of the Service Declaration attribute.

uint16_t endHandle

The last handle belonging to this Service (followed by another Service declaration of the end of the database).

bleUuidType_t uuidType

Service UUID type.

bleUuid_t uuid

Service UUID.

uint8_t cNumCharacteristics

Size of the Characteristic array.

gattCharacteristic_t *aCharacteristics

Characteristic array.

uint8_t cNumIncludedServices

Size of the Included Services array.

struct *gattService_tag* *aIncludedServices

Included Services array.

struct gattDbCharPresFormat_t

#include <gatt_types.h> Characteristic Presentation Format Descriptor structure

Public Members

uint8_t format

Format of the value of this characteristic.

int8_t exponent

Exponent field to determine how the value of this characteristic is further formatted.

uint16_t unitUuid16
: The unit of this characteristic.

uint8_t ns
: The name space of the description.

uint16_t description
: The description of this characteristic.

struct gattHandleRange_t
: *#include <gatt_types.h>* GATT Handle Range structure definition

Public Members

uint16_t startHandle
: Start Handle.

uint16_t endHandle
: End Handle - shall be greater than or equal to Start Handle.

struct procStatus_t
: *#include <gatt_types.h>* GATT Procedure Status structure definition

Public Members

bool_t isOngoing
: Indicates whether the GATT procedure type is ongoing.

gattProcedureType_t ongoingProcedureType
: Procedure type.

gattProcedurePhase_t ongoingProcedurePhase
: Procedure phase.

struct procDataStruct_t
: *#include <gatt_types.h>* GATT Procedure Data structure definition. Structure members are ordered to minimize the padding size

Public Members

uint16_t index
: Number of entries currently stored.

uint16_t max
: Maximum number of entries to store.

bleUuid_t charUuid
: UUID of the characteristic to be discovered.

bleUuidType_t charUuidType
: UUID type of the characteristic to be discovered.

bool_t reliableLongWrite
: Indicates whether to do reliable long writes.

bool_t bAllocatedArray
: Indicates whether the array is allocated.

union *procDataStruct_t* pOutActualCount
: Pointer to the memory locatio where to store the final number of added entries.

union *procDataStruct_t* array
: Array where data is stored.

union *procDataStruct_t* reqParams
: Parameters of the GATT procedure

union pOutActualCount

Public Members

uint8_t *pCount8b
: Pointer to a uint8_t memory location where to store the final number of added entries.

uint16_t *pCount16b
: Pointer to a uint16_t memory location where to store the final number of added entries.

union array

Public Members

gattService_t *aServices
: Array of services.

gattCharacteristic_t *aChars
: Array of characteristics.

gattAttribute_t *aDescriptors
: Array of descriptors.

uint8_t *aBytes
: Array of bytes.

uint16_t *aHandles
Array of handles.

union reqParams

Public Members

attReadByGroupTypeRequestParams_t rbgtParams
Read By Group Type Request Parameters.

attFindByTypeValueRequestParams_t fbtvParams
Find By Type Value Request Parameters.

attReadByTypeRequestParams_t rbtParams
Read By Type Request Parameters.

attFindInformationRequestParams_t fiParams
Find Information Request Parameters.

attReadRequestParams_t rParams
Read Request Parameters.

attReadBlobRequestParams_t rbParams
Read Blob Request Parameters.

attReadMultipleRequestParams_t rmParams
Read Multiple Request Parameters.

attVarWriteRequestAndCommandParams_t wParams
Write Request and Write Command Parameters for variable value length.

attSignedWriteCommandParams_t swParams
Signed Write Command parameters.

attPrepareWriteRequestResponseParams_t pwParams
Prepare Write Request and Prepare Write Response Parameters.

attExecuteWriteRequestParams_t ewParams
Execute Write Request Parameters

GATT Client

enum gattProcedureResult_t
GATT Client Procedure Result type
Values:

enumerator gGattProcSuccess_c

The procedure was completed successfully.

enumerator gGattProcError_c

The procedure was terminated due to an error.

typedef void (*gattClientProcedureCallback_t)(*deviceId_t* deviceId, *gattProcedureType_t* procedureType, *gattProcedureResult_t* procedureResult, *bleResult_t* error)

GATT Client Procedure Callback type

typedef void (*gattClientNotificationCallback_t)(*deviceId_t* deviceId, uint16_t characteristicValueHandle, uint8_t *aValue, uint16_t valueLength)

GATT Client Notification Callback prototype

typedef *gattClientNotificationCallback_t* gattClientIndicationCallback_t

GATT Client Indication Callback prototype

typedef void (*gattClientMultipleValueNotificationCallback_t)(*deviceId_t* deviceId, uint8_t *aHandleLenValue, uint32_t totalLength)

GATT Client Multiple Value Notification Callback prototype

typedef void (*gattClientEnhancedProcedureCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattProcedureType_t* procedureType, *gattProcedureResult_t* procedureResult, *bleResult_t* error)

GATT Client Enhanced Procedure Callback type

typedef void (*gattClientEnhancedNotificationCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t characteristicValueHandle, uint8_t *aValue, uint16_t valueLength)

GATT Client Enhanced Notification Callback prototype

typedef *gattClientEnhancedNotificationCallback_t* gattClientEnhancedIndicationCallback_t

GATT Client Enhanced Indication Callback prototype

typedef void (*gattClientEnhancedMultipleValueNotificationCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t *aHandleLenValue, uint32_t totalLength)

GATT Client Enhanced Multiple Value Notification Callback prototype

procDataStruct_t *pProcedureData[]

bleResult_t GattClient_Init(void)

Initializes the GATT Client functionality.

Remark

This should be called once at device startup, if necessary.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ResetProcedure(void)

Resets any ongoing GATT Client Procedure.

Remark

This function should be called if an ongoing Client procedure needs to be stopped.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterProcedureCallback(*gattClientProcedureCallback_t* callback)

Installs the application callback for the GATT Client module Procedures.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterNotificationCallback(*gattClientNotificationCallback_t* callback)

Installs the application callback for Server Notifications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterIndicationCallback(*gattClientIndicationCallback_t* callback)

Installs the application callback for Server Indications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterMultipleValueNotificationCallback(*gattClientMultipleValueNotificationCallback_t* callback)

Installs the application callback for Server Multiple Value Notification.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ExchangeMtu(*deviceId_t* deviceId, uint16_t mtu)

Initializes the MTU Exchange procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- mtu – **[in]** Desired MTU size.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.

- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllPrimaryServices(*deviceId_t* deviceId, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range and UUID.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverPrimaryServicesByUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery By UUID procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Service UUID type.
- pUuid – **[in]** Service UUID.

- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_FindIncludedServices(*deviceId_t* deviceId, *gattService_t* *pIoService, uint8_t maxServiceCount)

Initializes the Find Included Services procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoService – **[inout]** The service within which inclusions should be searched. The GATT module uses the Service's handle range and fills the included Services' handle ranges, UUID types and the UUIDs if they are 16-bit UUIDs.
- maxServiceCount – **[in]** Maximum number of included services to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllCharacteristicsOfService(*deviceId_t* deviceId, *gattService_t* *pIoService, uint8_t maxCharacteristicCount)

Initializes the Characteristic Discovery procedure for a given Service.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoService – **[inout]** The service within which characteristics should be searched. The GATT module uses the Characteristic's range.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverCharacteristicOfServiceByUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattService_t* *pService, *gattCharacteristic_t* *aOutCharacteristics, uint8_t maxCharacteristicCount, uint8_t *pOutDiscoveredCount)

Initializes the Characteristic Discovery procedure for a given Service, with a given UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pService – **[in]** The service within which characteristics should be searched.
- aOutCharacteristics – **[out]** The allocated array of Characteristics to be filled.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.
- pOutDiscoveredCount – **[out]** The actual number of characteristics discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllCharacteristicDescriptors(*deviceId_t* deviceId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t endingHandle, uint8_t maxDescriptorCount)

Initializes the Characteristic Descriptor Discovery procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback. The endingHandle parameter should be known by the application if Characteristic Discovery was performed, i.e., if the next Characteristic declaration handle is known, then subtract 1 to obtain the endingHandle for the current Characteristic. If the last handle of the Characteristic is still unknown, set the endingHandle parameter to 0xFFFF.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic within which descriptors should be searched. The GATT module uses the Characteristic's handle and fills each descriptor's handle and UUID.
- endingHandle – **[in]** The last handle of the Characteristic.
- maxDescriptorCount – **[in]** Maximum number of descriptors to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadCharacteristicValue(*deviceId_t* deviceId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t maxReadBytes)

Initializes the Characteristic Read procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic whose value must be read. The GATT module uses the value handle and fills the value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadUsingCharacteristicUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattHandleRange_t* *pHandleRange, uint8_t *aOutBuffer, uint16_t maxReadBytes, uint16_t *pOutActualReadBytes)

Initializes the Characteristic Read By UUID procedure.

Remark

This procedure returns the Characteristics found within the specified range with the specified UUID. aOutBuffer will contain the Handle-Value pair length (1 byte), then Handle-Value pairs for all Characteristic Values found with the specified UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pHandleRange – **[in]** Handle range for the search or NULL. If this is NULL, the search range is 0x0001-0xffff.
- aOutBuffer – **[out]** The allocated buffer to read into.
- maxReadBytes – **[in]** Maximum number of bytes to be read.
- pOutActualReadBytes – **[out]** The actual number of bytes read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadMultipleCharacteristicValues(*deviceId_t* deviceId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *aIoCharacteristics)

Initializes the Characteristic Read Multiple procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- aIoCharacteristics – **[inout]** Array of the characteristics whose values are to be read. The GATT module uses each Characteristic's value handle and maxValueLength fills each value and length.
- cNumCharacteristics – **[in]** Number of characteristics in the array.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_WriteCharacteristicValue(*deviceId_t* deviceId, const *gattCharacteristic_t* *pCharacteristic, uint16_t valueLength, const uint8_t *aValue, bool_t withoutResponse, bool_t signedWrite, bool_t doReliableLongCharWrites, const uint8_t *aCsrk)

Initializes the Characteristic Write procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pCharacteristic – **[in]** The characteristic whose value must be written. The GATT module uses the value handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.
- withoutResponse – **[in]** Indicates if a Write Command is used.
- signedWrite – **[in]** Indicates if a Signed Write is performed.
- doReliableLongCharWrites – **[in]** Indicates Reliable Long Writes.
- aCsrk – **[in]** The CSRK (gcCsrkSize_d bytes) if signedWrite is TRUE, ignored otherwise.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – TX queue for device is full.

- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadCharacteristicDescriptor(*deviceId_t* deviceId, *gattAttribute_t* *pIoDescriptor, uint16_t maxReadBytes)

Initializes the Characteristic Descriptor Read procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoDescriptor – **[inout]** The characteristic descriptor whose value must be read. The GATT module uses the attribute's handle and fills the attribute's value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_WriteCharacteristicDescriptor(*deviceId_t* deviceId, const *gattAttribute_t* *pDescriptor, uint16_t valueLength, const uint8_t *aValue)

Initializes the Characteristic Descriptor Write procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pDescriptor – **[in]** The characteristic descriptor whose value must be written. The GATT module uses the attribute's handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadMultipleVariableCharacteristicValues(*deviceId_t* deviceId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *pIoCharacteristics)

Initializes the Read Multiple Variable Length Characteristic Values procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- cNumCharacteristics – **[in]** Number of characteristics in the array.
- pIoCharacteristics – **[in]** Pointer to the array of the characteristics whose values are to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedProcedureCallback(*gattClientEnhancedProcedureCallback_t* callback)

Installs the application callback for the GATT Client module Procedures on Enhanced ATT bearers.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedNotificationCallback(*gattClientEnhancedNotificationCallback_t* callback)

Installs the application callback for Server Notifications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedIndicationCallback(*gattClientEnhancedIndicationCallback_t* callback)

Installs the application callback for Server Indications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedMultipleValueNotificationCallback(*gattClientEnhancedMultipleValueNotific* callback)

Installs the application callback for Server Multiple Value Notification.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllPrimaryServices(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range and UUID.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverPrimaryServicesByUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery By UUID procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Service UUID type.
- pUuid – **[in]** Service UUID.

- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedFindIncludedServices(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *pIoService, uint8_t maxServiceCount)

Initializes the Find Included Services procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoService – **[inout]** The service within which inclusions should be searched. The GATT module uses the Service's handle range and fills the included Services' handle ranges, UUID types and the UUIDs if they are 16-bit UUIDs.
- maxServiceCount – **[in]** Maximum number of included services to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllCharacteristicsOfService(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *pIoService, uint8_t maxCharacteristicCount)

Initializes the Characteristic Discovery procedure for a given Service.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoService – **[inout]** The service within which characteristics should be searched. The GATT module uses the Characteristic's range.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverCharacteristicOfServiceByUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattService_t* *pService, *gattCharacteristic_t* *aOutCharacteristics, uint8_t maxCharacteristicCount, uint8_t *pOutDiscoveredCount)

Initializes the Characteristic Discovery procedure for a given Service, with a given UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pService – **[in]** The service within which characteristics should be searched.
- aOutCharacteristics – **[out]** The allocated array of Characteristics to be filled.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

- pOutDiscoveredCount – **[out]** The actual number of characteristics discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllCharacteristicDescriptors(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t endingHandle, uint8_t maxDescriptorCount)

Initializes the Characteristic Descriptor Discovery procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback. The endingHandle parameter should be known by the application if Characteristic Discovery was performed, i.e., if the next Characteristic declaration handle is known, then subtract 1 to obtain the endingHandle for the current Characteristic. If the last handle of the Characteristic is still unknown, set the endingHandle parameter to 0xFFFF.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic within which descriptors should be searched. The GATT module uses the Characteristic's handle and fills each descriptor's handle and UUID.
- endingHandle – **[in]** The last handle of the Characteristic.
- maxDescriptorCount – **[in]** Maximum number of descriptors to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadCharacteristicValue(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t maxReadBytes)

Initializes the Characteristic Read procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic whose value must be read. The GATT module uses the value handle and fills the value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadUsingCharacteristicUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattHandleRange_t* *pHandleRange, uint8_t *aOutBuffer, uint16_t maxReadBytes, uint16_t *pOutActualReadBytes)

Initializes the Characteristic Read By UUID procedure.

Remark

This procedure returns the Characteristics found within the specified range with the specified UUID. aOutBuffer will contain the Handle-Value pair length (1 byte), then Handle-Value pairs for all Characteristic Values found with the specified UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pHandleRange – **[in]** Handle range for the search or NULL. If this is NULL, the search range is 0x0001-0xffff.

- aOutBuffer – **[out]** The allocated buffer to read into.
- maxReadBytes – **[in]** Maximum number of bytes to be read.
- pOutActualReadBytes – **[out]** The actual number of bytes read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadMultipleCharacteristicValues(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *aIoCharacteristics)

Initializes the Characteristic Read Multiple procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- aIoCharacteristics – **[inout]** Array of the characteristics whose values are to be read. The GATT module uses each Characteristic's value handle and maxValueLength fills each value and length.
- cNumCharacteristics – **[in]** Number of characteristics in the array.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedWriteCharacteristicValue(*deviceId_t* deviceId, *bearerId_t* bearerId, const *gattCharacteristic_t* *pCharacteristic, uint16_t valueLength, const uint8_t *aValue, bool_t withoutResponse, bool_t doReliableLongCharWrites, const uint8_t *aCsrk)

Initializes the Characteristic Write procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pCharacteristic – **[in]** The characteristic whose value must be written. The GATT module uses the value handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.
- withoutResponse – **[in]** Indicates if a Write Command is used.
- signedWrite – **[in]** Indicates if a Signed Write is performed.
- doReliableLongCharWrites – **[in]** Indicates Reliable Long Writes.
- aCsrk – **[in]** The CSRK (gcCsrkSize_d bytes) if signedWrite is TRUE, ignored otherwise.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadCharacteristicDescriptor(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattAttribute_t* *pIoDescriptor, uint16_t maxReadBytes)

Initializes the Characteristic Descriptor Read procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoDescriptor – **[inout]** The characteristic descriptor whose value must be read. The GATT module uses the attribute's handle and fills the attribute's value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedWriteCharacteristicDescriptor(*deviceId_t* deviceId, *bearerId_t* bearerId, const *gattAttribute_t* *pDescriptor, uint16_t valueLength, const uint8_t *aValue)

Initializes the Characteristic Descriptor Write procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pDescriptor – **[in]** The characteristic descriptor whose value must be written. The GATT module uses the attribute's handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadMultipleVariableCharacteristicValues(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *pIoCharacteristics)

Initializes the Read Multiple Variable Length Characteristic Values procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- cNumCharacteristics – **[in]** Number of characteristics in the array.
- pIoCharacteristics – **[in]** Pointer to the array of the characteristics whose values are to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GattClient_SimpleCharacteristicWrite(deviceId, pChar, valueLength, aValue)

Executes the basic Characteristic Write operation (with server confirmation).

Parameters

- deviceId – **[in]** Device ID of the connected GATT Server.
- pChar – **[in]** Pointer to the Characteristic being written.
- valueLength – **[in]** Size in bytes of the value to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GattClient_CharacteristicWriteWithoutResponse(deviceId, pChar, valueLength, aValue)

Executes the Characteristic Write Without Response operation.

Parameters

- deviceId – **[in]** Device ID of the connected GATT Server.
- pChar – **[in]** Pointer to the Characteristic being written.
- valueLength – **[in]** Size in bytes of the value to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GattClient_CharacteristicSignedWrite(deviceId, pChar, valueLength, aValue, aCsrk)

Executes the Characteristic Signed Write Without Response operation.

Parameters

- deviceId – **[in]** Device ID of the connected GATT Server.
- pChar – **[in]** Pointer to the Characteristic being written.
- valueLength – **[in]** Size in bytes of the value to be written.
- aValue – **[in]** Array of bytes to be written.
- aCsrk – **[in]** CSRK to be used for data signing.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GATT Server

enum gattServerEventType_t

GATT Server Event type enumeration

Values:

enumerator gEvtMtuChanged_c

ATT_MTU was changed after the MTU exchange.

enumerator gEvtHandleValueConfirmation_c

Received a Handle Value Confirmation from the Client.

enumerator gEvtAttributeWritten_c

An attribute registered with GattServer_RegisterHandlesForWriteNotifications was written. After receiving this event, application must call GattServer_SendAttributeWrittenStatus. Application must write the Attribute in the Database if it considers necessary.

enumerator gEvtCharacteristicCccdWritten_c

A CCCD was written. Application should save the CCCD value with Gap_SaveCccd.

enumerator gEvtAttributeWrittenWithoutResponse_c

An attribute registered with GattServer_RegisterHandlesForWriteNotifications was written without response (with ATT Write Command). Application must write the Attribute Value in the Database if it considers necessary.

enumerator gEvtError_c

An error appeared during a Server-initiated procedure.

enumerator gEvtLongCharacteristicWritten_c

A long characteristic was written.

enumerator gEvtAttributeRead_c

An attribute registered with GattServer_RegisterHandlesForReadNotifications is being read. After receiving this event, application must call GattServer_SendAttributeReadStatus.

enumerator gEvtInvalidPduReceived_c

An invalid PDU was received from Client. Application decides if disconnection is required

enum gattServerProcedureType_t

Server-initiated procedure type enumeration

Values:

enumerator gSendAttributeWrittenStatus_c

Procedure initiated by GattServer_SendAttributeWrittenStatus.

enumerator gSendAttributeReadStatus_c

Procedure initiated by GattServer_SendAttributeReadStatus.

enumerator gSendNotification_c

Procedure initiated by GattServer_SendNotification.

enumerator gSendIndication_c

Procedure initiated by GattServer_SendIndication.

enumerator gSendMultipleValNotification_c

Procedure initiated by GattServer_SendMultipleHandleValueNotification.

typedef void (*gattServerCallback_t)(*deviceId_t* deviceId, *gattServerEvent_t* *pServerEvent)

GATT Server Callback prototype

typedef void (*gattServerEnhancedCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattServerEvent_t* *pServerEvent)

GATT Server Enhanced Callback prototype

bleResult_t GattServer_Init(void)

Initializes the GATT Server module.

Remark

Application does not need to call this function if *Gatt_Init()* is called.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterCallback(*gattServerCallback_t* callback)

Installs an application callback for the GATT Server module.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application-defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterHandlesForWriteNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Registers the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to modify the attributes' values.

Remark

The application is responsible for actually writing the new requested values in the GATT database. Service and profile-specific control-point characteristics should have their value handles in this list so that the application may get notified when a GATT Client writes it.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – The maximum number of supported handles for write notifications has been reached.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_UnregisterHandlesForWriteNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Unregisters the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to modify the attributes' values.

Remark

To unregister all the list, pass 0 count and NULL.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendAttributeWrittenStatus(*deviceId_t* deviceId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute write operation.

Remark

This function must be called by the application when receiving the gEvtAttributeWritten_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was written.
- status – **[in]** The status of the write operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Write Response will be sent to the peer. Otherwise, an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterHandlesForReadNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Registers the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to read the attributes' values.

Remark

The application may modify the attribute's value in the GATT Database before sending the response with GattServer_SendAttributeReadStatus.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – The maximum number of supported handles for read notifications has been reached.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_UnregisterHandlesForReadNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Unregisters the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to read the attributes' values.

Remark

To unregister all the list, pass 0 count and NULL.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendAttributeReadStatus(*deviceId_t* deviceId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute read operation.

Remark

This function must be called by the application when receiving the gEvtAttributeRead_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was being read.
- status – **[in]** The status of the read operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Read Response will be sent to the peer containing the attribute value from the GATT Database. Otherwise, an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendNotification(*deviceId_t* deviceId, uint16_t handle)

Sends a notification to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendIndication(*deviceId_t* deviceId, uint16_t handle)

Sends an indication to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.

- handle – **[in]** Handle of the Value of the Characteristic to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendInstantValueNotification(*deviceId_t* deviceId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.
- valueLength – **[in]** Length of data to be notified.
- aValue – **[in]** Data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendMultipleHandleValueNotification(*deviceId_t* deviceId, uint32_t totalLength, const uint8_t *pHandleLengthValueList)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- totalLength – **[in]** Length of the handle, value, value length tuples.
- pHandleLengthValueList – **[in]** Pointer to data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendInstantValueIndication(*deviceId_t* deviceId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends an indication to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.
- valueLength – **[in]** Length of data to be indicated.
- aValue – **[in]** Data to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – TX queue for device is full.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterUniqueHandlesForNotifications(bool_t bWrite, bool_t bRead)

Registers all GATT DB dynamic attribute handles with unique value buffers to be notified through the GATT Server callback when a GATT Client attempts to read/write the attributes' values.

Remark

This function executes synchronously.

Remark

This function should be called when adding GATT DB unique value buffer characteristics or descriptors.

Parameters

- bWrite – **[in]** Enables/Disables write notifications.
- bRead – **[in]** Enables/Disables read notifications.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterEnhancedCallback(*gattServerEnhancedCallback_t* callback)

Installs an application callback for the Enhanced ATT GATT Server module.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application-defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendAttributeWrittenStatus(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute write operation.

Remark

This function must be called by the application when receiving the gEvtAttributeWritten_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was written.
- status – **[in]** The status of the write operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Write Response will be sent to the peer. Else an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendAttributeReadStatus(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute read operation.

Remark

This function must be called by the application when receiving the gEvtAttributeRead_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was being read.

- status – **[in]** The status of the read operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Read Response will be sent to the peer containing the attribute value from the GATT Database. Else an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle)

Sends a notification to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendIndication(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle)

Sends an indication to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendInstantValueNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.
- valueLength – **[in]** Length of data to be notified.
- aValue – **[in]** Data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendMultipleHandleValueNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint32_t totalLength, const uint8_t *pHandleLengthValueList)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- totalLength – **[in]** Length of the handle, value, value length tuples.
- pHandleLengthValueList – **[in]** Pointer to data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendInstantValueIndication(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends an indication to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.
- valueLength – **[in]** Length of data to be indicated.
- aValue – **[in]** Data to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

struct gattServerMtuChangedEvent_t

#include <gatt_server_interface.h> GATT Server MTU Changed Event structure

Public Members

uint16_t newMtu

Value of the agreed ATT_MTU for this connection.

struct gattServerAttributeWrittenEvent_t

#include <gatt_server_interface.h> GATT Server Attribute Written Event structure

Public Members

uint16_t handle

Handle of the attribute.

uint16_t cValueLength

Length of the attribute value array.

uint8_t *aValue

Attribute value array attempted to be written.

bearerId_t bearerId

Used by EATT. Send response on the same bearer. For ATT value is 0.

struct gattServerLongCharacteristicWrittenEvent_t

#include <gatt_server_interface.h> GATT Server Long Characteristic Written Event structure

Public Members

uint16_t handle

Handle of the Characteristic Value.

uint16_t cValueLength

Length of the value written.

uint8_t *aValue
Pointer to the attribute value in the database.

struct gattServerCccdWrittenEvent_t
#include <gatt_server_interface.h> GATT Server CCCD Written Event structure

Public Members

uint16_t handle
Handle of the CCCD attribute.

gattCccdFlags_t newCccd
New value of the CCCD.

struct gattServerAttributeReadEvent_t
#include <gatt_server_interface.h> GATT Server Attribute Read Event structure

Public Members

uint16_t handle
Handle of the attribute.

struct gattServerProcedureError_t
#include <gatt_server_interface.h> Server-initiated procedure error structure

Public Members

gattServerProcedureType_t procedureType
Procedure that generated error.

bleResult_t error
Error generated.

struct gattServerInvalidPdu_t
#include <gatt_server_interface.h> ATT PDU that generated the error

Public Members

attOpcode_t attOpCode
The invalid ATT op code that generated the error

struct gattServerEvent_t
#include <gatt_server_interface.h> GATT Server Event structure: type + data.

Public Members

gattServerEventType_t eventType
Event type.

union *gattServerEvent_t* eventData
Event data : selected according to event type.

union eventData

Public Members

gattServerMtuChangedEvent_t mtuChangedEvent
For event type gEvtMtuChanged_c: the new value of the ATT_MTU.

gattServerAttributeWrittenEvent_t attributeWrittenEvent
For event types gEvtAttributeWritten_c, gEvtAttributeWrittenWithoutResponse_c: handle and value of the attempted write.

gattServerCccdWrittenEvent_t charCccdWrittenEvent
For event type gEvtCharacteristicCccdWritten_c: handle and value of the CCCD.

gattServerProcedureError_t procedureError
For event type gEvtError_c: error that terminated a Server-initiated procedure.

gattServerLongCharacteristicWrittenEvent_t longCharWrittenEvent
For event type gEvtLongCharacteristicWritten_c: handle and value.

gattServerAttributeReadEvent_t attributeReadEvent
For event types gEvtAttributeRead_c: handle of the attempted read.

gattServerInvalidPdu_t attributeOpCode
For event type gEvtInvalidPduReceived_c: the ATT PDU that generated the error

GATT Database

enum gattCharacteristicPropertiesBitFields_tag
Values:

enumerator gGattCharPropNone_c
No Properties selected.

enumerator gGattCharPropBroadcast_c
Characteristic can be broadcast.

enumerator gGattCharPropRead_c
Characteristic can be read.

enumerator gGattCharPropWriteWithoutRsp_c
Characteristic can be written without response.

enumerator gGattCharPropWrite_c
Characteristic can be written with response.

enumerator gGattCharPropNotify_c
Characteristic can be notified.

enumerator gGattCharPropIndicate_c
Characteristic can be indicated.

enumerator gGattCharPropAuthSignedWrites_c
Characteristic can be written with signed data.

enumerator gGattCharPropExtendedProperties_c
Extended Characteristic properties.

enum gattDbAccessType_t
Attribute access type
Values:

enumerator gAccessRead_c

enumerator gAccessWrite_c

enumerator gAccessNotify_c

typedef uint8_t gattCharacteristicPropertiesBitFields_t
Bit fields for Characteristic properties

typedef uint8_t gattAttributePermissionsBitFields_t
Bit fields for attribute permissions

uint16_t gGattDbAttributeCount_c
The number of attributes in the GATT Database.

gattDbAttribute_t *gattDatabase
Reference to the GATT database

uint16_t mServerServiceChangedCharHandle

uint16_t mServerServiceChangedCCCDHandle

uint32_t gGattDynamicAttrSize

uint8_t gaGattDynamicAttrBlob[]

uint32_t gGattDynamicValSize

uint8_t gaGattDynamicValBlob[]

uint16_t GattDb_GetIndexOfHandle(uint16_t handle)

Returns the database index for a given attribute handle.

Parameters

- handle – **[in]** The attribute handle.

Returns

The index of the given attribute in the database or gGattDbInvalidHandleIndex_d.

uint16_t GattDb_ServiceStartHandle(uint16_t handle)

Returns the handle of the service to which the given attribute belongs.

Parameters

- handle – **[in]** The attribute handle.

Returns

The handle of the service or gGattDbInvalidHandleIndex_d.

bleResult_t GattDb_FindServiceRange(uint16_t serviceHandle, uint16_t *pOutStartIndex, uint16_t *pOutAttributeCount)

Finds the start index and attribute count for a given service.

Parameters

- serviceHandle – **[in]** The service handle.
- pOutStartIndex – **[in]** The index in the database where the service declaration begins.
- pOutAttributeCount – **[in]** The number of attributes contained by the service.

Return values

- gBleSuccess_c –
- gGattDbInvalidHandleIndex_d – Service does not exist in gatt database.

uint16_t GattDb_GetAttributeValueSize(uint16_t handle)

Returns the value length for a given attribute handle.

Parameters

- handle – **[in]** The attribute handle.

Returns

The length of the value of the attribute at the given handle or maxVariableValueLength in case of variable length attributes.

bleResult_t GattDb_ComputeDatabaseHash(void)

Computes the database hash for a static or a dynamic database.

Parameters

- void. –

Return values

- gBleSuccess_c –
- gBleInvalidState_c – Database not initialized or empty.

- gBleOutOfMemory_c – Could not allocate memory for the database content used to compute the database hash.
- gGattDbInvalidHandleIndex_d – Service does not exist in gatt database.

bleResult_t GattDb_Init(void)

Initializes the GATT database at runtime.

Remark

This function should be called only once at device start-up. In the current stack implementation, it is called internally by Ble_HostInitialize.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Could not allocate memory for the GATT dynamic database.
- gBleAlreadyInitialized_c – GATT database already initialized.

bleResult_t GattDb_Deinit(void)

Function performing runtime deinitialization of the GATT database.

Remark

This function has effect for dynamic data base only .

Parameters

- none. –

Return values

- gBleSuccess_c –
- gBleInvalidState_c – GATT dynamic database not initialized.

bleResult_t GattDb_WriteAttribute(uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Writes an attribute from the application level.

This function can be called by the application code to modify an attribute in the database. It should only be used by the application to modify a Characteristic's value based on the application logic (e.g., external sensor readings).

Remark

This function executes synchronously.

Parameters

- handle – **[in]** The handle of the attribute to be written.
- valueLength – **[in]** The number of bytes to be written.
- aValue – **[in]** The source buffer containing the value to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – aValue NULL and length higher than 0.
- gAttStatusBase_c – + gAttErrCodeInvalidHandle_c handle not found in GATT database.
- gGattInvalidValueLength_c – valueLength not equal to attribute value length, or higher than the maximum length for variable size attributes.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_ReadAttribute(uint16_t handle, uint16_t maxBytes, uint8_t *aOutValue, uint16_t *pOutValueLength)

Reads an attribute from the application level.

This function can be called by the application code to read an attribute in the database.

Remark

This function executes synchronously.

Parameters

- handle – **[in]** The handle of the attribute to be read.
- maxBytes – **[in]** The maximum number of bytes to be received.
- aOutValue – **[out]** The pre-allocated buffer ready to receive the bytes.
- pOutValueLength – **[out]** The actual number of bytes received.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – aOutValue NULL, pOutValueLength NULL or maxBytes 0.
- gAttStatusBase_c – + gAttErrCodeInvalidHandle_c handle not found in GATT database.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindServiceHandle(uint16_t startHandle, *bleUuidType_t* serviceUuidType, const *bleUuid_t* *pServiceUuid, uint16_t *pOutServiceHandle)

Finds the handle of a Service Declaration with a given UUID inside the database.

Remark

This function executes synchronously.

Remark

The startHandle should be set to 0x0001 when this function is called for the first time. If multiple Services with the same UUID are expected, then after the first successful call the function may be called again with the startHandle equal to the found service handle plus one.

Parameters

- startHandle – **[in]** The handle to start the search. Should be 0x0001 on the first call.
- serviceUuidType – **[in]** Service UUID type.
- pServiceUuid – **[in]** Service UUID.
- pOutServiceHandle – **[out]** Pointer to the service declaration handle to be written.

Return values

- gBleSuccess_c – Service Declaration found, handle written in pOutCharValueHandle.
- gGattDbInvalidHandle_c – Invalid Start Handle.
- gBleInvalidParameter_c – pServiceUuid or pOutServiceHandle NULL.
- gGattDbServiceNotFound_c – Service with given UUID not found.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindCharValueHandleInService(uint16_t serviceHandle, *bleUuidType_t* characteristicUuidType, const *bleUuid_t* *pCharacteristicUuid, uint16_t *pOutCharValueHandle)

Finds the handle of a Characteristic Value with a given UUID inside a Service.

The Service is input by its declaration handle.

Remark

This function executes synchronously.

Parameters

- serviceHandle – **[in]** The handle of the Service declaration.
- characteristicUuidType – **[in]** Characteristic UUID type.
- pCharacteristicUuid – **[in]** Characteristic UUID.
- pOutCharValueHandle – **[out]** Pointer to the characteristic value handle to be written.

Return values

- gBleSuccess_c – Characteristic Value found, handle written in pOutCharValueHandle.
- gGattDbInvalidHandle_c – Handle not found or not a Service declaration.

- gBleInvalidParameter_c – pCharacteristicUuid or pOutCharValueHandle NULL.
- gGattDbCharacteristicNotFound_c – Characteristic Value with given UUID not found.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindCccdHandleForCharValueHandle(uint16_t charValueHandle, uint16_t *pOutCccdHandle)

Finds the handle of a Characteristic's CCCD given the Characteristic's Value handle.

Remark

This function executes synchronously.

Parameters

- charValueHandle – **[in]** The handle of the Service declaration.
- pOutCccdHandle – **[out]** Pointer to the CCCD handle to be written.

Return values

- gBleSuccess_c – CCCD found, handle written in pOutCccdHandle.
- gBleInvalidParameter_c – pOutCccdHandle is NULL.
- gGattDbInvalidHandle_c – Invalid Characteristic Value handle.
- gGattDbCccdNotFound_c – CCCD not found for this Characteristic.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindDescriptorHandleForCharValueHandle(uint16_t charValueHandle, *bleUuidType_t* descriptorUuidType, const *bleUuid_t* *pDescriptorUuid, uint16_t *pOutDescriptorHandle)

Finds the handle of a Characteristic Descriptor given the Characteristic's Value handle and Descriptor's UUID.

Remark

This function executes synchronously.

Parameters

- charValueHandle – **[in]** The handle of the Service declaration.
- descriptorUuidType – **[in]** Descriptor's UUID type.
- pDescriptorUuid – **[in]** Descriptor's UUID.
- pOutDescriptorHandle – **[out]** Pointer to the Descriptor handle to be written.

Return values

- gBleSuccess_c – Descriptor found.

- gBleInvalidParameter_c – pDescriptorUuid or pOutDescriptorHandle are NULL.
- gGattDbInvalidHandle_c – Invalid Characteristic Value handle.
- gGattDbDescriptorNotFound_c – Descriptor not found for this Characteristic.
- gBleFeatureNotSupported_c – GATT Server not enabled.

gGattDbInvalidHandleIndex_d

Special value returned by GattDb_GetIndexOfHandle to signal that an invalid attribute handle was given as parameter.

gGattDbInvalidHandle_d

Special value used to mark an invalid attribute handle. Attribute handles are strictly positive.

gPermissionNone_c

No permissions selected.

gPermissionFlagReadable_c

Attribute can be read.

gPermissionFlagReadWithEncryption_c

Attribute may be read only if link is encrypted.

gPermissionFlagReadWithAuthentication_c

Attribute may be read only by authenticated peers.

gPermissionFlagReadWithAuthorization_c

Attribute may be read only by authorized peers.

gPermissionFlagWritable_c

Attribute can be written.

gPermissionFlagWriteWithEncryption_c

Attribute may be written only if link is encrypted.

gPermissionFlagWriteWithAuthentication_c

Attribute may be written only by authenticated peers.

gPermissionFlagWriteWithAuthorization_c

Attribute may be written only by authorized peers.

gGattDatabaseHashSize_c

struct gattDbAttribute_t

#include <gatt_database.h> Attribute structure

Public Members

uint16_t handle

The attribute handle - cannot be 0x0000. The attribute handles need not be consecutive, but must be strictly increasing.

uint16_t permissions

Attribute permissions as defined by the ATT.

uint32_t uuid

The UUID should be read according to the *gattDbAttribute_t.uuidType* member: for 2-byte and 4-byte UUIDs, this contains the value of the UUID; for 16-byte UUIDs, this is a pointer to the allocated 16-byte array containing the UUID.

uint8_t *pValue

A pointer to allocated value array.

uint16_t valueLength

The size of the value array.

uint16_t uuidType

Identifies the length of the UUID; values interpreted according to the *bleUuidType_t* enumeration.

uint16_t maxVariableValueLength

The maximum length of the attribute value array; if this is set to 0, then the attribute's length is fixed and cannot be changed.

Bluetooth Low Energy

enum bleResult_tag

BLE result type - the return value of BLE API functions

Values:

enumerator gBleStatusBase_c

General status base.

enumerator gBleSuccess_c

Function executed successfully.

enumerator gBleInvalidParameter_c

Parameter has an invalid value or is outside the accepted range.

enumerator gBleOverflow_c

An internal limit is reached.

enumerator gBleUnavailable_c

A requested parameter is not available.

enumerator gBleFeatureNotSupported_c
The requested feature is not supported by this stack version.

enumerator gBleOutOfMemory_c
An internal memory allocation failed.

enumerator gBleAlreadyInitialized_c
Ble_HostInitialize function is incorrectly called a second time.

enumerator gBleOsError_c
An error occurred at the OS level.

enumerator gBleUnexpectedError_c
A “should never get here”-type error occurred.

enumerator gBleInvalidState_c
The requested API cannot be called in the current state.

enumerator gBleTimerError_c
Timer allocation failed.

enumerator gBleReassemblyInProgress_c
HCI Packet reassembly was in progress. The old packet was discarded

enumerator gBleNVMError_c
An error occurred when performing an NVM operation

enumerator gBleRngError_c
error generated by a RNG function call e.g. RNG_GetPseudoRandomNo

enumerator gBleSecLibError_c
error generated by a SecLib function call e.g. SecLib_VerifyBluetoothAh

enumerator gHciStatusBase_c
HCI status base.

enumerator gHciSuccess_c
Alias.

enumerator gHciUnknownHciCommand_c
The HCI Command packet opcode is unknown.

enumerator gHciUnknownConnectionIdentifier_c
The connection does not exist or it is of a wrong type.

enumerator gHciHardwareFailure_c
A failure occurred in the Controller.

enumerator gHciPageTimeout_c

A page timed out based on the Page Timeout parameter.

enumerator gHciAuthenticationFailure_c

Pairing or authentication failed. Possible causes: incorrect PIN or Link Key.

enumerator gHciPinOrKeyMissing_c

Pairing failed because of missing PIN or authentication failed because of missing Key.

enumerator gHciMemoryCapacityExceeded_c

The controller doesn't have enough memory to store new parameters.

enumerator gHciConnectionTimeout_c

The link supervision timeout has expired for a connection or the synchronization timeout has expired for a broadcast.

enumerator gHciConnectionLimitExceeded_c

The maximum number of connections has already been reached.

enumerator gHciSynchronousConnectionLimitToADeviceExceeded_c

The maximum number of synchronous connections has already been reached.

enumerator gHciAclConnectionAlreadyExists_c

A connection to this device already exists.

enumerator gHciCommandDisallowed_c

The controller can't process the command at this time.

enumerator gHciConnectionRejectedDueToLimitedResources_c

The connection was rejected due to limited resources.

enumerator gHciConnectionRejectedDueToSecurityReasons_c

The connection was rejected because the security requirements were not fulfilled.

enumerator gHciConnectionRejectedDueToUnacceptableBdAddr_c

The connection was rejected because the device does not accept the given BD_ADDR. A possible cause is that the device only accepts connections from specific BD_ADDRs.

enumerator gHciConnectionAcceptTimeoutExceeded_c

The connection accept timeout for this connection has been exceeded.

enumerator gHciUnsupportedFeatureOrParameterValue_c

The feature or parameter in the HCI command is not supported.

enumerator gHciInvalidHciCommandParameters_c

At least one of the HCI command parameters is invalid.

enumerator gHciRemoteUserTerminatedConnection_c

The remote device's user terminated the connection or stopped broadcasting.

enumerator gHciRemoteDeviceTerminatedConnectionLowResources_c

The remote device terminated the connection due to low resources.

enumerator gHciRemoteDeviceTerminatedConnectionPowerOff_c

The remote device terminated the connection because it will power off.

enumerator gHciConnectionTerminatedByLocalHost_c

The local device terminated the connection, synchronization with a broadcaster or stopped broadcasting.

enumerator gHciRepeatedAttempts_c

The Controller disallows the authentication or pairing because not enough time has passed since the failed attempt.

enumerator gHciPairingNotAllowed_c

The device does not allow pairing.

enumerator gHciUnknownLpmPdu_c

The Controller has received an unknown LMP code.

enumerator gHciUnsupportedRemoteFeature_c

The remote device doesn't support the feature associated with the issued command, LMP PDU or LL Control PDU.

enumerator gHciScoOffsetRejected_c

The offset requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciScoIntervalRejected_c

The interval requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciScoAirModeRejected_c

The air mode requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciInvalidLpmParameters_c

Some LMP PDU or LL Control PDU parameters were invalid.

enumerator gHciUnspecifiedError_c

No other error code was appropriate.

enumerator gHciUnsupportedLpmParameterValue_c

An LMP PDU or LL Control PDU contains at least one parameter value not supported by the Controller at this time.

enumerator gHciRoleChangeNotAllowed_c

The Controller doesn't allow a role change at this time.

enumerator gHciLLResponseTimeout_c

An LMP transaction failed to respond within the LMP response timeout or an LL transaction failed to respond within the LL response timeout.

enumerator gHciLmpErrorTransactionCollision_c

An LMP transaction or LL procedure has collided with the same transaction or procedure which is already in progress.

enumerator gHciLmpPduNotAllowed_c

A controller sent an LMP PDU with an opcode that isn't allowed.

enumerator gHciEncryptionModeNotAcceptable_c

The requested encryption mode is not allowed at this time.

enumerator gHciLinkKeyCannotBeChanged_c

The link key can't be changed because a fixed unit key is being used.

enumerator gHciRequestedQosNotSupported_c

The requested QoS is not supported.

enumerator gHciInstantPassed_c

An LMP PDU or LL PDU that includes an instant cannot be performed because the instant when this would have occurred has passed.

enumerator gHciPairingWithUnitKeyNotSupported_c

It wasn't possible to pair because a unit key was requested but it is not supported.

enumerator gHciDifferentTransactionCollision_c

An LMP transaction or LL Procedure was started that collides with another one.

enumerator gHciReserved_0x2B_c

enumerator gHciQosNotAcceptableParameter_c

The specified QoS parameters couldn't be accepted at this time.

enumerator gHciQosRejected_c

The specified QoS parameters couldn't be accepted and QoS negotiation should be terminated.

enumerator gHciChannelClassificationNotSupported_c

The Controller can't perform a channel assessment because it isn't supported.

enumerator gHciInsufficientSecurity_c

The HCI command or LMP PDU sent is only possible on an encrypted link.

enumerator gHciParameterOutOfMandatoryRange_c

A parameter value requested is outside the mandatory range of parameters for the given HCI command or LMP PDU. The recipient does not accept that value.

enumerator gHciReserved_0x31_c

enumerator gHciRoleSwitchPending_c

A role switch is pending.

enumerator gHciReserved_0x33_c

enumerator gHciReservedSlotViolation_c

The Synchronous negotiation was terminated with its state set to Reserved Slot Violation.

enumerator gHciRoleSwitchFailed_c

A role switch was attempted but it failed. The original piconet structure is restored.

enumerator gHciExtendedInquiryResponseTooLarge_c

The extended inquiry response, with the requested requirements for FEC, is too large to fit in any of the packet types supported by the Controller.

enumerator gHciSecureSimplePairingNotSupportedByHost_c

The IO capabilities request / response was rejected because the sending Host does not support Secure Simple Pairing even though the receiving Link Manager does.

enumerator gHciHostBusyPairing_c

The Host is busy with another pairing request. Pairing should be retried again later.

enumerator gHciConnectionRejectedDueToNoSuitableChannelFound_c

An appropriate value for the Channel selection operation couldn't be calculated by the Controller.

enumerator gHciControllerBusy_c

The operation was rejected because the Controller was busy.

enumerator gHciUnacceptableConnectionParameters_c

The remote device terminated the connection or rejected a request because of at least one unacceptable connection parameter.

enumerator gHciDirectedAdvertisingTimeout_c

Advertising for a fixed duration completed. For directed advertising, advertising completed without a connection being created.

enumerator gHciConnectionTerminatedDueToMicFailure_c

The connection or the synchronization was terminated because the Message Integrity Check failed.

enumerator gHciConnectionFailedToBeEstablishedOrSyncTimeout_c

The LL initiated a connection / synchronization to periodic advertising but the connection has failed to be established or the LL failed to synchronize with the periodic advertising within 6 periodic advertising events of the first attempt.

enumerator gHciMacConnectionFailed_c

enumerator gHciCoarseClockAdjustmentRejected_c

The Central is unable to make a coarse adjustment to the piconet clock, using the given parameters. It will attempt to move the clock using clock dragging.

enumerator gHciType0SubmapNotDefined_c

The LMP PDU is rejected because the Type 0 submap is not currently defined.

enumerator gHciUnknownAdvertisingIdentifier_c

A command was sent from the Host that should identify an Advertising or Sync handle, but that handle doesn't exist.

enumerator gHciLimitReached_c

The number of operations requested has been reached.

enumerator gHciOperationCancelledByHost_c

A request sent from the Host to the Controller which was still pending has been successfully canceled.

enumerator gHciPacketTooLong_c

An attempt was made to send or receive a packet that exceeds the maximum allowed packet length.

enumerator gHciPacketTooLate_c

Information was provided too late to the Controller.

enumerator gHciPacketTooEarly_c

Information was provided too early to the Controller.

enumerator gHciAlreadyInit_c

HCI has already been initialized.

enumerator gHciInvalidParameter_c

At least one of the HCI command parameters is invalid.

enumerator gHciCallbackNotInstalled_c

Callback was not installed.

enumerator gHciCallbackAlreadyInstalled_c

Callback has already been installed.

enumerator gHciCommandNotSupported_c

The HCI command is not supported.

enumerator gHciEventNotSupported_c

The HCI event is not supported.

enumerator gHciTransportError_c

HCI layer initialization failure. HCI data buffering semaphore allocation error. HCI layer received an unknown packet type. Serial initialization or write failure.

enumerator gCtrlStatusBase_c

Controller status base.

enumerator gCtrlSuccess_c
Alias.

enumerator gL2caStatusBase_c
L2CAP status base.

enumerator gL2caSuccess_c
Alias.

enumerator gL2caAlreadyInit_c
L2CAP has already been initialized.

enumerator gL2caInsufficientResources_c
L2CAP could not allocate resources to perform operations (memory or timers).

enumerator gL2caCallbackNotInstalled_c
Callback was not installed.

enumerator gL2caCallbackAlreadyInstalled_c
Callback has already been installed.

enumerator gL2caLePsmInvalid_c
Invalid LE_PSM value.

enumerator gL2caLePsmAlreadyRegistered_c
LE_PSM has already been registered.

enumerator gL2caLePsmNotRegistered_c
LE_PSM value was not registered.

enumerator gL2caLePsmInsufficientResources_c
No free LE_PSM registration slot was found.

enumerator gL2caChannelInvalid_c
L2CA channel is invalid.

enumerator gL2caChannelClosed_c
L2CA channel is closed.

enumerator gL2caChannelAlreadyConnected_c
L2CA channel is already connected.

enumerator gL2caConnectionParametersRejected_c
Connection parameters were rejected.

enumerator gL2caChannelBusy_c
L2CA channel is busy.

enumerator gL2caInvalidParameter_c

The command contains at least one invalid parameter.

enumerator gL2caInternalError_c

There are connected L2CAP channels for the LE_PSM.

enumerator gSmStatusBase_c

Security Manager status base.

enumerator gSmSuccess_c

Alias.

enumerator gSmNullCBFunction_c

enumerator gSmCommandNotSupported_c

The Security Manager (SM) does not have the required features or version to support this command

enumerator gSmUnexpectedCommand_c

This command is not or cannot be handled in the current context of the SM.

enumerator gSmInvalidCommandCode_c

The provided SM command code is invalid.

enumerator gSmInvalidCommandLength_c

The provided command length is not valid for the SM command code.

enumerator gSmInvalidCommandParameter_c

One of the parameters of the SM command is not valid.

enumerator gSmInvalidDeviceId_c

The provided Device ID is invalid.

enumerator gSmInvalidInternalOperation_c

There is a problem with the internal state of the SM. This should not happen during normal operation. A memory corruption or invalid operation may have occurred.

enumerator gSmInvalidConnectionHandle_c

The target device does not have a valid connection handle. It might be disconnected.

enumerator gSmInproperKeyDistributionField_c

The Responder upper layer has set to “1” one or more flags in the Initiator or Responder Key Distribution Fields from the Pairing Request which were set to “0” by the peer device.

enumerator gSmUnexpectedKeyType_c

The Responder upper layer has set a key type field in the Passkey Request Reply command, which is different than the field negotiated with the peer device.

enumerator gSmUnexpectedPairingTerminationReason_c

The upper layer tried to cancel the pairing procedure with an unexpected pairing failure reason for the current phase of the pairing procedure.

enumerator gSmUnexpectedKeyset_c

The Responder upper layer is trying to distribute keys which were not requested during the pairing procedure or the peer device has sent a Key Distribution packet which was not expected.

enumerator gSmSmpTimeoutOccurred_c

An SMP timeout has occurred for the peer device. No more operations are accepted until a new physical link is established.

enumerator gSmUnknownSmpPacketType_c

An SMP packet with an unknown (or invalid) type has been received.

enumerator gSmInvalidSmpPacketLength_c

An SMP packet with an invalid length for the SMP packet type has been received.

enumerator gSmInvalidSmpPacketParameter_c

An SMP packet with an invalid parameter has been received.

enumerator gSmReceivedUnexpectedSmpPacket_c

An unexpected SMP packet was received.

enumerator gSmReceivedSmpPacketFromUnknownDevice_c

An SMP packet is received but the source Device ID cannot be identified.

enumerator gSmReceivedUnexpectedHciEvent_c

An HCI event has been received which cannot be handled by the SM or cannot be handled in the current context.

enumerator gSmReceivedHciEventFromUnknownDevice_c

An HCI event is received but the source Device ID cannot be identified.

enumerator gSmInvalidHciEventParameter_c

An HCI Event is received with an invalid parameter.

enumerator gSmLlConnectionEncryptionInProgress_c

A Link Layer Connection encryption was requested by the upper layer or attempted internally by the SM, but it could no be completed because an encryption was already in progress. This situation could lead to an SMP Pairing Failure when the SM cannot encrypt the link with the STK. An unspecified pairing failure reason is used in this instance.

enumerator gSmLlConnectionEncryptionFailure_c

The Link Layer connection encryption procedure has failed.

enumerator gSmInsufficientResources_c

The SM could not allocate resources to perform operations (memory or timers).

enumerator gSmOobDataAddressMismatch_c

The address of the peer contained in the remote OOB data sent to the stack does not match the address used by the remote device for the connection/pairing procedure.

enumerator gSmSmpPacketReceivedAfterTimeoutOccurred_c

A SMP packet has been received from a peer device for which a pairing procedure has timed out. No further operations are permitted until a new connection is established.

enumerator gSmReceivedTimerEventForUnknownDevice_c

An Timer event is received but the source Device ID cannot be identified.

enumerator gSmUnattainableLocalDeviceSecRequirements_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum security properties for the local device.

enumerator gSmUnattainableLocalDeviceMinKeySize_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum encryption key size for the local device.

enumerator gSmUnattainablePeripheralSecReqRequirements_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum security properties requested by the local device via an SMP Peripheral Security Request.

enumerator gSmInvalidPeerPublicKey_c

This status covers the case where the peer provides a public key with an identical X coordinate to our own.

enumerator gSmPairingErrorPasskeyEntryFailed_c

The passkey entry failed.

enumerator gSmPairingErrorConfirmValueFailed_c

The received random confirm value does not match the computed random confirm value.

enumerator gSmPairingErrorCommandNotSupported_c

The command is not supported.

enumerator gSmPairingErrorInvalidParameters_c

The command contains at least one invalid parameter.

enumerator gSmPairingErrorUnknownReason_c

Unknown pairing failure reason.

enumerator gSmPairingErrorTimeout_c

The pairing procedure timed out.

enumerator gSmPairingErrorAuthenticationRequirements_c

Authentication requirements were not met.

enumerator gSmPairingAlreadyStarted_c

The pairing process has already started.

enumerator gSmPairingErrorKeyRejected_c

The key was rejected.

enumerator gSmPairingErrorPairingNotSupported_c

Pairing is not supported by the device.

enumerator gSmPairingErrorEncryptionKeySize_c

The resultant encryption key size is not long enough for the security requirements of this device.

enumerator gSmPairingErrorDhKeyCheckFailed_c

DHKey Check value received doesn't match the one calculated by the local device.

enumerator gSmPairingErrorNumericComparisonFailed_c

Indicates that the confirm values in the numeric comparison protocol do not match.

enumerator gSmPairingErrorOobNotAvailable_c

Indicates that the OOB data is not available.

enumerator gSmPairingErrorRepeatedAttempts_c

Indicates that the pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request.

enumerator gSmPairingErrorBusy_c

Indicates that the device is not ready to perform a pairing procedure

enumerator gSmTbResolvableAddressDoesNotMatchIrk_c

The provided Resolvable Private Address and IRK do not match.

enumerator gSmTbInvalidDataSignature_c

The provided data signature does not match the computed data signature.

enumerator gSmKeySessionKeyDerivationFailed_c

The session key derivation failed.

enumerator gAttStatusBase_c

ATT status base.

enumerator gAttSuccess_c

Alias.

enumerator gGattStatusBase_c

GATT status base.

enumerator gGattSuccess_c

Alias.

enumerator gGattAnotherProcedureInProgress_c

Trying to start a GATT procedure while one is already in progress.

enumerator gGattLongAttributePacketsCorrupted_c

Writing a Long Characteristic failed because Prepare Write Request packets were corrupted.

enumerator gGattMultipleAttributesOverflow_c

Too many Characteristics are given for a Read Multiple Characteristic procedure.

enumerator gGattUnexpectedReadMultipleResponseLength_c

Read Multiple Characteristic procedure failed because unexpectedly long data was read.

enumerator gGattInvalidValueLength_c

An invalid value length was supplied to a Characteristic Read/Write operation.

enumerator gGattServerTimeout_c

No response was received from the Server.

enumerator gGattIndicationAlreadyInProgress_c

A Server Indication is already waiting for Client Confirmation.

enumerator gGattClientConfirmationTimeout_c

No Confirmation was received from the Client after a Server Indication.

enumerator gGattInvalidPduReceived_c

An invalid PDU length was received.

enumerator gGattPeerDisconnected_c

An ongoing GATT procedure could not be finished due to peer's disconnection.

enumerator gGattMtuExchangeInProgress_c

A Server Indication is already waiting for Client Confirmation.

enumerator gGattOutOfSyncProceduresOngoing_c

Client can't read the Database Hash until all pending procedures are done.

enumerator gGattConnectionSecurityRequirementsNotMet_c

Client cannot initiate communication if the device security requirements are not met, e.g., link is not yet encrypted/authenticated.

enumerator gGapStatusBase_c

GAP status base.

enumerator gGapSuccess_c

Alias.

enumerator gGapAdvDataTooLong_c
Trying to set too many bytes in the advertising payload.

enumerator gGapScanRspDataTooLong_c
Trying to set too many bytes in the scan response payload.

enumerator gGapDeviceNotBonded_c
Trying to execute an API that is only available for bonded devices.

enumerator gGapAnotherProcedureInProgress_c
Trying to start a GAP procedure while one is already in progress.

enumerator gDevDbStatusBase_c
DeviceDatabase status base.

enumerator gDevDbSuccess_c
Alias.

enumerator gDevDbCccdLimitReached_c
CCCD value cannot be saved because Server's CCCD list is full for the current client.

enumerator gDevDbCccdNotFound_c
CCCD with the given handle is not found in the Server's list for the current client.

enumerator gGattDbStatusBase_c
GATT Database status base.

enumerator gGattDbSuccess_c
Alias.

enumerator gGattDbInvalidHandle_c
An invalid handle was passed as parameter.

enumerator gGattDbCharacteristicNotFound_c
Characteristic was not found.

enumerator gGattDbCccdNotFound_c
CCCD was not found.

enumerator gGattDbServiceNotFound_c
Service Declaration was not found.

enumerator gGattDbDescriptorNotFound_c
Characteristic Descriptor was not found.

enumerator gGattDbServiceOrCharAlreadyDeclared_c
Service or characteristic already declared

enumerator gCsStatusBase_c
Channel Sounding status base.

enumerator gCsSuccess_c
Alias.

enumerator gCsCallbackAlreadyInstalled_c
The selected callback was already installed.

enumerator gCsSecurityCheckFail_c
GAP Mode 4 security check fail.

enum bearerStatus_tag
Enhanced ATT bearer status values
Values:

enumerator gEattBearerActive_c

enumerator gEattBearerReconfInProgress_c

enumerator gEattBearerSuspended_c

enumerator gEattBearerAllocated_c

enumerator gEattBearerFree_c

enumerator gEattBearerAlreadyAllocated_c

enum bleAdvertisingType_t
Advertising Type
Values:

enumerator gAdvConnectableUndirected_c
Answers to both connect and scan requests.

enumerator gAdvDirectedHighDutyCycle_c
Answers only to connect requests; smaller advertising interval for quicker connection.

enumerator gAdvScannable_c
Answers only to scan requests.

enumerator gAdvNonConnectable_c
Does not answer to connect nor scan requests.

enumerator gAdvDirectedLowDutyCycle_c
Answers only to connect requests; larger advertising interval.

enum bleAdvReportEventProperties_tag

Values:

enumerator gAdvEventConnectable_c
Connectable Advertisement

enumerator gAdvEventScannable_c
Scannable Advertisement

enumerator gAdvEventDirected_c
Directed Advertisement

enumerator gAdvEventScanResponse_c
Scan Response

enumerator gAdvEventLegacy_c
Legacy Advertisement PDU

enumerator gAdvEventAnonymous_c
Anonymous Advertisement

enum bleAdvRequestProperties_tag

Values:

enumerator gAdvReqConnectable_c
Connectable Advertising

enumerator gAdvReqScannable_c
Scannable Advertising

enumerator gAdvReqDirected_c
Directed Advertising

enumerator gAdvReqHighDutyCycle_c
High Duty Cycle

enumerator gAdvReqLegacy_c
Legacy Advertising PDU

enumerator gAdvReqAnonymous_c
Anonymous Advertising

enumerator gAdvIncludeTxPower_c
Set this option to include the Tx power in advertising packet.

enumerator gAdvUseDecisionPDU_c
Use ADV_DECISION_IND PDU(0b1001).

enumerator gAdvIncludeAdvAinDecisionPDU_c
Include AdvA in the extended header of all decision PDUs.

enumerator gAdvIncludeADIinDecisionPDU_c
Include ADI in the extended header of all decision PDUs.

enum bleAdvertisingFilterPolicy_t
Values:

enumerator gBleAdvFilterAllowScanFromAnyAllowConnFromAny_c
Filter Accept List is ignored.

enumerator gBleAdvFilterAllowScanFromWLAllowConnFromAny_c
Filter Accept List is used only for Scan Requests.

enumerator gBleAdvFilterAllowScanFromAnyAllowConnFromWL_c
Filter Accept List is used only for Connection Requests.

enumerator gBleAdvFilterAllowScanFromWLAllowConnFromWL_c
Filter Accept List is used for both Scan and Connection Requests.

enum bleLlConnectionRole_t
Values:

enumerator gBleLlConnectionCentral_c
Link Layer Central Role

enumerator gBleLlConnectionPeripheral_c
Link Layer Peripheral Role

enum bleCentralClockAccuracy_tag
Values:

enumerator gBleCentralClkAcc500ppm_c

enumerator gBleCentralClkAcc250ppm_c

enumerator gBleCentralClkAcc150ppm_c

enumerator gBleCentralClkAcc100ppm_c

enumerator gBleCentralClkAcc75ppm_c

enumerator gBleCentralClkAcc50ppm_c

enumerator gBleCentralClkAcc30ppm_c

enumerator gBleCentralClkAcc20ppm_c

enum bleAdvertiserClockAccuracy_tag

Values:

enumerator gBleAdvertiserClkAcc500ppm_c

enumerator gBleAdvertiserClkAcc250ppm_c

enumerator gBleAdvertiserClkAcc150ppm_c

enumerator gBleAdvertiserClkAcc100ppm_c

enumerator gBleAdvertiserClkAcc75ppm_c

enumerator gBleAdvertiserClkAcc50ppm_c

enumerator gBleAdvertiserClkAcc30ppm_c

enumerator gBleAdvertiserClkAcc20ppm_c

enum bleAdvertisingReportEventType_t

Values:

enumerator gBleAdvRepAdvInd_c

enumerator gBleAdvRepAdvDirectInd_c

enumerator gBleAdvRepAdvScanInd_c

enumerator gBleAdvRepAdvNonconnInd_c

enumerator gBleAdvRepScanRsp_c

enum hciPacketType_t

Values:

enumerator gHciCommandPacket_c

HCI Command

enumerator gHciDataPacket_c

L2CAP Data Packet

enumerator gHciSynchronousDataPacket_c

Not used in BLE

enumerator gHciEventPacket_c
HCI Event

enumerator gHciIsoDataPacket_c
HCI ISO data packet

enum bleScanType_t
Scanning type enumeration.
Values:

enumerator gScanTypePassive_c
Passive Scanning - advertising packets are immediately reported to the Host.

enumerator gScanTypeActive_c
Active Scanning - the scanner sends scan requests to the advertiser and reports to the Host after the scan response is received.

enum bleTransmitPowerLevelType_t
Values:

enumerator gReadCurrentTxPowerLevel_c
Current TX Power level.

enumerator gReadMaximumTxPowerLevel_c
Maximum recorded TX Power level.

enum bleTransmitPowerChannelType_t
Values:

enumerator gTxPowerAdvChannel_c
Advertising channel type when setting Tx Power.

enumerator gTxPowerConnChannel_c
Connection channel type when setting Tx Power.

enum bleChannelOverrideMode_t
Values:

enumerator gOverrideAdvChannels_c
Set the advertising channels

enumerator gOverrideScanChannels_c
Set the scanning channels

enumerator gOverrideInitChannels_c
Set the connection initiation channels

enum bleChannelFrequency_t
Values:

enumerator gBleFreq2402MHz_c

enumerator gBleFreq2404MHz_c

enumerator gBleFreq2406MHz_c

enumerator gBleFreq2408MHz_c

enumerator gBleFreq2410MHz_c

enumerator gBleFreq2412MHz_c

enumerator gBleFreq2414MHz_c

enumerator gBleFreq2416MHz_c

enumerator gBleFreq2418MHz_c

enumerator gBleFreq2420MHz_c

enumerator gBleFreq2422MHz_c

enumerator gBleFreq2424MHz_c

enumerator gBleFreq2426MHz_c

enumerator gBleFreq2428MHz_c

enumerator gBleFreq2430MHz_c

enumerator gBleFreq2432MHz_c

enumerator gBleFreq2434MHz_c

enumerator gBleFreq2436MHz_c

enumerator gBleFreq2438MHz_c

enumerator gBleFreq2440MHz_c

enumerator gBleFreq2442MHz_c

enumerator gBleFreq2444MHz_c

enumerator gBleFreq2446MHz_c

enumerator gBleFreq2448MHz_c

enumerator gBleFreq2450MHz_c

enumerator gBleFreq2452MHz_c

enumerator gBleFreq2454MHz_c

enumerator gBleFreq2456MHz_c

enumerator gBleFreq2458MHz_c

enumerator gBleFreq2460MHz_c

enumerator gBleFreq2462MHz_c

enumerator gBleFreq2464MHz_c

enumerator gBleFreq2466MHz_c

enumerator gBleFreq2468MHz_c

enumerator gBleFreq2470MHz_c

enumerator gBleFreq2472MHz_c

enumerator gBleFreq2474MHz_c

enumerator gBleFreq2476MHz_c

enumerator gBleFreq2478MHz_c

enumerator gBleFreq2480MHz_c

enum bleTxTestPacketPayload_t

Values:

enumerator gBleTestPacketPayloadPrbs9_c

enumerator gBleTestPacketPayloadPattern11110000_c

enumerator gBleTestPacketPayloadPattern10101010_c

enumerator gBleTestPacketPayloadPrbs15_c

enumerator gBleTestPacketPayloadPatternAllBits1_c

enumerator gBleTestPacketPayloadPatternAllBits0_c

enumerator gBleTestPacketPayloadPattern00001111_c

enumerator gBleTestPacketPayloadPattern01010101_c

enum bleHardwareErrorCode_t

Values:

enumerator bleHwErrCodeNoError_c

enum leSupportedFeatures_tag

Values:

enumerator gLeEncryption_c

enumerator gLeConnectionParametersRequestProcedure_c

enumerator gLeExtendedRejectIndication_c

enumerator gLePeripheralInitiatedFeaturesExchange_c

enumerator gLePing_c

enumerator gLeDataPacketLengthExtension_c

enumerator gLeLlPrivacy_c

enumerator gLeExtendedScannerFilterPolicies_c

enumerator gLe2MbPhy_c

enumerator gLeStableModulationIdxTx_c

enumerator gLeStableModulationIdxRx_c

enumerator gLeCodedPhy_c

enumerator gLeExtendedAdv_c

enumerator gLePeriodicAdv_c

enumerator gLeChannelSelAlg2_c

enumerator gLePowerClass1_c

enumerator gLeMinNumOfUsedChanProcedure_c

enumerator gLePeriodicAdvSyncTransferSender_c

enumerator gLePeriodicAdvSyncTransferReceiver_c

enumerator gLePowerControlRequest1_c

enumerator gLePowerControlRequest2_c

enumerator gLePathLossMonitoring_c

enumerator gAdvertisingCodingSelection_c

enumerator gAdvertisingCodingSelectionHostSupport_c

enumerator gLePawrAdvertiser_c

enumerator gLePawrScanner_c

enumerator gLeDecisionBasedAdvertisingFiltering_c

enum leExtendedSupportedFeatures_tag

Values:

enumerator gLeMonitoringAdvertisers_c

enum gapGenericEventType_t

Generic Event Type

Values:

enumerator gInitializationComplete_c

Initial setup started by Ble_HostInitialize is complete.

enumerator gInternalError_c

An internal error occurred.

enumerator gAdvertisingSetupFailed_c

Error during advertising setup.

enumerator gAdvertisingParametersSetupComplete_c

Advertising parameters have been successfully set. Response to Gap_SetAdvertisingParameters.

enumerator gAdvertisingDataSetupComplete_c

Advertising and/or scan response data has been successfully set. Response to Gap_SetAdvertisingData.

enumerator gFilterAcceptListSizeRead_c

Contains the Filter Accept List size. Response to Gap_ReadFilterAcceptListSize.

enumerator gDeviceAddedToFilterAcceptList_c

Device has been added to Filter Accept List. Response to Gap_AddDeviceToFilterAcceptList.

enumerator gDeviceRemovedFromFilterAcceptList_c

Device has been removed from the Filter Accept List. Response to Gap_RemoveDeviceFromFilterAcceptList.

enumerator gFilterAcceptListCleared_c

Filter Accept List has been cleared. Response to Gap_ClearFilterAcceptList.

enumerator gRandomAddressReady_c

A random device address has been created. Response to Gap_CreateRandomDeviceAddress.

enumerator gCreateConnectionCanceled_c

Connection initiation was successfully cancelled. Response to Gap_CancelInitiatingConnection.

enumerator gPublicAddressRead_c

Contains the public device address. Response to Gap_ReadPublicDeviceAddress.

enumerator gAdvTxPowerLevelRead_c

Contains the TX power on the advertising channel. Response to Gap_ReadAdvertisingTxPowerLevel.

enumerator gPrivateResolvableAddressVerified_c

Contains the result of PRA verification. Response to Gap_VerifyPrivateResolvableAddress.

enumerator gRandomAddressSet_c

Random address has been set into the Controller. Response to Gap_SetRandomAddress.

enumerator gLeScPublicKeyRegenerated_c

The private/public key pair used for LE Secure Connections pairing has been regenerated.

enumerator gLeScLocalOobData_c

Local OOB data used for LE Secure Connections pairing.

enumerator gHostPrivacyStateChanged_c

Host Privacy was enabled or disabled.

enumerator gControllerPrivacyStateChanged_c
Controller Privacy was enabled or disabled.

enumerator gControllerTestEvent_c
Controller Test was started or stopped.

enumerator gTxPowerLevelSetComplete_c
Controller Tx Power Level set complete or invalid.

enumerator gLePhyEvent_c
Phy Mode of a connection has been updated by the Controller.

enumerator gControllerNotificationEvent_c
Controller Enhanced Notification received.

enumerator gGetConnParamsComplete_c
Get Connection Parameters command complete

enumerator gBondCreatedEvent_c
Bond Created Event signalling the stack created a bond after pairing or at app request.

enumerator gChannelMapSet_c
Channel map set complete in the Controller.

enumerator gExtAdvertisingParametersSetupComplete_c
Extended advertising parameters have been successfully set.

enumerator gExtAdvertisingDataSetupComplete_c
Extended advertising data has been successfully set.

enumerator gExtAdvertisingSetRemoveComplete_c
An advertising set has been removed from the Controller.

enumerator gPeriodicAdvParamSetupComplete_c
Periodic advertising parameters have been successfully set.

enumerator gPeriodicAdvDataSetupComplete_c
Periodic advertising data have been successfully set.

enumerator gPeriodicAdvertisingStateChanged_c
Event received when periodic advertising has been successfully enabled or disabled.

enumerator gPeriodicAdvListUpdateComplete_c
Periodic advertiser list has been successfully updated.

enumerator gPeriodicAdvCreateSyncCancelled_c
Periodic advertising create sync command was successfully cancelled

enumerator gTxEntryAvailable_c

This event is generated when a TX entry becomes available after they were all in use.

enumerator gControllerLocalRPARead_c

Contains the resolvable private device address. Response to Gap_ReadControllerLocalRPA.

enumerator gConnectionlessCteTransmitParamsSetupComplete_c

Connectionless CTE transmit parameters have been successfully set.

enumerator gConnectionlessCteTransmitStateChanged_c

Connectionless CTE for an advertising set was enabled or disabled.

enumerator gConnectionlessIqSamplingStateChanged_c

Connectionless CTE IQ sampling for an advertising train was enabled or disabled.

enumerator gAntennaInformationRead_c

Antenna information was read from the controller.

enumerator gModifiedSleepClockAccuracy_c

The Sleep Clock accuracy was changed

enumerator gPeriodicAdvRecvEnableComplete_c

Enable or disable reports for the periodic advertising train command is complete

enumerator gPeriodicAdvSyncTransferComplete_c

The command used to instruct the Controller to send synchronization information about the periodic advertising train identified by the Sync_Handle parameter to a connected device is complete

enumerator gPeriodicAdvSetInfoTransferComplete_c

The command used to instruct the Controller to send synchronization information about the periodic advertising in an advertising set to a connected device is complete

enumerator gSetPeriodicAdvSyncTransferParamsComplete_c

The command specifying how the Controller will process periodic advertising synchronization information is complete

enumerator gSetDefaultPeriodicAdvSyncTransferParamsComplete_c

The command which set the default parameters for periodic advertising synchronization information is complete

enumerator gPeriodicAdvSyncTransferSucceeded_c

Event received when Controller has succeeded the Synchronization to the periodic advertising train

enumerator gPeriodicAdvSyncTransferFailed_c

Event received when Controller has failed the Synchronization to the periodic advertising train

enumerator gConnEvtLeGenerateDhKeyComplete_c

DHKey generation is complete. Key can be found in gapConnectionEvent_t.eventData.leGenerateDhKeyCompleteEvent

enumerator gHandoverGetComplete_c

Handover data get complete.

enumerator gHandoverSetComplete_c

Handover data set complete.

enumerator gHandoverGetCsLlContextComplete_c

Handover CS LL context data get complete.

enumerator gHandoverSetCsLlContextComplete_c

Handover CS LL context data set complete.

enumerator gHandoverGetTime_c

Handover Get Time command complete

enumerator gHandoverSuspendTransmitComplete_c

Handover Suspend Transmit command complete

enumerator gHandoverResumeTransmitComplete_c

Handover Resume Transmit command complete

enumerator gHandoverAnchorNotificationStateChanged_c

Handover Anchor Notification command complete

enumerator gHandoverAnchorSearchStarted_c

Handover Anchor Search Start command complete

enumerator gHandoverAnchorSearchStopped_c

Handover Anchor Search Stop command complete

enumerator gHandoverTimeSyncTransmitStateChanged_c

Handover Time Sync Transmit command complete

enumerator gHandoverTimeSyncReceiveComplete_c

Handover Time Sync Receive command complete

enumerator gHandoverAnchorMonitorEvent_c

Event received from Controller - Handover Anchor Monitor

enumerator gHandoverTimeSyncEvent_c

Event received from Controller - Handover Time Sync

enumerator gHandoverConnParamUpdateEvent_c

Event received from Controller - Handover Conn Params Update

enumerator gRemoteVersionInformationRead_c

Version Information of a peer was read

enumerator gLlSkdReportEvent_c

Session Key Diversifier report

enumerator gLeSetSchedulerPriorityComplete_c

LE Set Scheduler Priority command complete

enumerator gDeInitializationComplete_c

Event received when Ble_HostDeInitialize is complete.

enumerator gHandoverAnchorMonitorPacketEvent_c

Event received from Controller - Handover Anchor Monitor Packet. pPdu must be freed by the application

enumerator gHandoverAnchorMonitorPacketContinueEvent_c

Event received from Controller - Handover Anchor Monitor Packet Continue. pPdu must be freed by the application

enumerator gHandoverFreeComplete_c

Handover data free complete.

enumerator gHandoverUpdateConnParamsComplete_c

Handover Update Connection Parameters command complete

enumerator gExtAdvertisingDecisionDataSetupComplete_c

Extended advertising decision data has been successfully set.

enumerator gDecisionInstructionsSetupComplete_c

decision instructions used when listening for decision advertisements PDUs has been successfully set.

enumerator gHandoverLlPendingData_c

ACL Data pending to be transmited by the LL

enumerator gLeChannelOverrideComplete_c

LE Channel Override command complete

enumerator gPeriodicAdvSetSubeventDataComplete_c

Periodic advertising subevent data has been successfully set.

enumerator gPeriodicAdvSetResponseDataComplete_c

Periodic advertising response data has been successfully set.

enumerator gPeriodicSyncSubeventComplete_c

Set Sync Subevent command successfully completed.

enumerator gHandoverConnectionUpdateProcedureEvent_c

This event is used to report the new connection parameters indicated during the Connection Update procedure

enumerator gHandoverApplyConnectionUpdateProcedureComplete_c

This event is used to report the new connection parameters indicated during the Connection Update procedure

enumerator gVendorUnitaryTestComplete_c

Vendor Unitary Test command complete

enumerator gLeSetDataRelatedAddressChangesComplete_c

Set Data Related Address Changes command complete

enumerator gLePeriodicAdvUpdateSyncComplete_c

Le Periodic Adv Update Sync command complete

enumerator gDeviceAddedToMonAdvList_c

LE Add Device To Monitored Advertisers List command complete

enumerator gDeviceRemovedFromMonAdvList_c

Remove Device From Monitored Advertisers List command complete

enumerator gMonAdvListCleared_c

Clear Monitored Advertisers List command complete

enumerator gMonAdvEnabled_c

Enable Monitoring Advertisers command complete

enumerator gMonAdvListSizeRead_c

Read Monitored Advertisers List Size command complete

enum gapInternalErrorSource_t

Internal Error Source - the command that triggered the error

Values:

enumerator gHciCommandStatus_c

enumerator gCheckPrivateResolvableAddress_c

enumerator gVerifySignature_c

enumerator gAddNewConnection_c

enumerator gResetController_c

enumerator gSetEventMask_c

enumerator gReadLeBufferSize_c

enumerator gSetLeEventMask_c

enumerator gReadDeviceAddress_c

enumerator gReadLocalSupportedFeatures_c

enumerator gReadFilterAcceptListSize_c

enumerator gClearFilterAcceptList_c

enumerator gAddDeviceToFilterAcceptList_c

enumerator gRemoveDeviceFromFilterAcceptList_c

enumerator gCancelCreateConnection_c

enumerator gReadRadioPower_c

enumerator gSetRandomAddress_c

enumerator gCreateRandomAddress_c

enumerator gEncryptLink_c

enumerator gProvideLongTermKey_c

enumerator gDenyLongTermKey_c

enumerator gConnect_c

enumerator gDisconnect_c

enumerator gTerminatePairing_c

enumerator gSendPeripheralSecurityRequest_c

enumerator gEnterPasskey_c

enumerator gProvideOob_c

enumerator gSendSmpKeys_c

enumerator gWriteSuggestedDefaultDataLength_c

enumerator gReadSuggestedDefaultDataLength_c

enumerator gUpdateLeDataLength_c

enumerator gEnableHostPrivacy_c

enumerator gEnableControllerPrivacy_c

enumerator gLeScSendKeypressNotification_c

enumerator gLeScSetPeerOobData_c

enumerator gLeScGetLocalOobData_c

enumerator gLeScValidateNumericValue_c

enumerator gLeScRegeneratePublicKey_c

enumerator gLeSetResolvablePrivateAddressTimeout_c

enumerator gDefaultPairingProcedure_c

enumerator gLeControllerTest_c

enumerator gLeReadPhy_c

enumerator gLeSetPhy_c

enumerator gSaveKeys_c

enumerator gSetChannelMap_c

enumerator gReadLocalSupportedCommands_c

enumerator gEnableLdmTimer_c

enumerator gRemoveAdvertisingSet_c

enumerator gLePeriodicAdvSyncEstb_c

enumerator gLePeriodicAdvSyncLost_c

enumerator gLeRemoveDeviceFromPeriodicAdvList_c

enumerator gLeClearPeriodicAdvList_c

enumerator gLeAddDeviceToPeriodicAdvList_c

enumerator gLeReadNumOfSupportedAdvSets_c

enumerator gLeReadPeriodicAdvListSize_c

enumerator gLeReadMaxAdvDataLen_c

enumerator gPeriodicAdvCreateSync

enumerator gPeriodicAdvCancelSync

enumerator gPeriodicAdvTerminateSync

enumerator gL2capRxPacket_c

enumerator gExtAdvReportProcess_c

enumerator gReadControllerLocalRPA_c

enumerator gHciEventReceiveHandler_c

enumerator gSetConnectionlessCteTransmitParams_c

enumerator gSetConnectionlessCteTransmitEnable_c

enumerator gSetConnectionlessIqSamplingEnable_c

enumerator gReadAntennaInformation_c

enumerator gSetConnectionCteReceiveParams_c

enumerator gSetConnectionCteTransmitParams_c

enumerator gConnectionCteReqEnable_c

enumerator gConnectionCteRspEnable_c

enumerator gGenerateDHKeyV2_c

enumerator gModifySleepClockAccuracy_c

enumerator gPeriodicAdvRcvEnable_c

enumerator gPeriodicAdvSyncTransfer_c

enumerator gPeriodicAdvSetInfoTransfer_c

enumerator gSetPeriodicAdvSyncTransferParams_c

enumerator gSetDefaultPeriodicAdvSyncTransferParams_c

enumerator gEnhancedReadTransmitPowerLevel_c

enumerator gReadRemoteTransmitPowerLevel_c

enumerator gSetPathLossReportingParams_c

enumerator gSetPathLossReportingEnable_c

enumerator gSetTransmitPowerReportingEnable_c

enumerator gEattConnectionRequest_c

enumerator gEattConnectionAccept_c

enumerator gEattReconfigureRequest_c

enumerator gEattSendCreditsRequest_c

enumerator gEattDisconnectRequest_c

enumerator gEattL2caCancelConnection_c

enumerator gEattL2caSendLeFlowControlCredit_c

enumerator gEattL2caDisconnectLePsm_c

enumerator gEattL2caHandleSendLeCbData_c

enumerator gEattL2caHandleRecvLeCbData_c

enumerator gEattL2caEnhancedReconfigureReq_c

enumerator gEattL2caEnhancedCancelConnection_c

enumerator gHciRecvFragmentOfPacket_c

enumerator gHciDataDiscardedAlloc_c

A memory allocation failure occurred in the HCI layer. Data was discarded.

enumerator gHciDataDiscardedInvalidStateParam_c

A packet with an invalid parameter or length was received by the HCI layer. Data was discarded.

enumerator gGetConnParams_c

An error occurred during the Get Connection Params procedure

enumerator gHandover_c

An error occurred during the connection handover process

enumerator gHandoverGetLlContext_c

An error occurred during the Handover Get LL Context procedure

enumerator gHandoverConnect_c

An error occurred during the Handover Connect procedure

enumerator gHandoverDisconnect_c

An error occurred during the Handover Disconnect procedure

enumerator gHandoverSetSkd_c

An error occurred during the Handover Set SKD procedure

enumerator gHandoverSuspendTransmitLlProcInProgress_c

Could not suspend transmit at LL level due to an LL procedure being in progress

enumerator gHandoverSuspendTransmitHostTxInProgress_c

Could not suspend transmit at Host level due to ongoing data transfers

enumerator gHandoverGetCsLlContext_c

An error occurred during the Handover Get CS LL Context procedure

enumerator gHandoverSetCsLlContext_c

An error occurred during the Handover Set CS LL Context procedure

enumerator gHandoverUpdateConnParams_c

An error occurred during the Handover Update Connection Parameters procedure

enumerator gReadRemoteVersionInfo_c

enumerator gLeSetSchedulerPriority_c

enumerator gLeSetHostFeature_c

enumerator gSetExtAdvDecisionData_c

enumerator gSetDecisionInstructions_c

enumerator gSetExpmSupportedFeatures_c

enumerator gHandoverSuspendTransmitHciTx_c
Could not suspend transmit due to HCI tx error

enumerator gHandoverResumeTransmitHciTx_c
Could not resume transmit due to HCI tx error

enumerator gHandoverAnchorNotifHciTx_c
Could not enable/disable Anchor Notification due to HCI tx error

enumerator gHandoverAnchorSearchStartHciTx_c
Could not start Anchor Search due to HCI tx error

enumerator gHandoverAnchorSearchStopHciTx_c
Could not stop Anchor Search due to HCI tx error

enumerator gHandoverTimeSyncTxHciTx_c
Could not start/stop Anchor Time Sync transmit due to HCI tx error

enumerator gHandoverTimeSyncRxHciTx_c
Could not start/stop Anchor Time Receive transmit due to HCI tx error

enumerator gHandoverSuspendTransmitLl_c
Handover Suspend Transmit Link Layer error

enumerator gHandoverGetTimeLl_c
Handover Get Time Link Layer error

enumerator gHandoverResumeTransmitLl_c
Handover Resume Transmit Link Layer error

enumerator gHandoverAnchorNotifLl_c
Handover Anchor Notification Link Layer error

enumerator gHandoverAnchorSearchStartLl_c
Handover Anchor Search Start Link Layer error

enumerator gHandoverAnchorSearchStopLl_c
Handover Anchor Search Stop Link Layer error

enumerator gHandoverTimeSyncTxLl_c
Handover Time Sync Transmit Link Layer error

enumerator gHandoverTimeSyncRxLl_c
Handover Time Sync Receive Link Layer error

enumerator gHandoverSetLlPendingData_c
Handover Set LL Pending Data error

enumerator gLeChannelOverride_c
LE Channel Override command unsuccessful

enumerator gLeSetPeriodicAdvParamsV2_c

enumerator gLeSetPeriodicAdvSubeventData_c

enumerator gLeSetPeriodicAdvResponseData_c

enumerator gLeSetPeriodicSyncSubevent_c

enumerator gLePeriodicAdvResponseReport_c

enumerator gHandoverApplyConnectionUpdateProcedure_c
An error occurred during the Handover Apply Connection Update procedure

enumerator gVendorUnitaryTest_c
An error occurred during the Vendor Unitary Test procedure

enumerator gSetDataRelatedAddressChanges_c
An error occurred during the Set Data Related Address procedure

enumerator gLePeriodicAdvUpdateSync_c
An error occurred during the Le Periodic Adv Update Sync procedure

enumerator gReadAllLocalSupportedFeatures_c
An error occurred during the LE Read All Local Supported Features command

enumerator gAddDeviceToMonAdvList_c
An error occurred during the Add Device To Monitored Advertisers List

enumerator gRemoveDeviceFromMonAdvList_c
An error occurred during the Remove Device From Monitored Advertisers List

enumerator gClearMonAdvList_c
An error occurred during the Clear Monitored Advertisers List

enumerator gEnableMonAdv_c
An error occurred during the Enable Monitoring Advertisers

enumerator gReadMonAdvListSize_c
An error occurred during the Read Monitored Advertisers List Size

enum gapControllerTestEventType_t
Controller Test Event Type
Values:

enumerator gControllerReceiverTestStarted_c

enumerator gControllerTransmitterTestStarted_c

enumerator gControllerTestEnded_c

enum gapLeAllPhyFlags_t

Le All Phys Preferences flags.

Values:

enumerator gLeTxPhyNoPreference_c

Host has no preference for Tx Phy

enumerator gLeRxPhyNoPreference_c

Host has no preference for Rx Phy

enum gapLePhyOptionsFlags_t

Le Phys Options Preferences flags.

Values:

enumerator gLeCodingNoPreference_c

Host has no preference on the LE Coded Phy

enumerator gLeCodingS2_c

Host prefers to use S=2 on the LE Coded Phy

enumerator gLeCodingS8_c

Host prefers to use S=8 on the LE Coded Phy

enumerator gLeCodingS2Req_c

Host requires to use S=2 on the LE Coded Phy

enumerator gLeCodingS8Req_c

Host requires to use S=8 on the LE Coded Phy

enum gapLePhyMode_tag

Values:

enumerator gLePhy1M_c

Tx/Rx Phy on the connection is LE 1M

enumerator gLePhy2M_c

Tx/Rx Phy on the connection is LE 2M

enumerator gLePhyCoded_c

Tx/Rx Phy on the connection is LE Coded

enum gapPhyEventType_t

Phy Event Type

Values:

enumerator gPhySetDefaultComplete_c

Gap_LeSetPhy default mode was successful

enumerator gPhyRead_c

Gap_LeReadPhy return values

enumerator gPhyUpdateComplete_c

Gap_LeSetPhy return values for a connection or an update occurred

enum bleNotificationEventType_tag

Values:

enumerator gNotifEventNone_c

No enhanced notification event enabled

enumerator gNotifConnEventOver_c

Connection event over

enumerator gNotifConnRxPdu_c

Connection Rx PDU

enumerator gNotifAdvEventOver_c

Advertising event over

enumerator gNotifAdvTx_c

Advertising ADV transmitted

enumerator gNotifAdvScanReqRx_c

Advertising SCAN REQ Rx

enumerator gNotifAdvConnReqRx_c

Advertising CONN REQ Rx

enumerator gNotifScanEventOver_c

Scanning event over

enumerator gNotifScanAdvPktRx_c

Scanning ADV PKT Rx

enumerator gNotifScanRspRx_c

Scanning SCAN RSP Rx

enumerator gNotifScanReqTx_c

Scanning SCAN REQ Tx

enumerator gNotifConnCreated_c
Connection created

enumerator gNotifChannelMatrix_c
Enable channel status monitoring (KW37 only)

enumerator gNotifPhyReq_c
Phy Req Pdu ack received (KW37 only)

enumerator gNotifConnChannelMapUpdate_c
Channel map update

enumerator gNotifConnInd_c
Connect indication

enumerator gNotifPhyUpdateInd_c
Phy update indication

enum bleCteTransmitEnable_t
Values:

enumerator gCteTransmitDisable_c

enumerator gCteTransmitEnable_c

enum bleIqSamplingEnable_t
Values:

enumerator gIqSamplingDisable_c

enumerator gIqSamplingEnable_c

enum bleCteReqEnable_t
Values:

enumerator gCteReqDisable_c

enumerator gCteReqEnable_c

enum bleCteRspEnable_t
Values:

enumerator gCteRspDisable_c

enumerator gCteRspEnable_c

enum bleCteType_tag
Values:

enumerator gCteTypeAoA_c

AoA Constant Tone Extension

enumerator gCteTypeAoD1us_c

AoD Constant Tone Extension with 1 microsecond slots

enumerator gCteTypeAoD2us_c

AoD Constant Tone Extension with 2 microsecond slots

enumerator gCteTypeNoCte_c

No Constant Tone Extension

enum bleSlotDurations_tag

Values:

enumerator gSlotDurations1us_c

Switching and sampling slot durations are 1 microsecond each

enumerator gSlotDurations2us_c

Switching and sampling slot durations are 2 microseconds each

enum bleIqReportPacketStatus_tag

Values:

enumerator gIqReportPacketStatusCorrectCrc_c

CRC was correct

enumerator gIqReportPacketStatusCrcIncorrectUsedLength_c

CRC was incorrect and the Length and CTETime fields of the packet were used to determine sampling points

enumerator gIqReportPacketStatusCrcIncorrectUsedOther_c

CRC was incorrect but the Controller has determined the position and length of the Constant Tone Extension in some other way

enumerator gIqReportPacketStatusInsufficientResources_c

Insufficient resources to sample (Channel_Index, CTE_Type, and Slot_Durations invalid).

enum blePathLossThresholdZoneEntered_tag

Values:

enumerator gPathLossThresholdLowZone_c

Entered low zone.

enumerator gPathLossThresholdMiddleZone_c

Entered middle zone.

enumerator gPathLossThresholdHighZone_c
Entered high zone.

enum bleTxPowerReportingReason_tag
Values:

enumerator gLocalTxPowerChanged_c
Local transmit power changed.

enumerator gRemoteTxPowerChanged_c
Remote transmit power changed.

enumerator gReadRemoteTxPowerLevelCommandCompleted_c
HCI_LE_Read_Remote_Transmit_Power_Level command completed.

enum blePowerControlPhyType_tag
Values:

enumerator gPowerControlLePhy1M_c
LE 1M Phy.

enumerator gPowerControlLePhy2M_c
LE 2M Phy.

enumerator gPowerControlLePhyCodedS8_c
LE Coded Phy with S=8 data coding.

enumerator gPowerControlLePhyCodedS2_c
LE Coded Phy with S=2 data coding.

enum blePathLossReportingEnable_t
Values:

enumerator gPathLossReportingDisable_c

enumerator gPathLossReportingEnable_c

enum bleTxPowerReportingEnable_t
Values:

enumerator gTxPowerReportingDisable_c

enumerator gTxPowerReportingEnable_c

enum bleHandoverSuspendTransmitMode_t
Values:

enumerator gDoNotUseEventCounter_c

enumerator gUseEventCounter_c

enumerator gSafeStopLlTx_c

enumerator gSafeStopHostLlTx_c

enum bleHandoverAnchorSearchMode_t

Values:

enumerator gSuspendTxMode_c

enumerator gRssiSniffingMode_c

enumerator gPacketMode_c

enum bleGetConnParamsMode_t

Values:

enumerator gSendEventAfterRemoteSNChange_c

enumerator gSendEventAfterLLProcUpdate_c

enum bleHandoverAnchorNotificationEnable_t

Values:

enumerator gAnchorNotificationDisable_c

enumerator gAnchorNotificationEnable_c

enum bleHandoverTimeSyncEnable_t

Values:

enumerator gTimeSyncDisable_c

enumerator gTimeSyncEnable_c

enum bleHandoverTimeSyncStopWhenFound_t

Values:

enumerator gTimeSyncDoNotStopWhenFound_c

enumerator gTimeSyncStopWhenFound_c

enum bleAdvIndexType_t

Values:

enumerator gAdvIndexAscend_c

enumerator gAdvIndexDescend_c

enumerator gAdvIndexUser_c

enumerator gAdvIndexRandom_c

typedef enum *bleResult_tag* bleResult_t

BLE result type - the return value of BLE API functions

typedef uint8_t deviceId_t

Unique identifier type for a connected device.

typedef uint8_t bearerId_t

ATT bearer identifier.

typedef enum *bearerStatus_tag* bearerStatus_t

Enhanced ATT bearer status values

typedef uint8_t bleAddressType_t

Bluetooth Device Address Type - Size: 1 Octet, Range: [gBleAddrTypePublic_c:gBleAddrTypeRandom_c]

typedef uint8_t bleDeviceAddress_t[(6U)]

Bluetooth Device Address - array of 6 bytes.

typedef uint8_t bleUuidType_t

Bluetooth UUID type - values chosen to correspond with the ATT UUID format

typedef uint16_t bleAdvReportEventProperties_t

Advertising Event properties

typedef uint16_t bleAdvRequestProperties_t

Advertising Request properties

typedef uint8_t bleCentralClockAccuracy_t

typedef uint8_t bleAdvertiserClockAccuracy_t

typedef uint8_t bleScanningFilterPolicy_t

Scanning filter policy enumeration - Size: 1 Octet, Range: [gScanAll_c:gScanWithFilterAcceptList_c]

typedef uint8_t bleInitiatorFilterPolicy_t

Initiator filter policy enumeration - Size: 1 Octet, Range: [gUseDeviceAddress_c:gUseFilterAcceptList_c]

typedef uint8_t blePrivacyMode_t
: Privacy Mode enumeration - Size: 1 Octet, Range: [gNetworkPrivacy_c:gDevicePrivacy_c]

typedef uint8_t bleChannelMap_t[(5U)]
: Bluetooth Channel map - array of 5 bytes.

typedef uint64_t leSupportedFeatures_t

typedef uint8_t gapLePhyFlags_t
: Le Tx/Rx Phys Preferences flags.

typedef uint8_t gapLePhyMode_t
: Le Tx/Rx Phys.

typedef uint16_t bleNotificationEventType_t
: Controller Enhanced Notification Event Type

typedef struct *bleBondCreatedEvent_tag* bleBondCreatedEvent_t
: Bond Created Event

typedef struct *gapAddrReadyEvent_t_tag* gapAddrReadyEvent_t
: Address Ready Event

typedef struct *bleAntennaInformation_tag* bleAntennaInformation_t
: Antenna Information Read Event

typedef struct *periodicAdvSyncTransferEvent_tag* periodicAdvSyncTransferEvent_t

typedef struct *periodicAdvSetInfoTransferEvent_tag* periodicAdvSetInfoTransferEvent_t

typedef struct *periodicAdvSetSyncTransferParamsEvent_tag*
periodicAdvSetSyncTransferParamsEvent_t

typedef struct *gapSyncTransferReceivedEventData_tag* gapSyncTransferReceivedEventData_t

typedef struct *getConnParams_tag* getConnParams_t

typedef struct *handoverGetTime_tag* handoverGetTime_t

typedef struct *handoverAnchorSearchStart_tag* handoverAnchorSearchStart_t

typedef struct *handoverAnchorSearchStop_tag* handoverAnchorSearchStop_t

typedef struct *handoverConnect_tag* handoverConnect_t

typedef struct *handoverGetData_tag* handoverGetData_t

typedef struct *handoverSetData_tag* handoverSetData_t

typedef struct *handoverGetCsLlContext_tag* handoverGetCsLlContext_t

typedef struct *handoverAnchorMonitorEvent_tag* handoverAnchorMonitorEvent_t

typedef struct *handoverAnchorMonitorPacketEvent_tag* handoverAnchorMonitorPacketEvent_t

typedef struct *handoverAnchorMonitorPacketContinueEvent_tag*
handoverAnchorMonitorPacketContinueEvent_t

typedef struct *handoverConnectionUpdateProcedureEvent_tag*
handoverConnectionUpdateProcedureEvent_t

typedef struct *handoverTimeSyncEvent_tag* handoverTimeSyncEvent_t

typedef struct *handoverConnParamUpdateEvent_tag* handoverConnParamUpdateEvent_t

typedef struct *handoverSuspendTransmitCompleteEvent_tag*
handoverSuspendTransmitCompleteEvent_t

typedef struct *handoverResumeTransmitCompleteEvent_tag*
handoverResumeTransmitCompleteEvent_t

typedef struct *handoverUpdateConnParams_tag* handoverUpdateConnParams_t

typedef struct *handoverApplyConnectionUpdateProcedure_tag*
handoverApplyConnectionUpdateProcedure_t

typedef struct *handoverAnchorNotificationStateChanged_tag*
handoverAnchorNotificationStateChanged_t

typedef struct *handoverLlPendingDataIndication_tag* handoverLlPendingDataIndication_t

typedef struct *gapRemoteVersionInfoRead_tag* gapRemoteVersionInfoRead_t

typedef struct *gapLlSkdReport_tag* gapLlSkdReport_t

typedef struct *vendorUnitaryTestEvent_tag* vendorUnitaryTestEvent_t

typedef void (*gapGenericCallback_t)(*gapGenericEvent_t* *pGenericEvent)

Generic Callback prototype.

typedef *bleResult_t* (*hciHostToControllerInterface_t)(*hciPacketType_t* packetType, void *pPacket, uint16_t packetSize)

Host-to-Controller API prototype.

typedef uint8_t bleCteType_t

typedef struct *bleSyncCteType_tag* bleSyncCteType_t

typedef uint8_t bleSlotDurations_t

typedef uint8_t bleIqReportPacketStatus_t

typedef uint8_t blePathLossThresholdZoneEntered_t

typedef uint8_t bleTxPowerReportingReason_t

typedef uint8_t blePowerControlPhyType_t

const uint8_t gBleMaxActiveConnections

const uint16_t gcConnectionEventMinDefault_c

const uint16_t gcConnectionEventMaxDefault_c

const uint8_t gBleEattMaxConnectionChannels

const uint16_t gBleEattPsmMtu

const uint8_t gMaxAdvReportQueueSize

const bool_t gUseHciCommandFlowControl

Enables/Disables Hci Command Flow Control if the host lib supports this feature

messaging_t gApp2Host_TaskQueue

App to Host message queue for the Host Task

messaging_t gHci2Host_TaskQueue

HCI to Host message queue for the Host Task

bleResult_t Ble_HostInitialize(*gapGenericCallback_t* genericCallback,
hciHostToControllerInterface_t hostToControllerInterface)

Performs initialization of the BLE Host stack.

Remark

Application must wait for the gInitializationComplete_c generic event.

Parameters

- genericCallback – **[in]** Callback used to propagate GAP generic events to the application.

- hostToControllerInterface – **[in]** LE Controller uplink interface function pointer

Returns
gBleSuccess_c or error.

bleResult_t Ble_HostDeInitialize(void)
performs De-Initialize the Bluetooth LE Host

Remark

Application must wait for the gDeInitializationComplete_c generic event.

Parameters

- none –

Return values
gBleSuccess_c – or error.

bleResult_t Ble_HciRecv(*hciPacketType_t* packetType, void *pHciPacket, uint16_t packetSize)
This is the BLE Host downlink interface function.

Remark

This function must be registered as a callback by the LE Controller and called to send HCI packets (events and LE-U data) to the BLE Host.

Parameters

- packetType – **[in]** The type of the packet sent by the LE Controller
- pHciPacket – **[in]** Pointer to the packet sent by the LE Controller
- packetSize – **[in]** Number of bytes sent by the LE Controller

Returns
gBleSuccess_c or gBleOutOfMemory_c

bleResult_t Ble_HciRecvFromIsr(*hciPacketType_t* packetType, void *pHciPacket, uint16_t packetSize)
This is the BLE Host downlink interface function to be used in ISR context.

Remark

This function must be registered as a callback by the HCI Transport layer and called to send HCI packets from the controller (events and LE-U data) to the BLE Host.

Parameters

- packetType – **[in]** The type of the packet sent by the LE Controller
- pHciPacket – **[in]** Pointer to the packet sent by the LE Controller
- packetSize – **[in]** Number of bytes sent by the LE Controller

Returns
gBleSuccess_c or gBleOutOfMemory_c

bool_t Ble_CheckMemoryStorage(void)

This function performs runtime checks to determine whether memory storage is properly configured for the application settings.

Returns
True if memory is configured correctly False otherwise.

OSA_EVENT_HANDLE_DEFINE (gHost_TaskEvent)

Event for the Host Task Queue

void Host_TaskHandler(void *args)

Contains the Host Task logic.

Remark

This function must be called exclusively by the Host Task code from the application.

---

gMaxAdvSets_c

gInvalidDeviceId_c

gInvalidNvmIndex_c

gcConnectionIntervalMin_c

Boundary values for the Connection Parameters (Standard GAP).

gcConnectionIntervalMax_c

gcConnectionPeripheralLatencyMax_c

gcConnectionSupervisionTimeoutMin_c

gcConnectionSupervisionTimeoutMax_c

gcConnectionIntervalMinDefault_c

Default values for the Connection Parameters (Preferred).

connIntervalmin = Conn_Interval_Min * 1.25 ms

Value of 0xFFFF indicates no specific minimum.

gcConnectionIntervalMaxDefault_c

connIntervalmax = Conn_Interval_Max * 1.25 ms

Value of 0xFFFF indicates no specific maximum.

gcConnectionPeripheralLatencyDefault_c

gcConnectionSupervisionTimeoutDefault_c
: Time = N * 10 ms

STATIC
: When unit testing is performed, access from unit test module to static functions/variables within the tested module is not possible and therefore the static storage class identifier shall be removed

gBleAddrTypePublic_c
: Bluetooth Device Address Types Public Device Address - fixed into the Controller by the manufacturer.

gBleAddrTypeRandom_c
: Random Device Address - set by the Host into the Controller for privacy reasons.

Ble_IsPrivateResolvableDeviceAddress(bleAddress)
: PRA condition: check the 6th byte - MSB should be 0; 2nd MSB should be 1.

Ble_IsPrivateNonresolvableDeviceAddress(bleAddress)
: PNRA condition: check the 6th byte - MSB should be 0; 2nd MSB should be 0.

Ble_IsRandomStaticDeviceAddress(bleAddress)
: RSA condition: check the 6th byte - MSB should be 1; 2nd MSB should be 1.

Ble_DeviceAddressesMatch(bleAddress1, bleAddress2)
: A macro used to compare two device addresses

Ble_CopyDeviceAddress(destinationAddress, sourceAddress)
: A macro used to copy device addresses

gBleUuidType16_c
: 16-bit standard UUID

gBleUuidType128_c
: 128-bit long/custom UUID

gBleUuidType32_c
: 32-bit UUID - not available as ATT UUID format

gLePhy1MFlag_c
: Host prefers to use LE 1M Tx/Rx Phy, possibly among others

gLePhy2MFlag_c
: Host prefers to use LE 2M Tx/Rx Phy, possibly among others

gLePhyCodedFlag_c
: Host prefers to use LE Coded Tx/Rx Phy, possibly among others

gAdvDataChange_c
: Advertising Data Change reason

gScanRspDataChange_c
: Scan Response Data Change reason

gUseDeviceAddress_c

Initiator filter policy values Initiates a connection with a specific device identified by its address.

gUseFilterAcceptList_c

Initiates connections with all the devices in the Filter Accept List at the same time.

gUseDeviceAddressNoDecisionPDUs_c

Initiator filter policy values extended for gBLE60_DecisionBasedAdvertisingFilteringSupport_d == TRUE Filter Accept List is not used to determine which advertiser to connect to. Decision PDUs shall be ignored. Peer_Address_Type and Peer_Address shall be used.

gUseFilterAcceptListNoDecisionPDUs_c

Filter Accept List is used to determine which advertiser to connect to. Decision PDUs shall be ignored. Peer_Address_Type and Peer_Address shall be ignored.

gUseOnlyDecisionPDUs_c

Filter Accept List is not used to determine which advertiser to connect to. Only Decision PDUs shall be processed. Peer_Address_Type and Peer_Address shall be ignored.

gUseFilterAcceptListAllPDUs_c

Filter Accept List is used to determine which advertiser to connect to. All PDUs shall be processed. Peer_Address_Type and Peer_Address shall be ignored.

gUseDecisionPDU_UseFilterAcceptListForOtherPDUs_c

All decision PDUs shall be processed. Filter Accept List is used to determine which other PDUs to process. Peer_Address_Type and Peer_Address shall be ignored.

gScanAll_c

Basic Scanning filter policy values. Kept for apps backward compatibility Basic unfiltered scanning filter policy.

gScanWithFilterAcceptList_c

Basic filtered scanning filter policy (using the Filter Accept List).

gBasicUnfilteredScan_c

Scanning filter policy values. Basic unfiltered scanning filter policy.

gBasicFilteredScan_c

Basic filtered scanning filter policy (using the Filter Accept List).

gExtendedUnfilteredScan_c

Extended unfiltered scanning filter policy.

gExtendedFilteredScan_c

Extended filtered scanning filter policy(using the Filter Accept List).

gScanOnlyNonDecisionPDUs_c

PDU Scanning filter policy values. To be used in bitwise OR operations with scanning filter policy above only with gBLE60_DecisionBasedAdvertisingFilteringSupport_d == TRUE e.g.

((uint8_t)(gExtendedUnfilteredScan_c | gScanAllPDUs_c)) Scan Only Non Decision Advertising PDU.

gScanAllPDUs_c
: Scan Decision and Non Decision Advertising PDU.

gScanOnlyDecisionPDUs_c
: Scan Only Decision Advertising PDU.

gNetworkPrivacy_c
: Privacy mode values Use Network Privacy Mode for the peer device (default)

gDevicePrivacy_c
: Use Device Privacy Mode for the peer device

gUnenhancedBearerId_c
: bearer id 0 is used for the Unenhanced ATT bearer

gDHKeySize_c
: LE Secure Connections 256 bit DHKey

gSkdSize_c
: LL Session Key Diversfier size

gHCICSChannelMapSize
: CS Channel Map size

gCSReflectorTableSize_c
: CS reflector table size size

gCSNumPhysMax_c
: CS reflector table size size

gCSNumToneAntennaIds_c
: CS number of antenna IDs used in the pattern

gCSTestMaxChannelLength_c
: CS number of channels used in the the pattern

gDRBGKeySize_c
: CS debugger key size

gCSSyncRandomSize_c
: CS Sync Random size

gCSMaxSubeventLen_c
: CS maximum duration for each CS sub-event in microseconds

gCsAccessAdressSize_c
: CS Access Address used in SYNC packets

LL_PHY_1M

LL_PHY_2M

LL_PHY_S8

LL_PHY_S2

gVendorHandoverMaxCsLlContextSize_c
: Maximum size of the LL Context for Handover (CS context is largest)

gVendorUnitaryTestSize_c
: Maximum size of the VENDOR_UNITARY_TEST response

gLeExtendedFeaturesSize_c
: Current size of the leExtendedFeatures, can be raised up to 240

getLeExtendedFeatureByte(bitNumber)

getLeExtendedFeatureBit(bitNumber)

isSupportedLeExtendedFeature(features, bitNumber)

gBleSig_PrimaryService_d
: Bluetooth SIG UUID constants for GATT declarations

  Primary Service declaration UUID

gBleSig_SecondaryService_d
: Secondary Service declaration UUID

gBleSig_Include_d
: Include declaration UUID

gBleSig_Characteristic_d
: Characteristic declaration UUID

gBleSig_CharExtendedProperties_d
: Characteristic Extended Properties UUID

gBleSig_CharUserDescription_d
: Client Characteristic User Description UUID

gBleSig_CCCD_d
: Client Characteristic Configuration Descriptor declaration UUID

gBleSig_SCCD_d
: Server Characteristic Configuration Descriptor declaration UUID

gBleSig_CharPresFormatDescriptor_d
: Characteristic Presentation Format declaration UUID

gBleSig_CharAggregateFormat_d
: Characteristic Aggregate Format UUID

gBleSig_ValidRangeDescriptor_d
: Valid Range Descriptor declaration UUID

gBleSig_GenericAccessProfile_d
: GAP Service UUID

gBleSig_GenericAttributeProfile_d
: GATT Service UUID

gBleSig_ImmediateAlertService_d
: Immediate Alert Service UUID

gBleSig_LinkLossService_d
: Link Loss Service UUID

gBleSig_TxPowerService_d
: Tx Power Service UUID

gBleSig_CurrentTimeService_d
: Current Time Service UUID

gBleSig_ReferenceTimeUpdateService_d
: Reference Time Update Service UUID

gBleSig_NextDSTChangeService_d
: Next DST Change Service UUID

gBleSig_GlucoseService_d
: Glucose Service UUID

gBleSig_HealthThermometerService_d
: Health Thermometer Service UUID

gBleSig_DeviceInformationService_d
: Device Information Service UUID

gBleSig_HeartRateService_d
: Heart Rate Service UUID

gBleSig_PhoneAlertStatusService_d
: Phone Alert Status Service UUID

gBleSig_BatteryService_d
: Battery Service UUID

gBleSig_BloodPressureService_d
: Blood Pressure Service UUID

gBleSig_AlertNotificationService_d
: Alert Notification Service UUID

gBleSig_HidService_d
: HID Service UUID

gBleSig_RunningSpeedAndCadenceService_d
: Running Speed And Cadence Service UUID

gBleSig_CyclingSpeedAndCadenceService_d
: Cycling Speed And Cadence Service UUID

gBleSig_CyclingPowerService_d
: Cycling Power Service UUID

gBleSig_LocationAndNavigationService_d
: Location And Navigation Service UUID

gBleSig_IpsService_d
: Internet Protocol Support Service UUID

gBleSig_PulseOximeterService_d
: Pulse Oximeter Service UUID

gBleSig_HTTPProxyService_d
: HTTP Proxy Service UUID

gBleSig_WPTService_d
: Wireless Power Transfer Service UUID

gBleSig_BtpService_d
: BTP Service UUID

gBleSig_GapDeviceName_d
: GAP Device Name Characteristic UUID

gBleSig_GapAppearance_d
: GAP Appearance Characteristic UUID

gBleSig_GapPpcp_d
: GAP Peripheral Preferred Connection Parameters Characteristic UUID

gBleSig_GattServiceChanged_d
: GATT Service Changed Characteristic UUID

gBleSig_GattClientSupportedFeatures_d
: GATT Client Supported Features Characteristic UUID

gBleSig_GattServerSupportedFeatures_d
: GATT Server Supported Features Characteristic UUID

gBleSig_GattDatabaseHash_d
: GATT Database Hash Characteristic UUID

gBleSig_AlertLevel_d
: Alert Level Characteristic UUID

gBleSig_TxPower_d
: TX Power Characteristic UUID

gBleSig_LocalTimeInformation_d
: Local Time Information Characteristic UUID

gBleSig_TimeWithDST_d
: Time With DST Characteristic UUID

gBleSig_ReferenceTimeInformation_d
: Reference Time Information Characteristic UUID

gBleSig_TimeUpdateControlPoint_d
: Time Update Control Point Characteristic UUID

gBleSig_TimeUpdateState_d
: Time Update State Characteristic UUID

gBleSig_GlucoseMeasurement_d
: Glucose Measurement Characteristic UUID

gBleSig_BatteryLevel_d
: Battery Level Characteristic UUID

gBleSig_TemperatureMeasurement_d
: Temperature Measurement Characteristic UUID

gBleSig_TemperatureType_d
: Temperature Type Characteristic UUID

gBleSig_IntermediateTemperature_d
: Intermediate Temperature Characteristic UUID

gBleSig_MeasurementInterval_d
: Measurement Interval Characteristic UUID

gBleSig_SystemId_d
: System ID Characteristic UUID

gBleSig_ModelNumberString_d
: Model Number String Characteristic UUID

gBleSig_SerialNumberString_d
: Serial Number String Characteristic UUID

gBleSig_FirmwareRevisionString_d
: Firmware Revision String Characteristic UUID

gBleSig_HardwareRevisionString_d
: Hardware Revision String Characteristic UUID

gBleSig_SoftwareRevisionString_d
: Software Revision String Characteristic UUID

gBleSig_ManufacturerNameString_d
: Manufacturer Name String Characteristic UUID

gBleSig_IeeeRcdl_d
: IEEE 11073-20601 Regulatory Certification Data List Characteristic UUID

gBleSig_CurrentTime_d
: Current Time Characteristic UUID

gBleSig_BootKeyboardInputReport_d
: Boot Keyboard Input Report UUID

gBleSig_BootKeyboardOutputReport_d
: Boot Keyboard output Report UUID

gBleSig_BootMouseInputReport_d
: Boot Mouse Input Report UUID

gBleSig_GlucoseMeasurementContext_d
: Glucose Measurement Context Characteristic UUID

gBleSig_BpMeasurement_d
: Blood Pressure Measurement UUID

gBleSig_BpEnhancedMeasurement_d
: Blood Pressure Enhanced Measurement UUID

gBleSig_BpRecord_d
: Blood Pressure Record UUID

gBleSig_IntermediateCuffPressure_d
: Intermediate Cuff Pressure UUID

gBleSig_HrMeasurement_d
: Heart Rate Measurement UUID

gBleSig_BodySensorLocation_d
: Body Sensor Location UUID

gBleSig_HrControlPoint_d
: Heart Rate Control Point UUID

gBleSig_AlertStatus_d
: Alert Status UUID

gBleSig_RingerControlPoint_d
: Ringer Control Point UUID

gBleSig_RingerSetting_d
: Ringer Setting UUID

gBleSig_AlertNotifControlPoint_d
: Alert Notif Control Point UUID

gBleSig_UnreadAlertStatus_d
: Unread Alert Status UUID

gBleSig_NewAlert_d
: New Alert UUID

gBleSig_SupportedNewAlertCategory_d
: Supported New Alert Category UUID

gBleSig_SupportedUnreadAlertCategory_d
: Supported Unread Alert Category UUID

gBleSig_BloodPressureFeature_d
: Blood Pressure Feature UUID

gBleSig_HidInformation_d
: HID Information UUID

gBleSig_HidBootMouseInputReport_d
: HID Boot Mouse Input Report UUID

gBleSig_HidCtrlPoint_d
: HID Control Point UUID

gBleSig_Report_d
: Report UUID

gBleSig_ProtocolMode_d
: Protocol Mode UUID

gBleSig_ScanIntervalWindow_d
: Scan Interval Window UUID

gBleSig_PnpId_d
: PnP Id UUID

gBleSig_GlucoseFeature_d
: Glucose Feature Characteristic UUID

gBleSig_RaCtrlPoint_d
: Record Access Ctrl Point Characteristic UUID

gBleSig_RscMeasurement_d
: RSC Measurement UUID

gBleSig_RscFeature_d
: RSC Feature UUID

gBleSig_ScControlPoint_d
: SC Control Point UUID

gBleSig_CscMeasurement_d
: CSC Measurement Characteristic UUID

gBleSig_CscFeature_d
: CSC Feature Characteristic UUID

gBleSig_SensorLocation_d
: Sensor Location Characteristic UUID

gBleSig_PlxSCMeasurement_d
: PLX Spot-Check Measurement Characteristic UUID

gBleSig_PlxContMeasurement_d
: PLX Continuous Measurement Characteristic UUID

gBleSig_PulseOximeterFeature_d
: PLX Feature Characteristic UUID

gBleSig_CpMeasurement_d
: CP Measurement Characteristic UUID

gBleSig_CpVector_d
: CP Measurement Vector UUID

gBleSig_CpFeature_d
: CP Feature CharacteristicUUID

gBleSig_CpControlPoint_d
: CP Control Point UUID

gBleSig_LocationAndSpeed_d
: Location and Speed Characteristic UUID

gBleSig_Navigation_d
: Navigation Characteristic UUID

gBleSig_PositionQuality_d
: Position Quality Characteristic UUID

gBleSig_LnFeature_d
: LN Feature Characteristic UUID

gBleSig_LnControlPoint_d
: LN Control Point Characteristic UUID

gBleSig_Temperature_d
: Temperature Characteristic UUID

gBleSig_CentralAddressResolution_d
: Central Address Resolution Characteristic UUID

gBleSig_URI_d
: URI Characteristic UUID

gBleSig_HTTP_Headers_d
: HTTP Headers Characteristic UUID

gBleSig_HTTP_StatusCode_d
: HTTP Status Code Characteristic UUID

gBleSig_HTTP_EntityBody_d
: HTTP Entity Body Characteristic UUID

gBleSig_HTTP_ControlPoint_d
: HTTP Control Point Characteristic UUID

gBleSig_HTTPS_Security_d
: HTTPS Security Characteristic UUID

gBleSig_ResolvablePrivateAddressOnly_d
: Resolvable Private Address Only Characteristic UUID

gBleSig_MeshProvisioningService_d
: BLE Mesh Provisioning Service UUID

gBleSig_MeshProxyService_d
: BLE Mesh Proxy Service UUID

gBleSig_MeshProvDataIn_d
: BLE Mesh Prov Data In Char UUID

gBleSig_MeshProvDataOut_d
: BLE Mesh Prov Data Out Char UUID

gBleSig_MeshProxyDataIn_d
: BLE Mesh Proxy Data In Char UUID

gBleSig_MeshProxyDataOut_d
: BLE Mesh Proxy Data Out Char UUID

gBleSig_GattSecurityLevels_d
: LE GATT Security Levels characteristic UUID

gBleSig_EncryptedDataKeyMaterial_d
: Encrypted Data Key Material characteristic UUID

gBleSig_CAR_NotSupported_d
: Central Address Resolution Characteristic Values

gBleSig_CAR_Supported_d

gBleSig_RPAO_Used_d
: Resolvable Private Address Only Characteristic Values

gBleSig_RangingService_d
: Ranging Service UUID

gBleSig_RasFeature_d
: RAS Feature Characteristic UUID

gBleSig_RasRealTimeProcData_d
: RAS Real-time Procedure Data Characteristic UUID

gBleSig_RasOnDemandProcData_d
: RAS On-demand Procedure Data Characteristic UUID

gBleSig_RasControlPoint_d
: RAS Control Point Characteristic UUID

gBleSig_RasProcDataReady_d
: RAS Procedure Data Ready Characteristic UUID

gBleSig_RasprocDataOverwritten_d
: RAS Procedure Data Overwritten Characteristic UUID

BleSig_IsGroupingAttributeUuid16(uuid16)
: Macro that returns whether or not an input 16-bit UUID is a grouping type.

BleSig_IsServiceDeclarationUuid16(uuid16)
: Macro that returns whether or not an input 16-bit UUID is a Service declaration.

Uuid16(uuid)
: Macro that declares a 16 bit UUID in a *bleUuid_t* union.

Uuid32(uuid)
: Macro that declares a 32 bit UUID in a *bleUuid_t* union.

UuidArray(value)

PACKED_STRUCT
: Type qualifier - does not affect local variables of integral type

__noreturn
: Type qualifier - does not affect local variables of integral type

  Type qualifier - does not affect local variables of integral type

  Storage class modifier - alignment of a variable. It does not affect the type of the function

  Marks a function that never returns.

Utils_ExtractTwoByteValue(buf)
: Returns a uint16_t from a buffer, little-endian

Utils_ExtractThreeByteValue(buf)
: Returns a 3-byte value from a buffer, little-endian

Utils_ExtractFourByteValue(buf)
: Returns a uint32_t from a buffer, little-endian

Utils_ExtractEightByteValue(buf)
: Returns a uint64_t from a buffer, little-endian

Utils_BeExtractTwoByteValue(buf)
: Returns a uint16_t from a buffer, big-endian

Utils_BeExtractThreeByteValue(buf)
: Returns a 3-byte value from a buffer, big-endian

Utils_BeExtractFourByteValue(buf)
: Returns a uint32_t from a buffer, big-endian

Utils_PackTwoByteValue(value, buf)
: Writes a uint16_t into a buffer, little-endian

Utils_PackThreeByteValue(value, buf)
: Writes a 3-byte value into a buffer, little-endian

Utils_PackFourByteValue(value, buf)
: Writes a uint32_t into a buffer, little-endian

Utils_PackEightByteValue(value, buf)
: Writes a uint64_t into a buffer, little-endian

Utils_BePackTwoByteValue(value, buf)
: Writes a uint16_t into a buffer, big-endian

Utils_BePackThreeByteValue(value, buf)
: Writes a 3-byte value into a buffer, big-endian

Utils_BePackFourByteValue(value, buf)
: Writes a uint32_t into a buffer, big-endian

Utils_Copy8(ptr, val8)
: Writes a uint8_t into a buffer, little-endian, and increments the pointer.

Utils_Copy16(ptr, val16)
: Writes a uint16_t into a buffer, little-endian, and increments the pointer.

Utils_Copy32(ptr, val32)
: Writes a uint32_t into a buffer, little-endian, and increments the pointer.

Utils_Copy64(ptr, val64)
: Writes a uint64_t into a buffer, little-endian, and increments the pointer.

Utils_RevertByteArray(array, size)
: Reverts the order of bytes in an array - useful for changing the endianness

struct bleIdentityAddress_t
: *#include <ble_general.h>* Bluetooth Identity Address - array of 6 bytes.

Public Members

bleAddressType_t idAddressType
: Public or Random (static).

bleDeviceAddress_t idAddress
: 6-byte address.

union bleUuid_t
: *#include <ble_general.h>* Union for a Bluetooth UUID; selected according to an accompanying bleUuidType_t

Public Members

uint16_t uuid16
: For gBleUuidType16_c.

uint32_t uuid32
: For gBleUuidType32_c.

uint8_t uuid128[16]
: For gBleUuidType128_c.

struct bleAdvertisingChannelMap_t

#include <ble_general.h>

Public Members

uint8_t enableChannel37

Bit for channel 37.

uint8_t enableChannel38

Bit for channel 38.

uint8_t enableChannel39

Bit for channel 39.

uint8_t reserved

Reserved for future use.

struct gapLeScOobData_t

#include <ble_general.h>

Public Members

uint8_t randomValue[(16U)]

LE SC OOB r (Random value)

uint8_t confirmValue[(16U)]

LE SC OOB Cr (Random Confirm value)

struct gapInternalError_t

#include <ble_general.h> Internal Error Event Data

Public Members

bleResult_t errorCode

Host Stack error code.

gapInternalErrorSource_t errorSource

The command that generated the error; useful when it is not obvious from the error code.

uint16_t hciCommandOpcode

Only for errorSource = gHciCommandStatus_c; the HCI Command that received an error status.

struct gapControllerTestEvent_t

#include <ble_general.h> Controller Test Event

struct gapPhyEvent_t

#include <ble_general.h> Phy Event

struct bleNotificationEvent_t

#include <ble_general.h> Controller Enhanced Notification Event

Public Members

bleNotificationEventType_t eventType

Enhanced notification event type

deviceId_t deviceId

Device id of the peer, valid for connection events

int8_t rssi

RSSI, valid for Rx event types

uint8_t channel

Channel, valid for conn event over or Rx/Tx events

uint16_t ce_counter

Connection event counter, valid for conn event over or Conn Rx event

bleResult_t status

Status of the request to select which events to be enabled/disabled

uint32_t timestamp

Timestamp in microseconds, valid for Conn Rx event and Conn Created event

uint8_t adv_handle

Advertising Handle, valid for advertising events, if multiple ADV sets supported

bleDeviceAddress_t scanned_addr

Scanned address, valid for gNotifScanAdvPktRx_c event, all-zeroes otherwise

struct gapInitComplete_t

#include <ble_general.h> gInitializationComplete_c event data

struct bleBondCreatedEvent_tag

#include <ble_general.h> Bond Created Event

Public Members

uint8_t nvmIndex

NVM index for the new created bond

bleAddressType_t addressType

Public or Random (static) address of the bond

bleDeviceAddress_t address

Address of the bond

struct gapAddrReadyEvent_t_tag

#include <ble_general.h> Address Ready Event

Public Members

bleDeviceAddress_t aAddress

Generated device address

uint8_t advHandle

Advertising set handle if the generated device address will be used on an extended set. Reserved value 0xFF for other purposes: legacy advertising or scanning and initiating address.

struct bleSupportedSwitchingSamplingRates_t

#include <ble_general.h>

Public Members

uint8_t switchingSupportedAodTransmission

Bit to mark that 1 us switching is supported for AoD transmission.

uint8_t samplingSupportedAodReception

Bit to mark that 1 us sampling is supported for AoD reception.

uint8_t switchingSamplingSupportedAoaReception

Bit to mark that 1 us switching and sampling is supported for AoA reception.

uint8_t reserved

Reserved for future use.

struct bleAntennaInformation_tag

#include <ble_general.h> Antenna Information Read Event

struct periodicAdvSyncTransferEvent_tag

#include <ble_general.h>

struct periodicAdvSetInfoTransferEvent_tag

#include <ble_general.h>

struct periodicAdvSetSyncTransferParamsEvent_tag

#include <ble_general.h>

struct gapSyncTransferReceivedEventData_tag

#include <ble_general.h>

Public Members

deviceId_t deviceId

Id of the connected device from which we received periodic advertising sync transfer

uint16_t serviceData

A value provided by the peer device

uint16_t syncHandle

Sync Handle identifying the periodic advertising train

uint8_t advSID

Value of the Advertising SID used to advertise the periodic advertising

bleAddressType_t advAddressType

Device's advertising address type.

bleDeviceAddress_t advAddress

Device's advertising address.

gapLePhyMode_t advPhy

PHY used for advertising

uint16_t periodicAdvInt

Periodic advertising interval

bleCentralClockAccuracy_t advClockAccuracy

Advertiser clock accuracy

uint8_t numSubevents

Only relevant for v2 of the event, otherwise value is 0

uint8_t subeventInterval

Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotDelay

Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotSpacing

Only relevant for v2 of the event, otherwise value is 0

struct getConnParams_tag

#include <ble_general.h>

Public Members

uint16_t connectionHandle
: Connection identifier

uint32_t ulTxAccCode
: Access address

uint8_t aCrcInitVal[3U]
: CRC init

uint16_t uiConnInterval
: Connection interval (unit 1.25ms)

uint16_t uiSuperTO
: Supervision timeout (unit 10ms)

uint16_t uiConnLatency
: Latency (unit connection interval)

uint8_t aChMapBm[5U]
: Channel map (FF FF FF FF 1F if all channels are used)

uint8_t ucChannelSelection
: Hop algorithm (0/1 for hop algorithm #1 or #2)

uint8_t ucHop
: Hop increment

uint8_t ucUnMapChIdx
: Unmapped channel index (used only for hop algorithm #1)

uint8_t ucCentralSCA
: Sleep clock accuracy (0 to 7)

uint8_t ucRole
: Role (0 for central and 1 for peripheral)

uint8_t aucRemoteMasRxPHY
: TX/RX PHY

uint8_t seqNum
: Sequence number; b1 and b0: the latest received SN and NESN; b5 and b4: the latest transmitted SN and NESN; other fields are reserved.

uint16_t uiConnEvent
: Current connection event counter

uint32_t ulAnchorClk
: Slot of the anchor point timing of the connection event

uint16_t uiAnchorDelay
: Slot offset of the anchor point timing of the connection event

uint32_t ulRxInstant
: Last successful access address reception instant (unit 625us)

struct handoverGetTime_tag
: *#include <ble_general.h>*

Public Members

bleResult_t status
: Command status

uint32_t slot
: LL timing slot counter (unit 625us)

uint16_t us_offset
: LL timing slot offset (0 to 624, unit us)

struct handoverAnchorSearchStart_tag
: *#include <ble_general.h>*

Public Members

bleResult_t status
: Command status

uint16_t connectionHandle
: Connection identifier

struct handoverAnchorSearchStop_tag
: *#include <ble_general.h>*

Public Members

bleResult_t status
: Command status

uint16_t connectionHandle
: Connection identifier

struct handoverConnect_tag
: *#include <ble_general.h>*

Public Members

bleResult_t status
Command status

uint16_t connectionHandle
Connection identifier

struct handoverGetData_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint32_t *pData
Connection handover data

struct handoverSetData_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint32_t *pData
Connection handover data

struct handoverGetCsLlContext_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint16_t responseMask
LL context bitmap indicating the Channel Sounding context

uint8_t llContextLength
Context data length

uint8_t llContext[(224U)]
LL Context

struct handoverAnchorMonitorEvent_tag

#include <ble_general.h>

Public Members

uint16_t connectionHandle

Connection identifier

uint16_t connEvent

Current connection event counter

int8_t rssiRemote

RSSI of the packet from the remote device (+127 if not available)

uint8_t lqiRemote

LQI (Link Quality Indicator) of the packet from the remote device

uint8_t statusRemote

Status of the packet from the remote device; b0: NESN; b1: SN; b2: CRC status valid (1) or invalid (0); b3: RSSI status valid (1) or invalid (0)

int8_t rssiActive

RSSI of the packet from the active device (+127 if not available)

uint8_t lqiActive

LQI (Link Quality Indicator) of the packet from the active device

uint8_t statusActive

Status of the packet from the active device; b0: NESN; b1: SN; b2: CRC status valid (1) or invalid (0); b3: RSSI status valid (1) or invalid (0)

uint32_t anchorClock625Us

Slot of the anchor point timing of the connection event

uint16_t anchorDelay

Slot offset of the anchor point timing of the connection event

uint8_t chIdx

BLE channel index

uint8_t ucNbReports

Number of remaining reports including the current one; 0: continuous reporting; 1-255: remaining number of reports to receive including the current one

struct handoverAnchorMonitorPacketEvent_tag

#include <ble_general.h>

Public Members

uint8_t packetCounter
: Packet counter, incremented for each event

uint16_t connectionHandle
: Connection identifier

uint8_t statusPacket
: Status of the packet

uint8_t phy
: PHY (0/1/2/3 for 1M/2M/LR S8/LR S2)

uint8_t chIdx
: BLE channel index

int8_t rssiPacket
: RSSI of the packet (+127 if not available)

uint8_t lqiPacket
: LQI (Link Quality Indicator) of the packet

uint16_t connEvent
: Current connection event counter

uint32_t anchorClock625Us
: Slot value of packet start time (in 625us unit)

uint16_t anchorDelay
: Slot offset value of packet start time (in 1us unit)

uint8_t ucNbConnIntervals
: Number of remaining connection intervals to monitor including the current one; 0: continuous reporting; 1-255: remaining number of reports to receive including the current one

uint8_t pduSize
: PDU length

uint8_t *pPdu
: Full or first part of the data PDU including: PDU header, payload if present, MIC if present, CRC

struct handoverAnchorMonitorPacketContinueEvent_tag
: *#include <ble_general.h>*

Public Members

uint8_t packetCounter
: Packet counter, incremented for each event

uint16_t connectionHandle
: Connection identifier

uint8_t pduSize
: PDU length

uint8_t *pPdu
: Full or first part of the data PDU including: PDU header, payload if present, MIC if present, CRC

struct handoverConnectionUpdateProcedureEvent_tag
: *#include <ble_general.h>*

Public Members

uint16_t connectionHandle
: Connection identifier

uint8_t winSize
: Used to indicate the transmitWindowSize value as: transmitWindowSize = winSize * 1.25 ms

uint16_t winOffset
: Used to indicate the transmitWindowOffset value as: transmitWindowOffset = winOffset * 1.25 ms

uint16_t interval
: Used to indicate the connInterval value, as: connInterval = interval * 1.25 ms

uint16_t latency
: Used to indicate the connPeripheralLatency value as: connPeripheralLatency = latency

uint16_t timeout
: Used to indicate the connSupervisionTimeout value as: connSupervisionTimeout = timeout * 10 ms

uint16_t instant
: Connection event counter value indicating when the new connection parameters are applied

uint16_t currentEventCounter
: Current connection event counter

struct handoverTimeSyncEvent_tag
: *#include <ble_general.h>*

Public Members

uint32_t txClkSlot
: Transmitter packet start time in slot (625 us)

uint16_t txUs
: Transmitter packet start time offset inside the slot (0 to 624 us)

uint32_t rxClkSlot
: Receiver packet start time in slot (625 us)

uint16_t rxUs
: Receiver packet start time offset inside the slot (0 to 624 us)

uint8_t rssi
: RSSI

struct handoverConnParamUpdateEvent_tag
: *#include <ble_general.h>*

Public Members

uint8_t status
: Status indicating the event trigger source: bit 0: remote SN^NESN toggled; bit 1: local SN^NESN toggled; bit 2: RFU; bit 3: phy update procedure; bit 4: channel map update procedure

uint16_t connectionHandle
: Connection identifier

uint32_t ulTxAccCode
: Access address

uint8_t aCrcInitVal[3U]
: CRC init

uint16_t uiConnInterval
: Connection interval (unit 1.25ms)

uint16_t uiSuperTO
: Supervision timeout (unit 10ms)

uint16_t uiConnLatency
: Latency (unit connection interval)

uint8_t aChMapBm[5U]
: Channel map (FF FF FF FF 1F if all channels are used)

uint8_t ucChannelSelection

Hop algorithm (0/1 for hop algorithm #1 or #2)

uint8_t ucHop

Hop increment

uint8_t ucUnMapChIdx

Unmapped channel index (used only for hop algorithm #1)

uint8_t ucCentralSCA

Sleep clock accuracy (0 to 7)

uint8_t ucRole

Role (0 for central and 1 for peripheral)

uint8_t aucRemoteMasRxPHY

TX/RX PHY

uint8_t seqNum

Sequence number; b1 and b0: the latest received SN and NESN; b5 and b4: the latest transmitted SN and NESN; other fields are reserved.

uint16_t uiConnEvent

Current connection event counter

uint32_t ulAnchorClk

Slot of the anchor point timing of the connection event

uint16_t uiAnchorDelay

Slot offset of the anchor point timing of the connection event

uint32_t ulRxInstant

Last successful access address reception instant (unit 625us)

struct handoverSuspendTransmitCompleteEvent_tag

#include <ble_general.h>

Public Members

uint16_t connectionHandle

Connection identifier

uint8_t noOfPendingAclPackets

Number of pending ACL packets (including the packet being transmitted partially or fully but not fully acked)

uint16_t sizeOfPendingAclPackets
: Pending ACL packet data size including all the data in the oldest packet:
  - data transmitted but not acked in the oldest ACL packet
  - data transmitted and acked in the oldest ACL packet

uint16_t sizeOfDataTxInOldestPacket
: Size of data transmitted and acked in the oldest ACL packet

uint8_t sizeOfDataNAckInOldestPacket
: Size of data transmitted but not acked in the oldest ACL packet

struct handoverResumeTransmitCompleteEvent_tag
: *#include <ble_general.h>*

Public Members

uint16_t connectionHandle
: Connection identifier

struct handoverUpdateConnParams_tag
: *#include <ble_general.h>*

Public Members

bleResult_t status
: Command status

uint16_t connectionHandle
: Connection identifier

struct handoverApplyConnectionUpdateProcedure_tag
: *#include <ble_general.h>*

Public Members

uint16_t connectionHandle
: Connection identifier

struct handoverAnchorNotificationStateChanged_tag
: *#include <ble_general.h>*

Public Members

uint16_t connectionHandle
Connection identifier

struct handoverLlPendingDataIndication_tag
#include <ble_general.h>

Public Members

uint16_t dataSize
Pending HCI ACL data packet size

uint8_t *pData
Pointer to message containing the HCI ACL data packet. Should be free by the application using MSG_Free()

struct gapLEGenerateDhKeyCompleteEvent_t
#include <ble_general.h>

struct gapRemoteVersionInfoRead_tag
#include <ble_general.h>

struct gapLlSkdReport_tag
#include <ble_general.h>

Public Members

deviceId_t deviceId
Peer device identifier

uint8_t aSKD[(16U)]
LL SKD

struct vendorUnitaryTestEvent_tag
#include <ble_general.h>

struct gapGenericEvent_t
#include <ble_general.h> Generic Event Structure = type + data

Public Members

gapGenericEventType_t eventType
Event type.

union *gapGenericEvent_t* eventData
Event data, selected according to event type.

struct bleBondIdentityHeaderBlob_t
#include <ble_general.h>

struct bleBondDataDynamicBlob_t
#include <ble_general.h>

struct bleBondDataStaticBlob_t
#include <ble_general.h>

struct bleBondDataLegacyBlob_t
#include <ble_general.h>

struct bleBondDataDeviceInfoBlob_t
#include <ble_general.h>

struct bleBondDataDescriptorBlob_t
#include <ble_general.h>

struct bleLocalKeysBlob_t
#include <ble_general.h>

struct bleBondDataBlob_t
#include <ble_general.h>

struct bleBondDataRam_t
#include <ble_general.h>

struct bleCteAllowedTypesMap_t
#include <ble_general.h>

Public Members

uint8_t allowAoA
Bit to allow AoA CTE Response.

uint8_t allowAoD1us
Bit to allow AoD CTE Response with 1 microsecond slots.

uint8_t allowAoD2us
Bit to allow AoD CTE Response with 2 microsecond slots.

uint8_t reserved
Reserved for future use.

struct bleSyncCteType_tag
#include <ble_general.h>

Public Members

uint8_t doNotSyncWithAoA
: Bit for CTE type AoA.

uint8_t doNotSyncWithAoD1us
: Bit for CTE type AoD 1us.

uint8_t doNotSyncWithAoD2us
: Bit for CTE type AoD 2us.

uint8_t doNotSyncWithType3
: Bit for CTE type 3 (reserved).

uint8_t doNotSyncWithoutCte
: Bit for no CTE type.

uint8_t reserved
: Reserved for future use.

struct bleTxPowerLevelFlags_t
: *#include <ble_general.h>*

Public Members

uint8_t minimum
: Tx Power Level is at minimum level.

uint8_t maximum
: Tx Power Level is at maximum level.

uint8_t reserved
: Reserved for future use.

union eventData

Public Members

gapInternalError_t internalError
: Data for the gInternalError_c event. The error that has occurred and the command that triggered it.

uint8_t filterAcceptListSize
: Data for the gFilterAcceptListSizeRead_c event. The size of the Filter Accept List.

bleDeviceAddress_t aAddress
: Data for the gPublicAddressRead_c event. Contains the requested device address.

gapAddrReadyEvent_t addrReady

Data for the gRandomAddressReady_c event. Contains the generated device address and advertising handle if applicable (0xFF otherwise).

uint8_t advHandle

Data for the gRandomAddressSet_c event. Contains the handle of the configured advertising set or 0xFF for legacy advertising.

bleResult_t setupFailError

Data for the gAdvertisingSetupFailed_c event. The error that occurred during the advertising setup.

int8_t advTxPowerLevel_dBm

Data for the gExtAdvertisingParametersSetupComplete_c and gAdvTxPowerLevelRead_c events. Value in dBm.

bool_t verified

Data for the gPrivateResolvableAddressVerified_c event. TRUE if the PRA was resolved with the given IRK.

gapLeScOobData_t localOobData

Data for the gLeScLocalOobData_c event. Contains local OOB data for LESC Pairing.

bool_t newHostPrivacyState

Data for the gHostPrivacyStateChanged_c event. TRUE if enabled, FALSE if disabled.

bool_t newControllerPrivacyState

Data for the gControllerPrivacyStateChanged_c event. TRUE if enabled, FALSE if disabled.

gapControllerTestEvent_t testEvent

Data for the gControllerTestEvent_c event. Contains test event type and received packets.

bleResult_t txPowerLevelSetStatus

Data for the gTxPowerLevelSetComplete_c event. Status of the set request.

gapPhyEvent_t phyEvent

Data for the gLePhyEvent_c event. Contains Tx and Rx Phy for a connection.

deviceId_t deviceId

Data for the gTxEntryAvailable_c event.

gapInitComplete_t initCompleteData

Data for the gInitializationComplete_c event. Contains the supported features, number of advertising sets and the size of the periodic advertiser list

bleNotificationEvent_t notifEvent

Data for the gControllerNotificationEvent_c event. Contains status and adv/scan/conn event data.

bleBondCreatedEvent_t bondCreatedEvent

Data for the gBondCreatedEvent_c event. Contains the NVM index and the address of the bond.

bleDeviceAddress_t aControllerLocalRPA

Data for the gControllerLocalRPARead_c event. Contains the requested device address.

getConnParams_t getConnParams

Data for the gGetConnParamsComplete_c event

uint16_t syncHandle

Data for the gConnectionlessIqSamplingStateChanged_c event. Contains the sync handle for the advertising train.

bleAntennaInformation_t antennaInformation

Data for the gAntennaInformationRead_c. Contains antenna information.

bleResult_t perAdvSyncTransferEnable

Data for the gPeriodicAdvRecvEnableComplete_c event

periodicAdvSyncTransferEvent_t perAdvSyncTransfer

Data for the gPeriodicAdvSyncTransferComplete_c event

periodicAdvSetInfoTransferEvent_t perAdvSetInfoTransfer

Data for the gPeriodicAdvSetInfoTransferComplete_c event

periodicAdvSetSyncTransferParamsEvent_t perAdvSetSyncTransferParams

Data for the gSetPeriodicAdvSyncTransferParamsComplete_c event

bleResult_t perAdvSetDefaultPerAdvSyncTransferParams

Data for the gSetDefaultPeriodicAdvSyncTransferParamsComplete_c event

gapSyncTransferReceivedEventData_t perAdvSyncTransferReceived

Data for the gPeriodicAdvSyncTransferReceived_c event

gapLEGenerateDhKeyCompleteEvent_t leGenerateDhKeyCompleteEvent

Data for gConnEvtLeGenerateDhKeyComplete_c: new key value

uint16_t pawrSyncHandle

Data for gPeriodicAdvSetResponseDataComplete_c and gPeriodicSyncSubeventComplete_c

uint8_t pawrAdvHandle

Data for gPeriodicAdvSetSubeventDataComplete_c

handoverGetData_t handoverGetData

Data for the gHandoverGetComplete_c event

handoverSetData_t handoverSetData

Data for the gHandoverSetComplete_c event

handoverGetCsLlContext_t handoverGetCsLlContext

Data for the gHandoverGetCsLlContextComplete_c event

handoverGetTime_t handoverGetTime

Data for the gHandoverGetTime_c event

handoverConnect_t handoverConnect

Data for the gHandoverConnect_c event

handoverAnchorSearchStart_t handoverAnchorSearchStart

Data for the gHandoverAnchorSearchStarted_c event

handoverAnchorSearchStop_t handoverAnchorSearchStop

Data for the gHandoverAnchorSearchStopped_c event

handoverAnchorMonitorEvent_t handoverAnchorMonitor

Data for the gHandoverAnchorMonitorEvent_c event

handoverTimeSyncEvent_t handoverTimeSync

Data for the gHandoverTimeSyncEvent_c event

handoverSuspendTransmitCompleteEvent_t handoverSuspendTransmitComplete

Data for the gHandoverSuspendTransmitComplete_c event

handoverResumeTransmitCompleteEvent_t handoverResumeTransmitComplete

Data for the gHandoverResumeTransmitComplete_c event

handoverAnchorNotificationStateChanged_t handoverAnchorNotificationStateChanged

Data for the gHandoverAnchorNotificationStateChanged_c event

handoverConnParamUpdateEvent_t handoverConnParamUpdate

Data for the gHandoverConnParamUpdateEvent_c event

gapRemoteVersionInfoRead_t gapRemoteVersionInfoRead

Data for the hciReadRemoteVersionInformationCompleteEvent_t event

gapLlSkdReport_t gapSkdReport

Data for the gLlSkdReportEvent_c event

handoverAnchorMonitorPacketEvent_t handoverAnchorMonitorPacket

Data for the gHandoverAnchorMonitorPacketEvent_c event. pPdu must be freed by the application

handoverAnchorMonitorPacketContinueEvent_t handoverAnchorMonitorPacketContinue

Data for the gHandoverAnchorMonitorPacketContinueEvent_c event. pPdu must be freed by the application

handoverUpdateConnParams_t handoverUpdateConnParams
: Data for the gHandoverUpdateConnParamsComplete_c event

handoverLlPendingDataIndication_t handoverLlPendingDataIndication
: Data for the gHandoverLlPendingData_c event

handoverConnectionUpdateProcedureEvent_t handoverConnectionUpdateProcedure
: Data for the gHandoverConnectionUpdateProcedureEvent_c event

handoverApplyConnectionUpdateProcedure_t handoverApplyConnectionUpdateProcedure
: Data for the gHandoverApplyConnectionUpdateProcedureComplete_c event

vendorUnitaryTestEvent_t unitaryTestData
: Data for the gVendorUnitaryTestComplete_c event

uint8_t monAdvListSize
: Data for the gMonAdvListSizeRead_c event. The size of the Monitored Advertisers List.

union ___unnamed2___

Public Members

uint32_t raw[((40U) + 3U) / sizeof(uint32_t)]

uint8_t pKey[(40U)]

Bluetooth configuration

gcBleDeviceAddressSize_c
: Size of a BLE Device Address

gBleBondDataDynamicSize_c
: Size of bond data structures for a bonded device

gBleBondDataStaticSize_c

gBleBondDataLegacySize_c

gBleBondDataDeviceInfoSize_c

gBleBondDataDescriptorSize_c

gcSecureModeSavedLocalKeysNo_c

gcGapMaximumSavedCccds_c
: Maximum number of CCCDs

gcGapMaxAuthorizationHandles_c
: Maximum number of attributes that require authorization

gBleBondDataSize_c
: Bonding Data Size

gcBleLongUuidSize_c
: Size of long UUIDs

gcSmpMaxLtkSize_c
: Maximum Long Term Key size in bytes

gcSmpMaxBlobSize_c
: Maximum Long Term Key blob size in bytes

gcSmpIrkSize_c
: Identity Resolving Key size in bytes

gcSmpMaxIrkBlobSize_c
: Maximum Identity Resolving Key blob size in bytes

gcSmpCsrkSize_c
: Connection Signature Resolving Key size in bytes

gcSmpMaxRandSize_c
: Maximum Rand size in bytes

gcSmpOobSize_c
: SMP OOB size in bytes

gSmpLeScRandomValueSize_c
: SMP LE Secure Connections Pairing Random size in bytes

gSmpLeScRandomConfirmValueSize_c
: SMP LE Secure Connections Pairing Confirm size in bytes

gcGapMaxDeviceNameSize_c
: Maximum device name size

gcGapMaxAdvertisingDataLength_c
: Maximum size of advertising and scan response data

gAttDefaultMtu_c
: Default value of the ATT_MTU

gAttMaxMtu_c
: Maximum possible value of the ATT_MTU for this device. This is used during the MTU Exchange.

gEattMaxMtu_c
: Maximum possible value of the ATT_MTU for enhanced ATT bearers on this device.

gAttMaxValueLength_c
: The maximum length of an attribute value shall be 512 octets.

gEattMinMtu_c
: Minimum MTU size for enhanced ATT channels is 64.

gHciTransportUartChannel_c
: Channel the number of the UART hardware module (For example, if UART1 is used, this value should be 1).

gcReservedFlashSizeForCustomInformation_c
: Number of bytes reserved for storing application-specific information about a device

gcBleChannelMapSize_c
: Size of a channel map in a connection

gBleMinTxOctets_c

gBleMinTxTime_c

gBleMaxTxOctets_c

gBleMaxTxTime_c

gBleMaxTxTimeCodedPhy_c

gBleCteMinTxOctets_c

gBleCteMinTxTime_c

gBleCteMaxTxOctets_c

gBleCteMaxTxTime_c

gBleExtAdvMaxSetId_c
: Maximum value of the advertising SID.

gBlePeriodicAdvMaxSyncHandle_c
: Maximum value of the periodic advertising handle.

gBleExtAdvLegacySetId_c
: SID of the legacy advertising set.

gBleExtAdvLegacySetHandle_c
: Handle of the legacy advertising set.

gBleExtAdvDefaultSetId_c
: Default SID for extended advertising.

gBleExtAdvDefaultSetHandle_c
: Default handle for extended advertising.

gBleAdvTxPowerNoPreference_c
: Host has no preference for Tx Power

gBleExtAdvNoDuration_c
: No advertising duration. Advertising to continue until the Host disables it.

gBleHighDutyDirectedAdvDuration
: Default advertising duration in high duty directed advertising 1.28s = 1280ms/10ms(unit) = 128

gBleExtAdvNoMaxEvents_c
: No maximum number of advertising events.

gBlePeriodicAdvDefaultHandle_c
: Periodic advertising default handle.

gBlePeriodicAdvSyncTimeoutMin_c
: Minimum value for the sync_timeout parameter

gBlePeriodicAdvSyncTimeoutMax_c
: Maximum value for the sync_timeout parameter

gBlePeriodicAdvSkipMax_c
: Maximum value for the skip parameter

gBleMaxADStructureLength_c
: Maximum length of an AD structure

gBleMaxExtAdvDataLength_c
: Maximum length of Extended Advertising Data

gBleExtAdvMaxAuxOffsetUsec_c
: Maximum value in us of AUX Offset(13 bits) in AuxPtr in 300us units, i.e. ((1 « 13) - 1) * 300

gBleP256KeyLength_c
: Length of P-256 key in octets

gcDecisionDataKeySize_c
: decision data key size in bytes

gcDecisionInstructionsParamSize_c
: decision instructions parameter size in bytes

gcDecisionInstructionsArbitraryDataMaskSize_c

decision instructions arbitrary data mask size in bytes

gcDecisionInstructionsArbitraryDataTargetSize_c

decision instructions arbitrary data target size in bytes

gcDecisionDataPrandSize_c

decision data random part size in bytes

gcDecisionDataMaxSize_c

decision data maximum size in bytes

gcDecisionDataResolvableTagSize_c

decision data maximum size in bytes

gcMaxNumTestsInDecisionInstruction_c

The maximum number of tests in the Gap_SetDecisionInstructions ((max param length) - sizeof(numTests))/((sizeof(TestFlag) + sizeof(TestField) + sizeof(TestParameters)) = (255-1)/(1 + 1 + 16)

gcEadRandomizerSize_c

Size of the Encrypted Advertising Data randomizer in bytes

gcEadMicSize_c

Size of the Encrypted Advertising Data MIC in bytes

gcEadIvSize_c

Size of the Encrypted Advertising Data Initialization Vector in bytes

gcEadKeySize_c

Size of the Encrypted Advertising Data Key in bytes

gcEadAadValue_c

Value of the Additional Authentication Data field for the CCM algorithm used by EAD

gBleMonAdvRssiThresholdMin_c

Minimum RSSI threshold value for Monitored Advertisers

gBleMonAdvRssiThresholdMax_c

Maximum RSSI threshold value for Monitored Advertisers

L2CA

enum l2caLeCbConnectionRequestResult_t

Values:

enumerator gSuccessful_c

enumerator gLePsmNotSupported_c

enumerator gNoResourcesAvailable_c

enumerator gInsufficientAuthentication_c

enumerator gInsufficientAuthorization_c

enumerator gInsufficientEncryptionKeySize_c

enumerator gInsufficientEncryption_c

enumerator gInvalidSourceCid_c

enumerator gSourceCidAreadyAllocated_c

enumerator gUnacceptableParameters_c

enumerator gInvalidParameters_c

enumerator gCommandRejected_c

enumerator gResponseTimeout_c

enum l2caErrorSource_t

Values:

enumerator gL2ca_CancelConnection_c

enumerator gL2ca_SendLeFlowControlCredit_c

enumerator gL2ca_DisconnectLePsm_c

enumerator gL2ca_HandleSendLeCbData_c

enumerator gL2ca_HandleRecvLeCbData_c

enumerator gL2ca_HandleLeFlowControlCredit_c

enumerator gL2ca_EnhancedReconfigureReq_c

enumerator gL2ca_EnhancedCancelConnection_c

enum l2capReconfigureResponse_t

Values:

enumerator gReconfigurationSuccessful_c

enumerator gMtuReductionNotAllowed_c

enumerator gMultipleChannelMpsReductionNotAllowed_c

enumerator gDestinationCidInvalid_c

enumerator gUnacceptableReconfParameters_c

enumerator gReconfigurationTimeout_c

enumerator gReconfReserved_c

enum l2caChannelStatus_t

Values:

enumerator gL2ca_ChannelStatusChannelIdle_c

enumerator gL2ca_ChannelStatusChannelBusy_c

enum l2capControlMessageType_t

Values:

enumerator gL2ca_LePsmConnectRequest_c

enumerator gL2ca_LePsmConnectionComplete_c

enumerator gL2ca_LePsmDisconnectNotification_c

enumerator gL2ca_NoPeerCredits_c

enumerator gL2ca_LowPeerCredits_c

enumerator gL2ca_LocalCreditsNotification_c

enumerator gL2ca_Error_c

enumerator gL2ca_ChannelStatusNotification_c

enumerator gL2ca_LePsmEnhancedConnectRequest_c

enumerator gL2ca_LePsmEnhancedConnectionComplete_c

enumerator gL2ca_EnhancedReconfigureRequest_c

enumerator gL2ca_EnhancedReconfigureResponse_c

enumerator gL2ca_HandoverConnectionComplete_c

typedef struct *l2caLeCbConnectionRequest_tag* l2caLeCbConnectionRequest_t

typedef struct *l2caLeCbConnectionComplete_tag* l2caLeCbConnectionComplete_t

typedef struct *l2caHandoverConnectionComplete_tag* l2caHandoverConnectionComplete_t

typedef struct *l2caLeCbDisconnection_tag* l2caLeCbDisconnection_t

typedef struct *l2caLeCbNoPeerCredits_tag* l2caLeCbNoPeerCredits_t

typedef struct *l2caLeCbLowPeerCredits_tag* l2caLeCbLowPeerCredits_t

typedef struct *l2caLeCbLocalCreditsNotification_tag* l2caLeCbLocalCreditsNotification_t

typedef struct *l2caLeCbError_tag* l2caLeCbError_t

typedef struct *l2caLeCbChannelStatusNotification_tag* l2caLeCbChannelStatusNotification_t

typedef struct *l2caEnhancedConnectionRequest_tag* l2caEnhancedConnectionRequest_t

typedef struct *l2caEnhancedConnectionComplete_tag* l2caEnhancedConnectionComplete_t

typedef struct *l2caEnhancedReconfigureRequest_tag* l2caEnhancedReconfigureRequest_t

typedef struct *l2caEnhancedReconfigureResponse_tag* l2caEnhancedReconfigureResponse_t

typedef struct *l2capControlMessage_tag* l2capControlMessage_t

typedef void (*l2caLeCbDataCallback_t)(*deviceId_t* deviceId, uint16_t channelId, uint8_t *pPacket, uint16_t packetLength)

typedef void (*l2caControlCallback_t)(*l2capControlMessage_t* *pMessage)

typedef *l2caControlCallback_t* l2caLeCbControlCallback_t

typedef void (*l2caGenericCallback_t)(*deviceId_t* deviceId, uint8_t *pPacket, uint16_t packetLength)

typedef void (*l2caEattCallback_t)(*deviceId_t* deviceId, uint16_t channelId, uint8_t *pPacket, uint16_t packetLength)

bleResult_t L2ca_RegisterLeCbCallbacks(*l2caLeCbDataCallback_t* pCallback, *l2caLeCbControlCallback_t* pCtrlCallback)

Registers callbacks for credit based data and control events on L2CAP.

Parameters

- pCallback – **[in]** Callback function for data plane messages.
- pCtrlCallback – **[in]** Callback function for control plane messages.

Return values

- gL2caSuccess_c – Successful callbacks installation.
- gL2caCallbackAlreadyInstalled_c – Callbacks already installed.

bleResult_t L2ca_RegisterLePsm(uint16_t lePsm, uint16_t lePsmMtu)

Registers the LE_PSM from the L2CAP.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- lePsmMtu – **[in]** MTU of the registered PSM.

Return values

- gL2caLePsmInsufficientResources_c – No free LE_PSM registration slot was found.
- gL2caLePsmInvalid_c – Invalid LE_PSM value.
- gL2caLePsmAlreadyRegistered_c – LE_PSM already registered.
- gL2caSuccess_c – Successful LE_PSM registration.

bleResult_t L2ca_DeregisterLePsm(uint16_t lePsm)

Unregisters the LE_PSM from the L2CAP.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.

Return values

- gL2caSuccess_c – Successful LE_PSM deregistration.
- gL2caLePsmNotRegistered_c – LE_PSM value not registered.
- gL2caInternalError_c – There are connected L2CAP channels for the LE_PSM

Pre

An LE_PSM must be registered a priori.

bleResult_t L2ca_ConnectLePsm(uint16_t lePsm, *deviceId_t* deviceId, uint16_t initialCredits)

Initiates a connection with a peer device for a registered LE_PSM.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- initialCredits – **[in]** Initial credits.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

Pre
An LE_PSM must be registered a priori.

bleResult_t L2ca_DisconnectLeCbChannel(*deviceId_t* deviceId, uint16_t channelId)
Disconnects a peer device for a registered LE_PSM.

Remark

Once this command is issued, all incoming data in transit for this device shall be discarded and any new additional outgoing data shall be discarded.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

Pre
A connection must have already been created.

bleResult_t L2ca_CancelConnection(uint16_t lePsm, *deviceId_t* deviceId,
l2caLeCbConnectionRequestResult_t refuseReason)
Terminates an L2CAP channel.

Remark

This interface can be used for a connection pending creation.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- refuseReason – **[in]** Reason to refuse the channel creation.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

bleResult_t L2ca_SendLeCbData(*deviceId_t* deviceId, uint16_t channelId, const uint8_t *pPacket,
uint16_t packetLength)
Sends a data packet through a Credit-Based Channel.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.

- pPacket – **[in]** Data buffer to be transmitted.
- packetLength – **[in]** Length of the data buffer.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleInvalidParameter_c – Invalid Device Id.
- gBleOverflow_c – Tx queue of the device is not empty.
- gBleOutOfMemory_c – Message or packet allocation fail.

Pre

An L2CAP Credit Based connection must be in place.

bleResult_t L2ca_SendLeCredit(*deviceId_t* deviceId, uint16_t channelId, uint16_t credits)

Sends credits to a device when capable of receiving additional LE-frames.

Parameters

- deviceId – **[in]** The DeviceID to which credits are given.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.
- credits – **[in]** Number of credits to be given.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id.

Pre

An L2CAP Credit Based connection must be in place.

bleResult_t L2ca_EnhancedConnectLePsm(uint16_t lePsm, *deviceId_t* deviceId, uint16_t mtu, uint16_t initialCredits, uint8_t noOfChannels, uint16_t *aCids)

Initiates a connection with a peer device for a registered LE_PSM. Enhanced mode: up to 5 channels can be opened.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- initialCredits – **[in]** Initial credits.
- noOfChannels – **[in]** Number of channels to request for opening. Max 5.
- aCids – **[in]** List of cids if an Enhanced Connection Request has been previously received. NULL if not.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id or invalid noOfChannels.

Pre

A LE_PSM must be registered a priori.

bleResult_t L2ca_EnhancedChannelReconfigure(*deviceId_t* deviceId, uint16_t newMtu, uint16_t newMps, uint8_t noOfChannels, uint16_t *aCids)

Reconfigures up to 5 channels with new values for MTU and/or MPS.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- newMtu – **[in]** New MTU value to be configured.
- newMps – **[in]** New MPS value to be configured.
- noOfChannels – **[in]** Number of channels to reconfigure. Max 5.
- *aCids – **[in]** The list of CIDs (endpoints on local device).

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id or invalid noOfChannels.

bleResult_t L2ca_EnhancedCancelConnection(uint16_t lePsm, *deviceId_t* deviceId, *l2caLeCbConnectionRequestResult_t* refuseReason, uint8_t noOfChannels, uint16_t *aCids)

Termination of pending L2CAP channels.

Remark

This interface can be used for an enhanced connection pending creation.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- refuseReason – **[in]** Reason to refuse the channel creation.
- noOfChannels – **[in]** Number of channels to disconnect. Max 5.
- *aCids – **[in]** The list of CIDs (endpoints on local device).

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id, invalid noOfChannels or invalid aCids pointer.

gL2capCidNull_c

gL2capCidAtt_c

gL2capCidSignaling_c

gL2capCidSmp_c

gL2capCidSigAssignedFirst_c

gL2capCidSigAssignedLast_c

gL2capCidLePsmDynamicFirst_c

gL2capCidLePsmDynamicLast_c

gL2capCidNotApplicable_c

gL2caLePsmSigAssignedFirst_c

gL2caLePsmSigAssignedEatt_c

gL2caLePsmSigAssignedLast_c

gL2caLePsmDynamicFirst_c

gL2caLePsmDynamicLast_c

gL2capDefaultMtu_c

gL2capDefaultMps_c

gL2capMaximumMps_c

gEnhancedL2capMinimumMps_c

gL2capHeaderLength_c

gExpandAsEnum_m(a, b, c)

gExpandAsTable_m(a, b, c)

gL2capEnhancedMaxChannels_c

gLePsmSigAssignedNumbersTable_m(entry)

struct l2caLeCbConnectionRequest_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbConnectionComplete_tag
#include <l2ca_cb_interface.h>

struct l2caHandoverConnectionComplete_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbDisconnection_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbNoPeerCredits_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbLowPeerCredits_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbLocalCreditsNotification_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbError_tag
#include <l2ca_cb_interface.h>

struct l2caLeCbChannelStatusNotification_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedConnectionRequest_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedConnectionComplete_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedReconfigureRequest_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedReconfigureResponse_tag
#include <l2ca_cb_interface.h>

struct l2capControlMessage_tag
#include <l2ca_cb_interface.h>

union messageData

Public Members

l2caLeCbConnectionRequest_t connectionRequest

l2caLeCbConnectionComplete_t connectionComplete

l2caLeCbDisconnection_t disconnection

l2caLeCbNoPeerCredits_t noPeerCredits

l2caLeCbLowPeerCredits_t lowPeerCredits

l2caLeCbLocalCreditsNotification_t localCreditsNotification

l2caLeCbError_t error

l2caLeCbChannelStatusNotification_t channelStatusNotification

l2caEnhancedConnectionRequest_t enhancedConnRequest

l2caEnhancedConnectionComplete_t enhancedConnComplete

l2caEnhancedReconfigureRequest_t reconfigureRequest

l2caEnhancedReconfigureResponse_t reconfigureResponse

l2caHandoverConnectionComplete_t handoverConnectionComplete

Files

ble_constants.h

gap_interface.h

ble_general.h

struct bleIdentityAddress_t

Public Members

bleAddressType_t idAddressType

bleDeviceAddress_t idAddress

union bleUuid_t

Public Members

uint16_t uuid16

uint32_t uuid32

uint8_t uuid128[16]

struct bleAdvertisingChannelMap_t

Public Members

uint8_t enableChannel37

uint8_t enableChannel38

uint8_t enableChannel39

uint8_t reserved

struct gapLeScOobData_t

Public Members

uint8_t randomValue[(16U)]

uint8_t confirmValue[(16U)]

struct gapInternalError_t

Public Members

bleResult_t errorCode

gapInternalErrorSource_t errorSource

uint16_t hciCommandOpcode

struct gapControllerTestEvent_t

Public Members

gapControllerTestEventType_t testEventType

uint16_t receivedPackets

struct gapPhyEvent_t

Public Members

gapPhyEventType_t phyEventType

deviceId_t deviceId

uint8_t txPhy

uint8_t rxPhy

struct bleNotificationEvent_t

Public Members

bleNotificationEventType_t eventType

deviceId_t deviceId

int8_t rssi

uint8_t channel

uint16_t ce_counter

bleResult_t status

uint32_t timestamp

uint8_t adv_handle

bleDeviceAddress_t scanned_addr

struct gapInitComplete_t

Public Members

leSupportedFeatures_t supportedFeatures

uint8_t leExtendedFeatures[(1U)]

uint16_t maxAdvDataSize

uint8_t numOfSupportedAdvSets

uint8_t periodicAdvListSize

struct bleBondCreatedEvent_tag

Public Members

uint8_t nvmIndex

bleAddressType_t addressType

bleDeviceAddress_t address

struct gapAddrReadyEvent_t_tag

Public Members

bleDeviceAddress_t aAddress

uint8_t advHandle

struct bleSupportedSwitchingSamplingRates_t

Public Members

uint8_t switchingSupportedAodTransmission

uint8_t samplingSupportedAodReception

uint8_t switchingSamplingSupportedAoaReception

uint8_t reserved

struct bleAntennaInformation_tag

Public Members

bleSupportedSwitchingSamplingRates_t supportedSwitchingSamplingRates

uint8_t numAntennae

uint8_t maxSwitchingPatternLength

uint8_t maxCteLength

struct periodicAdvSyncTransferEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct periodicAdvSetInfoTransferEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct periodicAdvSetSyncTransferParamsEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct gapSyncTransferReceivedEventData_tag

Public Members

bleResult_t status

deviceId_t deviceId

uint16_t serviceData

uint16_t syncHandle

uint8_t advSID

bleAddressType_t advAddressType

bleDeviceAddress_t advAddress

gapLePhyMode_t advPhy

uint16_t periodicAdvInt

bleCentralClockAccuracy_t advClockAccuracy

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

struct getConnParams_tag

Public Members

uint16_t connectionHandle

uint32_t ulTxAccCode

uint8_t aCrcInitVal[3U]

uint16_t uiConnInterval

uint16_t uiSuperTO

uint16_t uiConnLatency

uint8_t aChMapBm[5U]

uint8_t ucChannelSelection

uint8_t ucHop

uint8_t ucUnMapChIdx

uint8_t ucCentralSCA

uint8_t ucRole

uint8_t aucRemoteMasRxPHY

uint8_t seqNum

uint16_t uiConnEvent

uint32_t ulAnchorClk

uint16_t uiAnchorDelay

uint32_t ulRxInstant

struct handoverGetTime_tag

Public Members

bleResult_t status

uint32_t slot

uint16_t us_offset

struct handoverAnchorSearchStart_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverAnchorSearchStop_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverConnect_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverGetData_tag

Public Members

bleResult_t status

uint32_t *pData

struct handoverSetData_tag

Public Members

bleResult_t status

uint32_t *pData

struct handoverGetCsLlContext_tag

Public Members

bleResult_t status

uint16_t responseMask

uint8_t llContextLength

uint8_t llContext[(224U)]

struct handoverAnchorMonitorEvent_tag

Public Members

uint16_t connectionHandle

uint16_t connEvent

int8_t rssiRemote

uint8_t lqiRemote

uint8_t statusRemote

int8_t rssiActive

uint8_t lqiActive

uint8_t statusActive

uint32_t anchorClock625Us

uint16_t anchorDelay

uint8_t chIdx

uint8_t ucNbReports

struct handoverAnchorMonitorPacketEvent_tag

Public Members

uint8_t packetCounter

uint16_t connectionHandle

uint8_t statusPacket

uint8_t phy

uint8_t chIdx

int8_t rssiPacket

uint8_t lqiPacket

uint16_t connEvent

uint32_t anchorClock625Us

uint16_t anchorDelay

uint8_t ucNbConnIntervals

uint8_t pduSize

uint8_t *pPdu

struct handoverAnchorMonitorPacketContinueEvent_tag

Public Members

uint8_t packetCounter

uint16_t connectionHandle

uint8_t pduSize

uint8_t *pPdu

struct handoverConnectionUpdateProcedureEvent_tag

Public Members

uint16_t connectionHandle

uint8_t winSize

uint16_t winOffset

uint16_t interval

uint16_t latency

uint16_t timeout

uint16_t instant

uint16_t currentEventCounter

struct handoverTimeSyncEvent_tag

Public Members

uint32_t txClkSlot

uint16_t txUs

uint32_t rxClkSlot

uint16_t rxUs

uint8_t rssi

struct handoverConnParamUpdateEvent_tag

Public Members

uint8_t status

uint16_t connectionHandle

uint32_t ulTxAccCode

uint8_t aCrcInitVal[3U]

uint16_t uiConnInterval

uint16_t uiSuperTO

uint16_t uiConnLatency

uint8_t aChMapBm[5U]

uint8_t ucChannelSelection

uint8_t ucHop

uint8_t ucUnMapChIdx

uint8_t ucCentralSCA

uint8_t ucRole

uint8_t aucRemoteMasRxPHY

uint8_t seqNum

uint16_t uiConnEvent

uint32_t ulAnchorClk

uint16_t uiAnchorDelay

uint32_t ulRxInstant

struct handoverSuspendTransmitCompleteEvent_tag

Public Members

uint16_t connectionHandle

uint8_t noOfPendingAclPackets

uint16_t sizeOfPendingAclPackets

uint16_t sizeOfDataTxInOldestPacket

uint8_t sizeOfDataNAckInOldestPacket

struct handoverResumeTransmitCompleteEvent_tag

Public Members

uint16_t connectionHandle

struct handoverUpdateConnParams_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverApplyConnectionUpdateProcedure_tag

Public Members

uint16_t connectionHandle

struct handoverAnchorNotificationStateChanged_tag

Public Members

uint16_t connectionHandle

struct handoverLlPendingDataIndication_tag

Public Members

uint16_t dataSize

uint8_t *pData

struct gapLEGenerateDhKeyCompleteEvent_t

Public Members

uint8_t aDHKey[(32U)]

struct gapRemoteVersionInfoRead_tag

Public Members

bleResult_t status

deviceId_t deviceId

uint8_t version

uint16_t manufacturerName

uint16_t subversion

struct gapLlSkdReport_tag

Public Members

deviceId_t deviceId

uint8_t aSKD[(16U)]

struct vendorUnitaryTestEvent_tag

Public Members

bleResult_t status

uint8_t paramLength

uint8_t aParam[(255U)]

struct gapGenericEvent_t

Public Members

gapGenericEventType_t eventType

union *gapGenericEvent_t* eventData

struct bleBondIdentityHeaderBlob_t

Public Members

uint32_t raw[(gBleBondIdentityHeaderSize_c + 3U) / sizeof(uint32_t)]

struct bleBondDataDynamicBlob_t

Public Members

uint32_t raw[((8U) + 3U) / sizeof(uint32_t)]

struct bleBondDataStaticBlob_t

Public Members

uint32_t raw[((30U) + 3U) / sizeof(uint32_t)]

struct bleBondDataLegacyBlob_t

Public Members

uint32_t raw[((28U) + 3U) / sizeof(uint32_t)]

struct bleBondDataDeviceInfoBlob_t

Public Members

uint32_t raw[((60U) + 3U) / sizeof(uint32_t)]

struct bleBondDataDescriptorBlob_t

Public Members

uint32_t raw[((4U) + 3U) / sizeof(uint32_t)]

struct bleLocalKeysBlob_t

Public Members

union bleLocalKeysBlob_t

bool_t keyGenerated

struct bleBondDataBlob_t

Public Members

bleBondIdentityHeaderBlob_t bondHeader

bleBondDataDynamicBlob_t bondDataBlobDynamic

bleBondDataStaticBlob_t bondDataBlobStatic

bleBondDataLegacyBlob_t bondDataBlobLegacy

bleBondDataDescriptorBlob_t bondDataDescriptors[(5U)]

bleBondDataDeviceInfoBlob_t bondDataBlobDeviceInfo

struct bleBondDataRam_t

Public Members

bleBondIdentityHeaderBlob_t bondHeader

bleBondDataDynamicBlob_t bondDataBlobDynamic

bleBondDataStaticBlob_t bondDataBlobStatic

bleBondDataLegacyBlob_t bondDataBlobLegacy

bleBondDataDeviceInfoBlob_t bondDataBlobDeviceInfo

bleBondDataDescriptorBlob_t bondDataDescriptors[1]

struct bleCteAllowedTypesMap_t

Public Members

uint8_t allowAoA

uint8_t allowAoD1us

uint8_t allowAoD2us

uint8_t reserved

struct bleSyncCteType_tag

Public Members

uint8_t doNotSyncWithAoA

uint8_t doNotSyncWithAoD1us

uint8_t doNotSyncWithAoD2us

uint8_t doNotSyncWithType3

uint8_t doNotSyncWithoutCte

uint8_t reserved

struct bleTxPowerLevelFlags_t

Public Members

uint8_t minimum

uint8_t maximum

uint8_t reserved

union eventData

Public Members

gapInternalError_t internalError

uint8_t filterAcceptListSize

bleDeviceAddress_t aAddress

gapAddrReadyEvent_t addrReady

uint8_t advHandle

bleResult_t setupFailError

int8_t advTxPowerLevel_dBm

bool_t verified

gapLeScOobData_t localOobData

bool_t newHostPrivacyState

bool_t newControllerPrivacyState

gapControllerTestEvent_t testEvent

bleResult_t txPowerLevelSetStatus

gapPhyEvent_t phyEvent

deviceId_t deviceId

gapInitComplete_t initCompleteData

bleNotificationEvent_t notifEvent

bleBondCreatedEvent_t bondCreatedEvent

bleDeviceAddress_t aControllerLocalRPA

getConnParams_t getConnParams

uint16_t syncHandle

bleAntennaInformation_t antennaInformation

bleResult_t perAdvSyncTransferEnable

periodicAdvSyncTransferEvent_t perAdvSyncTransfer

periodicAdvSetInfoTransferEvent_t perAdvSetInfoTransfer

periodicAdvSetSyncTransferParamsEvent_t perAdvSetSyncTransferParams

bleResult_t perAdvSetDefaultPerAdvSyncTransferParams

gapSyncTransferReceivedEventData_t perAdvSyncTransferReceived

gapLEGenerateDhKeyCompleteEvent_t leGenerateDhKeyCompleteEvent

uint16_t pawrSyncHandle

uint8_t pawrAdvHandle

handoverGetData_t handoverGetData

handoverSetData_t handoverSetData

handoverGetCsLlContext_t handoverGetCsLlContext

handoverGetTime_t handoverGetTime

handoverConnect_t handoverConnect

handoverAnchorSearchStart_t handoverAnchorSearchStart

handoverAnchorSearchStop_t handoverAnchorSearchStop

handoverAnchorMonitorEvent_t handoverAnchorMonitor

handoverTimeSyncEvent_t handoverTimeSync

handoverSuspendTransmitCompleteEvent_t handoverSuspendTransmitComplete

handoverResumeTransmitCompleteEvent_t handoverResumeTransmitComplete

handoverAnchorNotificationStateChanged_t handoverAnchorNotificationStateChanged

handoverConnParamUpdateEvent_t handoverConnParamUpdate

gapRemoteVersionInfoRead_t gapRemoteVersionInfoRead

gapLlSkdReport_t gapSkdReport

handoverAnchorMonitorPacketEvent_t handoverAnchorMonitorPacket

handoverAnchorMonitorPacketContinueEvent_t handoverAnchorMonitorPacketContinue

handoverUpdateConnParams_t handoverUpdateConnParams

handoverLlPendingDataIndication_t handoverLlPendingDataIndication

handoverConnectionUpdateProcedureEvent_t handoverConnectionUpdateProcedure

handoverApplyConnectionUpdateProcedure_t handoverApplyConnectionUpdateProcedure

vendorUnitaryTestEvent_t unitaryTestData

uint8_t monAdvListSize

union ___unnamed2___

Public Members

uint32_t raw[((40U) + 3U) / sizeof(uint32_t)]

uint8_t pKey[(40U)]

ble_host_tasks.h

ble_sig_defines.h

ble_utils.h

gap_types.h

struct gapSmpKeys_t

Public Members

uint8_t cLtkSize

uint8_t *aLtk

uint8_t *aIrk

uint8_t *aCsrk

uint8_t cRandSize

uint8_t *aRand

uint16_t ediv

bleAddressType_t addressType

uint8_t *aAddress

struct gapSecurityRequirements_t

Public Members

gapSecurityModeAndLevel_t securityModeLevel

bool_t authorization

uint16_t minimumEncryptionKeySize

struct gapServiceSecurityRequirements_t

Public Members

uint16_t serviceHandle

gapSecurityRequirements_t requirements

struct gapDeviceSecurityRequirements_t

Public Members

gapSecurityRequirements_t *pSecurityRequirements

uint8_t cNumServices

gapServiceSecurityRequirements_t *aServiceSecurityRequirements

struct gapHandleList_t

Public Members

uint8_t cNumHandles

uint16_t aHandles[(1U)]

struct gapConnectionSecurityInformation_t

Public Members

bool_t authenticated

gapHandleList_t authorizedToRead

gapHandleList_t authorizedToWrite

struct gapPairingParameters_t

Public Members

bool_t withBonding

gapSecurityModeAndLevel_t securityModeAndLevel

uint8_t maxEncryptionKeySize

gapIoCapabilities_t localIoCapabilities

bool_t oobAvailable

gapSmpKeyFlags_t centralKeys

gapSmpKeyFlags_t peripheralKeys

bool_t leSecureConnectionSupported

bool_t useKeypressNotifications

struct gapPeripheralSecurityRequestParameters_t

Public Members

bool_t bondAfterPairing

bool_t authenticationRequired

struct gapAdvertisingParameters_t

Public Members

uint16_t minInterval

uint16_t maxInterval

bleAdvertisingType_t advertisingType

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapAdvertisingChannelMapFlags_t channelMap

gapAdvertisingFilterPolicy_t filterPolicy

struct gapExtAdvertisingParameters_tag

Public Members

uint8_t SID

uint8_t handle

uint32_t minInterval

uint32_t maxInterval

bleAddressType_t ownAddressType

bleDeviceAddress_t ownRandomAddr

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapAdvertisingChannelMapFlags_t channelMap

gapAdvertisingFilterPolicy_t filterPolicy

bleAdvRequestProperties_t extAdvProperties

int8_t txPower

gapLePhyMode_t primaryPHY

gapLePhyMode_t secondaryPHY

uint8_t secondaryAdvMaxSkip

bool_t enableScanReqNotification

gapLePhyOptionsFlags_t primaryAdvPhyOptions

gapLePhyOptionsFlags_t secondaryAdvPhyOptions

struct gapPeriodicAdvParameters_tag

Public Members

uint8_t handle

bool_t addTxPowerInAdv

uint16_t minInterval

uint16_t maxInterval

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

uint8_t numResponseSlots

struct gapPeriodicAdvSyncTransfer_tag

Public Members

deviceId_t deviceId

uint16_t serviceData

uint16_t syncHandle

struct gapPeriodicAdvSetInfoTransfer_tag

Public Members

deviceId_t deviceId

uint16_t serviceData

uint16_t advHandle

struct gapSetPeriodicAdvSyncTransferParams_tag

Public Members

deviceId_t deviceId

gapPeriodicAdvSyncMode_t mode

uint16_t skip

uint16_t syncTimeout

bleSyncCteType_t CTEType

struct gapScanningParameters_t

Public Members

bleScanType_t type

uint16_t interval

uint16_t window

bleAddressType_t ownAddressType

bleScanningFilterPolicy_t filterPolicy

gapLePhyFlags_t scanningPHYs

struct gapCreateSyncReqOptions_tag

Public Members

gapCreateSyncReqFilterPolicy_t filterPolicy

uint8_t reportingEnabled

uint8_t duplicateFilteringEnabled

struct gapPeriodicAdvSyncReq_tag

Public Members

gapCreateSyncReqOptions_t options

uint8_t SID

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t skipCount

uint16_t timeout

bleSyncCteType_t cteType

struct gapConnectionRequestParameters_t

Public Members

uint16_t scanInterval

uint16_t scanWindow

bleInitiatorFilterPolicy_t filterPolicy

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t connIntervalMin

uint16_t connIntervalMax

uint16_t connLatency

uint16_t supervisionTimeout

uint16_t connEventLengthMin

uint16_t connEventLengthMax

bool_t usePeerIdentityAddress

gapLePhyFlags_t initiatingPHYs

struct gapConnectionFromPawrParameters_t

Public Members

uint16_t scanInterval

uint16_t scanWindow

bleInitiatorFilterPolicy_t filterPolicy

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t connIntervalMin

uint16_t connIntervalMax

uint16_t connLatency

uint16_t supervisionTimeout

uint16_t connEventLengthMin

uint16_t connEventLengthMax

bool_t usePeerIdentityAddress

gapLePhyFlags_t initiatingPHYs

uint8_t advHandle

uint8_t subevent

struct gapConnectionParameters_t

Public Members

uint16_t connInterval

uint16_t connLatency

uint16_t supervisionTimeout

bleCentralClockAccuracy_t centralClockAccuracy

struct gapGenerateDHKeyV2Params_t

Public Members

ecdhPublicKey_t remoteP256PublicKey

gapPrivateKeyType_t keyType

struct gapConnectionlessCteTransmitParams_t

Public Members

uint8_t handle

uint8_t cteLength

bleCteType_t cteType

uint8_t cteCount

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionlessIqSamplingParams_t

Public Members

bleIqSamplingEnable_t iqSamplingEnable

bleSlotDurations_t slotDurations

uint8_t maxSampledCtes

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteTransmitParams_t

Public Members

bleCteAllowedTypesMap_t cteTypes

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteReceiveParams_t

Public Members

bleIqSamplingEnable_t iqSamplingEnable

bleSlotDurations_t slotDurations

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteReqEnableParams_t

Public Members

bleCteReqEnable_t cteReqEnable

uint16_t cteReqInterval

uint8_t requestedCteLength

bleCteType_t requestedCteType

struct gapPathLossReportingParams_t

Public Members

uint8_t highThreshold

uint8_t highHysteresis

uint8_t lowThreshold

uint8_t lowHysteresis

uint16_t minTimeSpent

struct gapAdStructure_t

Public Members

uint8_t length

gapAdType_t adType

uint8_t *aData

struct gapAdvertisingData_t

Public Members

uint8_t cNumAdStructures

gapAdStructure_t *aAdStructures

struct gapAdvertisingDecisionData_t

Public Members

uint8_t *pKey

uint8_t *pPrand

uint8_t *pDecisionData

uint8_t dataLength

bool_t resolvableTagPresent

struct gapDecisionInstructionsData_tag

Public Members

gapDecisionInstructionsTestGroup_t testGroup

gapDecisionInstructionsTestPassCriteria_t passCriteria

gapDecisionInstructionsRelevantField_t relevantField

union *gapDecisionInstructionsData_tag* testParameters

struct gapSubeventDataStructure_t

Public Members

uint8_t subevent

uint8_t responseSlotStart

uint8_t responseSlotCount

gapAdvertisingData_t *pAdvertisingData

struct gapPeriodicAdvertisingSubeventData_t

Public Members

uint8_t cNumSubevents

gapSubeventDataStructure_t *aSubeventDataStructures

struct gapPeriodicAdvertisingResponseData_t

Public Members

uint16_t requestEvent

uint8_t requestSubevent

uint8_t responseSubevent

uint8_t responseSlot

gapAdvertisingData_t *pResponseData

struct gapPeriodicSyncSubeventParameters_t

Public Members

uint16_t perAdvProperties

uint8_t numSubevents

uint8_t aSubevents[1]

struct gapExtScanNotification_t

Public Members

uint8_t handle

bleAddressType_t scannerAddrType

bleDeviceAddress_t aScannerAddr

bool_t scannerAddrResolved

struct gapAdvertisingSetTerminated_t

Public Members

bleResult_t status

uint8_t handle

deviceId_t deviceId

uint8_t numCompletedExtAdvEvents

struct gapPerAdvSubeventDataRequest_t

Public Members

uint8_t handle

uint8_t subeventStart

uint8_t subeventDataCount

struct gapPerAdvResponse_t

Public Members

uint8_t advHandle

uint8_t subevent

int8_t txPower

int8_t rssi

uint8_t cteType

uint8_t responseSlot

uint8_t dataLength

uint8_t aData[255]

struct gapAdvertisingEvent_t

Public Members

gapAdvertisingEventType_t eventType

union *gapAdvertisingEvent_t* eventData

struct gapScannedDevice_t

Public Members

bleAddressType_t addressType

bleDeviceAddress_t aAddress

int8_t rssi

uint8_t dataLength

uint8_t *data

bleAdvertisingReportEventType_t advEventType

bool_t directRpaUsed

bleDeviceAddress_t directRpa

bool_t advertisingAddressResolved

struct gapExtScannedDevice_t

Public Members

bleAddressType_t addressType

bleDeviceAddress_t aAddress

uint8_t SID

bool_t advertisingAddressResolved

bleAdvReportEventProperties_t advEventProperties

int8_t rssi

int8_t txPower

uint8_t primaryPHY

uint8_t secondaryPHY

uint16_t periodicAdvInterval

bool_t directRpaUsed

bleAddressType_t directRpaType

bleDeviceAddress_t directRpa

uint16_t dataLength

uint8_t *pData

struct gapPeriodicScannedDevice_t

Public Members

uint16_t syncHandle

int8_t txPower

int8_t rssi

bleCteType_t cteType

uint16_t dataLength

uint8_t *pData

struct gapPeriodicScannedDeviceV2_t

Public Members

uint16_t syncHandle

int8_t txPower

int8_t rssi

bleCteType_t cteType

uint16_t periodicEventCounter

uint8_t subevent

uint16_t dataLength

uint8_t *pData

struct gapSyncEstbEventData_t

Public Members

bleResult_t status

uint16_t syncHandle

uint8_t SID

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapLePhyMode_t PHY

uint16_t periodicAdvInterval

bleAdvertiserClockAccuracy_t advertiserClockAccuracy

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

struct gapSyncLostEventData_t

Public Members

uint16_t syncHandle

struct gapConnectionlessIqReport_t

Public Members

uint16_t syncHandle

uint8_t channelIndex

int16_t rssi

uint8_t rssiAntennaId

bleCteType_t cteType

bleSlotDurations_t slotDurations

bleIqReportPacketStatus_t packetStatus

uint16_t periodicEventCounter

uint8_t sampleCount

int8_t *aI_samples

int8_t *aQ_samples

struct gapMonAdvReportReport_t

Public Members

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bleMonAdvCondition_t condition

struct gapScanningEvent_t

Public Members

gapScanningEventType_t eventType

union *gapScanningEvent_t* eventData

struct gapConnectedEvent_t

Public Members

gapConnectionParameters_t connParameters

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bool_t peerRpaResolved

bleDeviceAddress_t peerRpa

bool_t localRpaUsed

bleDeviceAddress_t localRpa

bleLlConnectionRole_t connectionRole

uint8_t advHandle

uint16_t syncHandle

struct gapKeyExchangeRequestEvent_t

Public Members

gapSmpKeyFlags_t requestedKeys

uint8_t requestedLtkSize

struct gapKeysReceivedEvent_t

Public Members

gapSmpKeys_t *pKeys

struct gapAuthenticationRejectedEvent_t

Public Members

gapAuthenticationRejectReason_t rejectReason

struct gapPairingCompleteEvent_t

Public Members

bool_t pairingSuccessful

union *gapPairingCompleteEvent_t* pairingCompleteData

struct gapLongTermKeyRequestEvent_t

Public Members

uint16_t ediv

uint8_t aRand[(8U)]

uint8_t randSize

struct gapEncryptionChangedEvent_t

Public Members

bool_t newEncryptionState

struct gapDisconnectedEvent_t

Public Members

gapDisconnectionReason_t reason

struct gapConnParamsUpdateReq_t

Public Members

uint16_t intervalMin

uint16_t intervalMax

uint16_t peripheralLatency

uint16_t timeoutMultiplier

struct gapConnParamsUpdateComplete_t

Public Members

bleResult_t status

uint16_t connInterval

uint16_t connLatency

uint16_t supervisionTimeout

struct gapConnLeDataLengthChanged_t

Public Members

uint16_t maxTxOctets

uint16_t maxTxTime

uint16_t maxRxOctets

uint16_t maxRxTime

struct gapConnIqReport_t

Public Members

gapLePhyMode_t rxPhy

uint8_t dataChannelIndex

int16_t rssi

uint8_t rssiAntennaId

bleCteType_t cteType

bleSlotDurations_t slotDurations

bleIqReportPacketStatus_t packetStatus

uint16_t connEventCounter

uint8_t sampleCount

int8_t *aI_samples

int8_t *aQ_samples

struct gapConnCteRequestFailed_t

Public Members

bleResult_t status

struct gapPathLossThresholdEvent_t

Public Members

uint8_t currentPathLoss

blePathLossThresholdZoneEntered_t zoneEntered

struct gapTransmitPowerReporting_t

Public Members

bleTxPowerReportingReason_t reason

blePowerControlPhyType_t phy

int8_t txPowerLevel

bleTxPowerLevelFlags_t flags

int8_t delta

struct gapTransmitPowerInfo_t

Public Members

blePowerControlPhyType_t phy

int8_t currTxPowerLevel

int8_t maxTxPowerLevel

struct gapEattConnectionRequest_t

Public Members

uint16_t mtu

uint8_t cBearers

uint16_t initialCredits

struct gapEattConnectionComplete_t

Public Members

l2caLeCbConnectionRequestResult_t status

uint16_t mtu

uint8_t cBearers

bearerId_t aBearerIds[(5U)]

struct gapEattReconfigureResponse_t

Public Members

l2capReconfigureResponse_t status

uint16_t localMtu

uint8_t cBearers

bearerId_t aBearerIds[(5U)]

struct gapEattBearerStatusNotification_t

Public Members

bearerId_t bearerId

gapEattBearerStatus_t status

struct gapHandoverConnectedEvent_t

Public Members

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bleLlConnectionRole_t connectionRole

struct gapHandoverDisconnectedEvent_t

Public Members

bleResult_t status

struct gapConnectionEvent_t

Public Members

gapConnectionEventType_t eventType

union *gapConnectionEvent_t* eventData

struct gapIdentityInformation_t

Public Members

bleIdentityAddress_t identityAddress

uint8_t irk[(16U)]

blePrivacyMode_t privacyMode

struct gapAutoConnectParams_t

Public Members

uint8_t cNumAddresses

bool_t writeInFilterAcceptList

gapConnectionRequestParameters_t *aAutoConnectData

struct gapHostVersion_tag

Public Members

uint8_t bleHostVerMajor

uint8_t bleHostVerMinor

uint8_t bleHostVerPatch

union testParameters

Public Members

uint8_t resolvableTagKey[(16U)]

struct *gapDecisionInstructionsData_tag* arbitraryData

struct *gapDecisionInstructionsData_tag* rssi

struct *gapDecisionInstructionsData_tag* pathLoss

struct *gapDecisionInstructionsData_tag* advA

gapDecisionInstructionsAdvMode_t advMode

struct arbitraryData

Public Members

uint8_t mask[(8U)]

uint8_t target[(8U)]

struct rssi

Public Members

int8_t min

int8_t max

struct pathLoss

Public Members

uint8_t min

uint8_t max

struct advA

Public Members

gapDecisionInstructionsAdvAChecks_t check

bleAddressType_t address1Type

bleDeviceAddress_t address1

bleAddressType_t address2Type

bleDeviceAddress_t address2

union eventData

Public Members

bleResult_t failReason

gapExtScanNotification_t scanNotification

gapAdvertisingSetTerminated_t advSetTerminated

gapPerAdvSubeventDataRequest_t subeventDataRequest

gapPerAdvResponse_t perAdvResponse

uint8_t advHandle

union eventData

Public Members

bleResult_t failReason

gapScannedDevice_t scannedDevice

gapExtScannedDevice_t extScannedDevice

gapPeriodicScannedDevice_t periodicScannedDevice

gapPeriodicScannedDeviceV2_t periodicScannedDeviceV2

gapSyncEstbEventData_t syncEstb

gapSyncLostEventData_t syncLost

gapConnectionlessIqReport_t iqReport

gapMonAdvReportReport_t monAdvReport

union pairingCompleteData

Public Members

bool_t withBonding

bleResult_t failReason

union eventData

Public Members

gapConnectedEvent_t connectedEvent

gapPairingParameters_t pairingEvent

gapAuthenticationRejectedEvent_t authenticationRejectedEvent

gapPeripheralSecurityRequestParameters_t peripheralSecurityRequestEvent

gapKeyExchangeRequestEvent_t keyExchangeRequestEvent

gapKeysReceivedEvent_t keysReceivedEvent

gapPairingCompleteEvent_t pairingCompleteEvent

gapLongTermKeyRequestEvent_t longTermKeyRequestEvent

gapEncryptionChangedEvent_t encryptionChangedEvent

gapDisconnectedEvent_t disconnectedEvent

int8_t rssi_dBm

int8_t txPowerLevel_dBm

bleResult_t failReason

uint32_t passkeyForDisplay

gapConnParamsUpdateReq_t connectionUpdateRequest

gapConnParamsUpdateComplete_t connectionUpdateComplete

gapConnLeDataLengthChanged_t leDataLengthChanged

gapKeypressNotification_t incomingKeypressNotification

uint32_t numericValueForDisplay

bleChannelMap_t channelMap

gapConnIqReport_t connIqReport

gapConnCteRequestFailed_t cteRequestFailedEvent

bleResult_t perAdvSyncTransferStatus

gapPathLossThresholdEvent_t pathLossThreshold

gapTransmitPowerReporting_t transmitPowerReporting

gapTransmitPowerInfo_t transmitPowerInfo

gapEattConnectionRequest_t eattConnectionRequest

gapEattConnectionComplete_t eattConnectionComplete

gapEattReconfigureResponse_t eattReconfigureResponse

gapEattBearerStatusNotification_t eattBearerStatusNotification

gapHandoverConnectedEvent_t handoverConnectedEvent

gapHandoverDisconnectedEvent_t handoverDisconnectedEvent

bleResult_t smError

att_errors.h

gatt_types.h

struct gattAttribute_t

Public Members

uint16_t handle

bleUuidType_t uuidType

bleUuid_t uuid

uint16_t valueLength

uint16_t maxValueLength

uint8_t *paValue

struct gattCharacteristic_t

Public Members

gattCharacteristicPropertiesBitFields_t properties

gattAttribute_t value

uint8_t cNumDescriptors

gattAttribute_t *aDescriptors

struct gattService_tag

Public Members

uint16_t startHandle

uint16_t endHandle

bleUuidType_t uuidType

bleUuid_t uuid

uint8_t cNumCharacteristics

gattCharacteristic_t *aCharacteristics

uint8_t cNumIncludedServices

struct *gattService_tag* *aIncludedServices

struct gattDbCharPresFormat_t

Public Members

uint8_t format

int8_t exponent

uint16_t unitUuid16

uint8_t ns

uint16_t description

struct gattHandleRange_t

Public Members

uint16_t startHandle

uint16_t endHandle

struct procStatus_t

Public Members

bool_t isOngoing

gattProcedureType_t ongoingProcedureType

gattProcedurePhase_t ongoingProcedurePhase

struct procDataStruct_t

Public Members

uint16_t index

uint16_t max

bleUuid_t charUuid

bleUuidType_t charUuidType

bool_t reliableLongWrite

bool_t bAllocatedArray

union *procDataStruct_t* pOutActualCount

union *procDataStruct_t* array

union *procDataStruct_t* reqParams

union pOutActualCount

Public Members

uint8_t *pCount8b

uint16_t *pCount16b

union array

Public Members

gattService_t *aServices

gattCharacteristic_t *aChars

gattAttribute_t *aDescriptors

uint8_t *aBytes

uint16_t *aHandles

union reqParams

Public Members

attReadByGroupTypeRequestParams_t rbgtParams

attFindByTypeValueRequestParams_t fbtvParams

attReadByTypeRequestParams_t rbtParams

attFindInformationRequestParams_t fiParams

attReadRequestParams_t rParams

attReadBlobRequestParams_t rbParams

attReadMultipleRequestParams_t rmParams

attVarWriteRequestAndCommandParams_t wParams

attSignedWriteCommandParams_t swParams

attPrepareWriteRequestResponseParams_t pwParams

attExecuteWriteRequestParams_t ewParams

gatt_interface.h

gatt_client_interface.h

gatt_server_interface.h

struct gattServerMtuChangedEvent_t

Public Members

uint16_t newMtu

struct gattServerAttributeWrittenEvent_t

Public Members

uint16_t handle

uint16_t cValueLength

uint8_t *aValue

bearerId_t bearerId

struct gattServerLongCharacteristicWrittenEvent_t

Public Members

uint16_t handle

uint16_t cValueLength

uint8_t *aValue

struct gattServerCccdWrittenEvent_t

Public Members

uint16_t handle

gattCccdFlags_t newCccd

struct gattServerAttributeReadEvent_t

Public Members

uint16_t handle

struct gattServerProcedureError_t

Public Members

gattServerProcedureType_t procedureType

bleResult_t error

struct gattServerInvalidPdu_t

Public Members

attOpcode_t attOpCode

struct gattServerEvent_t

Public Members

gattServerEventType_t eventType

union *gattServerEvent_t* eventData

union eventData

Public Members

gattServerMtuChangedEvent_t mtuChangedEvent

gattServerAttributeWrittenEvent_t attributeWrittenEvent

gattServerCccdWrittenEvent_t charCccdWrittenEvent

gattServerProcedureError_t procedureError

gattServerLongCharacteristicWrittenEvent_t longCharWrittenEvent

gattServerAttributeReadEvent_t attributeReadEvent

gattServerInvalidPdu_t attributeOpCode

gatt_database.h

struct gattDbAttribute_t

Public Members

uint16_t handle

uint16_t permissions

uint32_t uuid

uint8_t *pValue

uint16_t valueLength

uint16_t uuidType

uint16_t maxVariableValueLength

gatt_db_app_interface.h

l2ca_cb_interface.h

struct l2caLeCbConnectionRequest_tag

Public Members

deviceId_t deviceId

uint16_t lePsm

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

struct l2caLeCbConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

l2caLeCbConnectionRequestResult_t result

struct l2caHandoverConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t peerMtu

uint16_t peerMps

uint16_t credits

struct l2caLeCbDisconnection_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbNoPeerCredits_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbLowPeerCredits_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbLocalCreditsNotification_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t localCredits

struct l2caLeCbError_tag

Public Members

deviceId_t deviceId

bleResult_t result

l2caErrorSource_t errorSource

struct l2caLeCbChannelStatusNotification_tag

Public Members

deviceId_t deviceId

uint16_t cId

l2caChannelStatus_t status

struct l2caEnhancedConnectionRequest_tag

Public Members

deviceId_t deviceId

uint16_t lePsm

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

l2caLeCbConnectionRequestResult_t result

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedReconfigureRequest_tag

Public Members

deviceId_t deviceId

uint16_t newMtu

uint16_t newMps

l2capReconfigureResponse_t result

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedReconfigureResponse_tag

Public Members

deviceId_t deviceId

l2capReconfigureResponse_t result

struct l2capControlMessage_tag

Public Members

l2capControlMessageType_t messageType

union *l2capControlMessage_tag* messageData

union messageData

Public Members

l2caLeCbConnectionRequest_t connectionRequest

l2caLeCbConnectionComplete_t connectionComplete

l2caLeCbDisconnection_t disconnection

l2caLeCbNoPeerCredits_t noPeerCredits

l2caLeCbLowPeerCredits_t lowPeerCredits

l2caLeCbLocalCreditsNotification_t localCreditsNotification

l2caLeCbError_t error

l2caLeCbChannelStatusNotification_t channelStatusNotification

l2caEnhancedConnectionRequest_t enhancedConnRequest

l2caEnhancedConnectionComplete_t enhancedConnComplete

l2caEnhancedReconfigureRequest_t reconfigureRequest

l2caEnhancedReconfigureResponse_t reconfigureResponse

l2caHandoverConnectionComplete_t handoverConnectionComplete

l2ca_types.h

Wireless Framework

Wireless Connectivity Framework Connectivity Framework repository provides both connectivity platform enablement with hardware abstraction layer and a set of Services for NXP connectivity stacks : BLE, Zigbee, OpenThread, Matter.

The connectivity framework repository consists of:

- Common folder to common header files for minimal type definition to be used in the repo
- Platform folder used for platform enablement with Hardware abstraction:
  - platform/include: common API header files used by several platforms
  - platform/common: common code for several platforms
  - specifics platform folders , See below the supported platform list
  - platform/../configs folder: configuration files for framework repository and other middlewares (rpmsg, mbedTls, etc.._)
- Services folder
- Zephyr folder for zephyr modules integrated in mcux SDK
- clang formatting script and script folder to format appropriately the source files of the repo

Supported platforms The following devices/platforms are supported in platform folder for connectivity applications:

- kw45x, k32w1x, mcxw71x, under wireless_mcu, kw45_k32w1_mcxw71 folders.
- kw47x, mcxw72x families under wireless_mcu, kw47_mcxw72, kw47_mcxw72_nbu folders.
- rw61x
- RT1060 and RT1170 for Matter
- Other RT devices such as i.MX RT595s

Supported services The supported services are provided for connectivity stacks and their demo application, and are usually dependent on PLATFORM API implementation:

- DBG: Light Debug Module, currently a stubbed header file
- FSCI: Framework Serial Communication Interface between BLE host stack and upper layer located on an other core/device
- FunctionLib: wrapper to toolchain memory manipulation functions (memcpy, memcmp, etc) or use its own implementation for code size reduction
- HWParameters: Store Factory hardware parameters and Application parameters in Flash or IFR
- LowPower: wrapper of SDK power manager for connectivity applications
- ModuleInfo: Store and handle connectivity component versions
- NVM: NXP proprietary File System used for KW45, KW47 automotive devices and RT1060/RT1170 platform for Matter
- OtaSupport: Handle OTA binary writes into internal or external flash.
- SecLib and RNG: Crypto and Random Number generator functions. It supports several ports:
  - Software algorithms
  - Secure subsystem interface to an HW enclave
  - MbedTls 2.x interface
- Sensors: Provides service for Battery and temperature measurements
- SFC: Smart Frequency Calibration to be run from KW47/MCXW71 from NBU core. Matter related modules:

- OTW: Over The Wire module for External Transceiver firmware update from RT platforms
- FactoryDataProvider to be used for Matter

Supported Zephyr modules integration in mcux SDK Connectivity framework provides integration and port layers to the following Zephyr Modules located into zephyr/subsys:

- NVS: Zephyr File System used by Matter and Zigbee
- Settings: Over layer module that allows to store keys into NVS File System used by Matter Port layer and required libraries for these zephyr modules are located in port and lib folder in zephyr directory

Connectivity framework CHANGELOG

7.1.2 mcux SDK 25.12.00 pvw2

Major Changes

- [wireless_mcu] Reduced RPMSG buffer payload size from 496 to 270 bytes on KW43/KW47 platforms, saving 226 bytes per buffer (1808 bytes total with 4 buffers on each core). This optimization is possible as rpmsg-lite no longer requires buffer sizes to be powers of two.
- [configs] Introduced RL_ALLOW_CUSTOM_SHMEM_CONFIG flag in rpmsg_config.h to enable connectivity applications to use platform_set_static_shmem_config() and platform_get_custom_shmem_config().

Minor Changes

- [wireless_mcu] Updated radio power management configuration with PLATFORM_InitRadio() API on kw47/mcxw72 platforms
- [DBG] Added NBU assert indication support to host with line/file info using debug structure.
- [DBG] Enhanced NBU debug framework with warning detection and notification capabilities. Extended NBUDBG_StateCheck() to monitor NBU warnings via PLATFORM_IsNbuWarningSet() with callback support for proactive warning monitoring.
- [Sensors] Added periodic temperature measurement support allowing app/host to request periodic temperature measurement.
- [Sensors] Added markdown documentation explaining periodic measurement functionality upon NBU requests.
- [SecLib_RNG] Added documentation for asynchronous seed handling using RNG_NotifyReseedNeeded() and SecLib implementation flavors (Software, EdgeLock, PSA Crypto, MbedTLS).
- [PSA] Simplified PSA configuration files and reduced imports/definitions for wireless MCU platforms.
- [mcxw23] Changed temperature dummy value for sensors to progress on application enablement.
- [NVS] Enhanced debug capabilities by adding CONFIG_NVS_LOG_LEVEL and improved LOG macros to adapt to PRINTF limitations.

Bug fixes

- [wireless_mcu] Fixed FRO32K as 32 kHz clock source with deferred OSC32K switching to improve initialization performance after warm reset.
- [wireless_mcu] Added wait loop for NBU power domain readiness in PLATFORM_InitNbu() to prevent race conditions when accessing NBU power domain in applications without NBU images.
- [wireless_mcu] Fixed external flash blank check procedure for LSPI external NOR Flash by correcting PLATFORM_ExternalFlashAreaIsBlank() to read from RAM buffer and perform erase pattern comparison in RAM with optimized 4-byte step loops.
- [NVS] Removed mflash dependency from NVS external flash port and fixed internal flash blank check of unaligned flash data.
- [mcxw23] Ensured TX power does not exceed 6dBm maximum limit.

7.1.1 mcux SDK 25.12.00 pvw1

Minor Changes

- [wireless_mcu][wireless_nbu] Migrated TSTMR implementation to use SDK fsl_tstmr driver for better maintainability and consistency. This migration replaces custom TSTMR register definitions with official SDK driver APIs while maintaining existing PLATFORM_* API compatibility.

Bug fixes

- [SecLib_RNG][mbedtls] Enhanced ECDH context preservation across low-power transitions on KW45_MCXW71 and KW47_MCXW72 platforms using export/import APIs to ensure cryptographic context is retained when hardware accelerator loses internal memory during power-down mode.
- [wireless_nbu] Fixed incorrect FRODIV values that were causing reduced peripheral clocks by updating PLATFORM_FroDiv[] mapping array to prevent over-division of flash APB and RF_CMC clocks.
- [rw61x/rt1060/rt1170] Added missing common files for external and OTA support on rw61x/rt1060/rt1170 platforms.

7.0.3 RFP mcux SDK 25.09.00

Major Changes

- [wireless_mcu] Replaced ICS RX linked list with message queue to eliminate memory allocation in ISR context and enable user callbacks to run in thread context where memory allocation is permitted.
- [wireless_mcu] Added HCI RX workqueue processing support to reduce ISR execution time and system impact. Feature controlled by gPlatformHciUseWorkqueueRxProcessing_d configuration option (enabled by default on kw45_k32w1_mcxw71 and kw47_mcxw72 platforms in freertos applications). When enabled, HCI transport processes received data in system workqueue thread, allowing user callbacks to run in thread context.
- [wireless_nbu] Introduced PLATFORM_ConfigureSmuDmemMapping() API to configure SMU and DMEM sharing on NBU for kw47_mcxw72 platform, using linker file symbols for correct configuration.

- [mcxw23] Implemented HCI interface using PLATFORM API as preliminary requirement for Zephyr enablement, introducing PLATFORM_SendHciMessageAlt() alternative API.
- [wireless_mcu][wireless_nbu] Added NBU2Host event manager for status indications to host (Information, Warning, Error) sent over RPMSG.
- [wireless_mcu] Added a call to PLATFORM_IcsRxWorkHandler() within PLATFORM_NbuApiReq() for baremetal applications to prevent potential deadlocks.

Minor Changes

- [wireless_mcu] Fixed variable underflow issue in PLATFORM_RemoteActiveRel().
- [SecLib_RNG] Fixed escaping local HashKeyBuffer address issue and added missing cast in RNG_GetTrueRandomNumber() function.
- [Common] Fixed heap memory manager return values and added missing include to fwk_freertos_utils.h.
- [rw61x] Prevented array out of bounds in PLATFORM_RegisterRtcHandle().
- [FSCI] Fixed memory leak in FSCI module.
- [NVM] Enhanced debug facilitation by restricting variable scope, assigning return statuses to variables, and fixing display format in NV_ShowFlashTable().
- [wireless_mcu] Added new chip revision A2.1 support in PLATFORM_SendChipRevision() API.
- [kw47_mcxw72] Implemented BLE BD address retrieval from IFR memory with fallback to OUI + RNG.
- [DBG] Added ThreadX support to fault handlers and reworked fault handler structure with dedicated RTOS files.
- [DBG][Common] Added NBU debug support on host side to detect faults and system errors, with debug info extraction capability (limited to MCXW72/KW47).
- [Common] Platform CMake rework and Kconfig renaming, removing unneeded checks and renaming PRJSEG platform Kconfigs to COMPONENT.
- [mcxw23] Added experimental SecLib PSA support with additional configuration for MBEDTLS_ECP_C and MBEDTLS_BIGNUM_C.
- [wireless_mcu] Cleaned CMakeLists.txt to avoid wrong inclusions of files and folders from incorrect platforms.
- [wireless_mcu][wireless_nbu] Added NBU2Host warning when 32MHz crystal is unready on low power exit.
- [wireless_mcu][ot] Introduced gPlatformUseOuiFromIfr to use OUI from IFR for the extended address (disabled by default). When enabled and IFR is not blank, copies first three bytes to OUI field of extended address, otherwise uses static OUI as fallback.
- [General] Removed useless warning about TSTMR_CLOCK_FREQUENCY_MHZ definition.
- [General] Updated framework license and SBOM for 25.09 RFP release.
- [wireless_mcu] Fixed unused variable warning when gPlatformIcsUseWorkqueueRxProcessing_d and gPlatformHciUseWorkqueueRxProcessing_dare disable

7.0.3 revB mcux SDK 25.09.00

Major Changes

- [wireless_mcu] Adjusted default value of BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0x16 to 0x10 to address stability issues observed with the previous setting. This change enhances system reliability but will reduce low-power performance.

Minor Changes (bug fixes)

- [Common] Added MDK compatibility for the errno framework header.
- [mcxw23] Implemented missing PLATFORM_OtaClearBootInterface() API.
- [mcxw23] Refactored fwk_platform.c to separate BLE-specific logic into fwk_platform_ble.c.
- [OTA] Corrected definition of gEepromParams_WriteAlignment_c flag for mcxw23
- [OTA] Enabled calling OTA_GetImgState() prior to OTA_Initialize().
- [wireless_mcu] Fixed PLATFORM_IsExternalFlashSectorBlank() to check the entire sector instead of just one page.
- [mcxw23] Added support for OTA using external flash.
- [mcxw23] Introduced PLATFORM_GetRadioIdleDuration32K() to estimate time until next radio event.
- [OTA] Removed gUseInternalStorageLink_d linker flag definition when external OTA storage is used.
- [mcxw23] Extended CopyAndReboot() to support external flash OTA.
- [wireless_mcu] Resolved counter wrap issue in PLATFORM_GetDeltaTimeStamp().
- [kw43_mcxw70] Defined LPTMR frequency constants in fwk_platform_definitions.h.
- [kw47_mcxw72] Updated shared memory allocation for RPMsg adapter.
- [mcxw23] Implement PLATFORM_IsExternalFlashBusy() API.
- [kw45_mcxw71][kw47_mcxw72] Moved RAM bank definitions from the connectivity framework to device-specific definitions.

7.0.3 revA mcux SDK 25.09.00

Major Changes

- [wireless_nbu] Enhanced XTAL32M trimming handling: updates are applied when requested by the application core and the NBU enters low-power mode, ensuring no interference from ongoing radio activity. Introduced new APIs to lock (PLATFORM_LockXtal32MTrim()) and unlock XTAL32M (PLATFORM_UnlockXtal32MTrim()) trimming updates using a counter-based mechanism. Also added a reset API (PLATFORM_ResetContext()) for platform-specific variables (currently limited to the trimming lock).
- [wireless_mcu] Introduced a new API, PLATFORM_SetLdoCoreNormalDriveVoltage(), to enable support for NBU clock frequency at 64 MHz, as required by BLE channel sounding applications.
- [wireless_mcu][wireless_nbu] Increased delayLpoCycle default from 2 to 3 to address link layer instabilities in low-power NBU use cases. Adjusted BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0x10 to 0x16 to balance power consumption and stability. NBU may malfunction if delayLpoCycle (or BOARD_LL_32MHz_WAKEUP_ADVANCE_HSLOT) is set to 2 while BOARD_RADIO_DOMAIN_WAKE_UP_DELAY is 0x16.

Minor Changes (bug fixes)

- [WorkQ] Increased stack size when RNG use mbedtls port and coverage is enabled.
- [FSCI] Resolved an issue where messages remained unprocessed in the queue by ensuring OSA_EventSet() is triggered when pending messages are detected.
- [OTA] Fixed a bug in in OTA_PullImageChunk() that prevented retrieval of data previously received via OTA_PushImageChunk() when still buffered in RAM during posted operations.
- [OTA] Various MISRA and coverity fixes.
- [mcxw23] Fixed an unused variable warning in PLATFORM_RegisterNbuTemperatureRequestEventCb() API.
- [SFC] Remove obsolete flag gNbuJtagCapability.
- [wireless_mcu] Introduced new API PLATFORM_GetRadioIdleDuration32K(). Deprecated PLATFORM_CheckNextBleConnectivityActivity() API.
- [mcxw23] Aligned platform-specific implementations with the corresponding prototypes defined in wireless_mcu.
- [DBG] Cleaned up fwk_fault_handler.c.

7.0.2 RFP mcux SDK 25.06.00

Major Changes

- [wireless_mcu][wireless_nbu] Introduced PLATFORM_Get32KTimeStamp() API, available on platforms that support it.
- [RNG] Switched to using a workqueue for scheduling seed generation tasks.
- [Sensors] Integrated workqueue to trigger temperature readings on periodic timer expirations.
- [wireless_nbu] Removed outdated configuration files from wireless_nbu/configs.
- [SecLib_RNG][PSA] Added a PSA-compliant implementation for SecLib_RNG. This is an experimental feature and should be used with caution.
- [wireless_mcu][wireless_nbu] Implemented PLATFORM_SendNBUXtal32MTrim() API to transmit XTAL32M trimming values to the NBU.

Minor Changes (bug fixes)

- [MWS] Migrated the Mobile Wireless Standard (MWS) service to the public repository. This service manages coexistence between connectivity protocols such as BLE, 802.15.4, and GenFSK.
- [HWParameter][NVM][SecLib_RNG][Sensors] Addressed various MISRA compliance issues across multiple modules.
- [Sensors] Applied a filtering mechanism to temperature data measured by the application core before forwarding it to the NBU, improving data reliability.
- [Common] Relocated the GetPowerOfTwoShift() function to a shared module for broader accessibility across components.
- [RNG] Resolved inconsistencies in RNG behavior when using the fsl_adapter_rng HAL by aligning it with other API implementations.
- [SecLib] Updated the AES CMAC block counter in AES_128_CMAC() and AES_128_CMAC_LsbFirstInput() to support data segments larger than 4KB.

- [SecLib] Utilized sss_sscp_key_object_free() with kSSS_keyObjFree_KeysStoreDefragment to avoid key allocation failures.
- [MCXW23] Removed redundant NVIC_SetPriority() call for the ctimer IRQ in the platform file, as it's already handled by the driver.
- [WorkQ] Increased workqueue stack size to accommodate RNG usage with mbedtls.
- [wireless_mcu][ot] Suppressed chip revision transmission when operating with nbu_15_4.
- [platform][mflash] Ensured proper address alignment for external flash reads in PLATFORM_ReadExternalFlash() when required by platform constraints.
- [RNG] Corrected reseed flag behavior in RNG_GetPseudoRandomData() after reaching gRngMaxRequests_d threshold.
- [platform][mflash] Fixed uninitialized variable issue in PLATFORM_ReadExternalFlash().
- [platform][wireless_nbu] Fixed an issue on KW47 where PLATFORM_InitFro192M incorrectly reads IFR1 from a hardcoded flash address (0x48000), leading to unstable FRO192M trimming. The function is now conditionally compiled for KW45 only.

7.0.2 revB mcux SDK 25.06.00

Major Changes

- [RNG][wireless_mcu][wireless_nbu] Rework RNG seeding on NBU request
- [wireless_mcu] [LowPower] Add gPlatformEnableFro6MCalLowpower_d macro to enable FRO6M frequency verification on exit of Low Power
  - add PLATFORM_StartFro6MCalibration() and PLATFORM_EndFro6MCalibration() new function for FRO6M calibration (6MHz or 2Mhz) on wake-up from low power mode.
  - Enabled by default in fwk_config.h
- [wireless_nbu][LowPower] Clear pending interrupt status of the systick before going in low-power - Reduce NBU active time
- [wireless_nbu] Fix impossibility to go to WFI in combo mode (15.4/BLE)
- [wireless_mcu] Implement XTAL32M temperature compensation mechanism. 2 new APIs:
  - PLATFORM_RegisterXtal32MTempCompLut(): register the temperature compensation table for XTAL32M.
  - PLATFORM_CalibrateXtal32M(): apply XTAL32M temperature compensation depending on current temperature.
- [Sensors][wireless_mcu] Add support for periodic temperature measurement. new API:
  - SENSORS_TriggerTemperatureMeasurementUnsafe(): to be called from Interrupt masked critical section, from ISR or when scheduler is stopped
- [SFC] Change default maximal ppm target of the SFC algorithm from 200 to 360ppm. Impact the SFC algorith of kw45 and mcxw71 platforms, 360ppm was already the default setting for kw47 and mcxw72 platforms

Minor Changes (bug fixes)

- [DBG] Fix FWK_DBG_PERF_DWT_CYCLE_CNT_STOP macro
- [wireless_nbu] Add gPlatformIsNbu_d compile Macro set to 1
- [wireless_nbu][ics] gFwkSrvHostChipRevision_c can be processed in the system workqueue
- [kw45_mcxw71][kw47_mcxw72]

  - Remove LTC dependency from platform in kconfig
  - gPlatformShutdownEccRamInLowPower moved from fwk_platform_definition.h to fwk_confg.h as this is a configuration flag.
- [wireless_mcu][sensors] Rework and remove unnecessary ADC APIs
- [wireless_nbu] Add PLATFORM_GetMCUUid() function from Chip UID
- [SecLib] Change AES_MMO_BlockUpdate() function from private to public for zigbee.

7.0.2 revA mcux SDK 25.06.00 Supported platforms:

- Same as 25.03.00 release

Major Changes

- [KW45/MCXW71] HW parameters placement now located in IFR section. Flash storage is not longer used:
  - Compilation: Macro gHwParamsProdDataPlacement_c changed from gHwParamsProdDataMainFlash2IfrMode_c to gHwParamsProdDataIfrMode_c
- [KW47] NBU: Add new fwk_platform_dcdc.[ch] files to allow DCDC stepping by using SPC high power mode. This requires new API in board_dcdc.c files. Please refer to new compilation MACROs gBoardDcdcRampTrim_c and gBoardDcdcEnableHighPowerModeOnNbu_d in board_platform.h files located in kw47evk, kw47loc, frdmmcxw72 board folders.
- [KW45/MCXW71/KW47/MCXW72] Trigger an interrupt each time App core calls PLATFORM_RemoteActiveReq() to access NBU power domain in order to restart NBU core for domain low power process

Minor Changes (bug fixes)

Services

- [SecLib_RNG]
  - Rename mSecLibMutexId mutex to mSecLibSssMutexId in SecLib_sss.c
  - Remove MEM_TRACKING flag from RNG.c
  - Implement port to fsl_adapter_rng.h API using gRngUseRngAdapter_c compil Macro from RNG.c
  - Add support for BLE debug Keys in SecLi and SecLin_sss.c with gSecLibUseBleDebugKeys_d - for Debug only
- [FSCI] Add queue mechanism to prevent corruption of FSCI global variableAllow the application to override the trig sample number parameter when gFsciOverRpmsg_c is set to 1
- [DBG][btsnoop] Add a mechanism to dump raw HCI data via UART using SBTSNOOP_MODE_RAW
- [OTA]
  - OtaInternalFlash.c: Take into account chunks smaller than a flash phrase worth
  - fwk_platform_ot.c: dependencies and include files to gpio, port, pin_mux removed

Platform specific

- [kw45_mcxw71][kw47_mcxw72]
  - fwk_platform_reset.h : add compil Macro gUseResetByLvdForce_c and gUseResetByDeepPowerDown_c to avoid compile the code if not supported on some platforms
  - New compile Flag gPlatformHasNbu_d
  - Rework FRO32K notification service for MISRA fix

7.0.1 RFP mcux SDK 25.03.00 Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x
- RW61x
- RT595, RT1060, RT1170
- MCXW23

Minor Changes (bug fixes)

- [General] Various MISRA/Coverity fixes in framework: NVM, RNG, LowPower, SecLib and platform files

Services

- [SecLib_RNG] fix return status from RNG_GetTrueRandomNumber() function: return correctly gRngSuccess_d when RNG_entropy_func() function is successful
- [SFC] Allow the application to override the trig sample number parameter
- [Settings] Re-define the framework settings API name to avoid double definition when gSettingsRedefineApiName_c flag is defined

Platform specific

- [wireless_mcu] fwk_platform_sensors update :
  - Enable temperature measurement over ADC ISR
  - Enable temperature handling requested by NBU
- [wireless_mcu] fwk_platform_lcl coex config update for KW45
- [kw47_mcxw72] Change the default ppm_target of SFC algorithm from 200 to 360ppm

7.0.1 revB mcux SDK 25.03.00 Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x
- RW61x
- RT595, RT1060, RT1170
- MCXW23

Minor Changes (bug fixes)

General

- [General] Various MISRA/Coverity fixes in framework: NVM, RNG, LowPower, FunctionLib and platform files

Services

- [SecLib_RNG] AES-CBC evolution:
  - added AES_CBC_Decrypt() API for sw, SSS and mbedtls variants.
  - Made AES-CBC SW implementation reentrant avoiding use of static storage of AES block.
  - fixed SSS version to update Initialization Vector within SecLib, simplifying caller's implementation.
  - modified AES_128_CBC_Encrypt_And_Pad() so as to avoid the constraint mandating that 16 byte headroom be available at end of input buffer.
- [SecLib_RNG] RNG modifications:
  - RNG_GetPseudoRandomData() could return 0 in some error cases where caller expected a negative status.
    - Explicited RNG error codes
    - Added argument checks for all APIs and return gRngBadArguments_d (-2) when wrong
    - added checks of RNG initalization and return gRngNotInitialized_d (-3) when not done
    - fixed correcteness of RNG_GetPrngFunc() and RNG_GetPrngContext() relative to API description.
    - Added RNG_DeInit() function mostly for test and coverage purposes.
    - Improved RNG description in README.md
    - Unified the APIs behaviour between mbedtls and non mbedtls variants.
  - RNG/mbedtls : Prevent RNG_Init() from corrupting RNG entropy context if called more than once.
  - RNG/mbedtls: fixed RNG_GetTrueRandomNumber() to return a proper mbedtls_entropy_func() result.
  - Use defragmetation option when freeing key object in SecLib_sss to avoid leak in S200 memory
  - Add new API ECP256_IsKeyValid() to check whether a public key is valid
- [OtaSupport] Update return status to OTA_Flash_Success when success at the end of InternalFlash_WriteData() and InternalFlash_FlushWriteBuffer() APIs
- [WorQ] Implementing a simple workqueue service to the framework
- [SFC] Keep using immediate measurement for some measurement before switching to configuration trig to confirm the calibration made
- [DBG] Adding modules to framework DBG :
  - sbtsnoop
  - SWO
- [Common] Fix HAL_CTZ and HAL_RBIT IAR versions
- [LowPower] Fix wrong tick error calculation in case of infinite timeout

- [Settings] Add new macro gSettingsRedefineApiName_c to avoid multiple definition of settings API when using connectivity framework repo

Platform specific

- [KW47/MCXW72] Change xtal cload default value from 4 to 8 in order to increase the precision of the link layer timebase in NBU
- [wireless_mcu] [wireless_nbu] Use new WorkQ service to process framework intercore messages
- [rw61x] Fix HCI message sending failure in some corner case by releasing controller wakes up after that the host has send its HCI message
- [MCXW23] Adding the initial support of MCXW23 into the framework

7.0.0 mcux SDK 24.12.00 Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x
- RW61x
- RT595, RT1060, RT1170

Minor Changes (bug fixes)

Platform specific

- [RW61X]
  - Add MCUX_COMPONENT_middleware.wireless.framework.platform.rng to the platform to fix a warning at generation
  - Retrieve IEEE 64 bits address from OTP memory
- [KW45x, MCXW71x, KW47x, MCXW72x]
  - Ignore the secure bit from RAM addresses when comparing used ram bank in bank retention mechanism
  - Add gPlatformNbuDebugGpioDAccessEnabled_d Compile Macro (enabled by default). Can be used to disable the NBU debug capability using IOs in case Trustzone is enabled (“PLATFORM_InitNbu()‘ code executed from unsecure world).
  - Fix in NBU firmware when sending ICS messages gFwkSrvNbuApiRequest_c (from controller_api.h API functions)

Services

- [OTA]
  - Add choice name to OtaSupport flash selection in Kconfig
- [NVM]
  - Add gNvmErasePartitionWhenFlashing_c feature support to gcc toolchain
- [SecLib_RNG]
  - Misra fixes

7.0.0 revB mcux SDK 24.12.00 Supported platforms: KW45x, KW47x, MCXW71, MCXW72, K32W1x, RW61x, RT595, RT1060, RT1170

Major Changes (User Applications may be impacted)

- mcux github support with cmake/Kconfig from sdk3 user shall now use CmakeLists.txt and Kconfig files from root folder. Compilation should be done using west build command. In order to see the Framework Kconfig, use command >west build -t guiconfig
- Board files and linker scripts moved to examples repository

Bugfixes

- [platform lowpower]
  - Entering Deep down power mode will no longer call PLATFORM_EnterPowerDown(). This API is now called only when going to Power down mode

Platform specific

- [KW47/MCXW72]: Early access release only
  - Deep sleep power mode not fully tested. User can experiment deep sleep and deep down modes using low power reference design applications
  - XTAL32K-less support using FRO32K not tested
- [KW45/MCXW71/K32W148]
  - Deep sleep mode is supported. Power down mode is supported in low power reference design applications as experimental only
  - XTAL32K-less support using FRO32K is experimental - FRO32K notifications callback is debug only and should not be used for mass production firmware builds

Minor Changes (no impact on application)

- Overall folder restructuring for SDK3
  - [Platform]:
    - Rename platform_family from connected_mcu/nbu to wireless_mcu/nbu
    - platform family have now a dedicated fwk_config.h, rpmsg_config.h and SecLib_mbedtls_config.h
  - [Services]
    - Move all framework services in a common directory “services/”

7.0.0 revA: KW45/KW47/MCX W71/MCX W72/K32W148

Experimental Features only

- Power down on application power domain: Some tests have shown some failure. Power consumption higher than Deep Sleep. => This feature is not fully supported in this release
- XTAL32K less board with FRO32K support: Some additional stress tests are under progress.
- FRO32K notifications callback is for debug only and shall not be used for production. User shall not execute long processing (such as PRINTF) as it is executed in ISR context.

Main Changes

- Cmake/Kconfig support for SDK3.0
- [Sensors] API renaming:
  - SENSORS_InitAdc() renamed to SENSORS_Init()
  - SENSORS_DeinitAdc() remamed to SENSORS_Deinit()
- [HWparams]
  - Repair PROD_DATA sector in case of ECC error (implies loss of previous contents of sector)
- [NVM] Linker script modification for armgcc whenever gNvTableKeptInRam_d option is used:
  - placement of NVM_TABLE_RW in data initialized section, providing start and end address symbols. For details see NVM_Interface.h comments.
- [OtaSupport]
  - OTA_Initialize(): now transitions the image state from RunCandidate to Permanent if not done by the application. OTA module shall always be initialized on a Permanent image, this change ensures it is the case.
  - OTA_MakeHeadRoomForNextBlock(): now erases the OTA partition up to the image total size (rounded to the sector) if known.

Minor changes

- [Platform]
  - Updated macro values: -kw47: BOARD_32MHZ_XTAL_CDAC_VALUE from 12U to 16U, BOARD_32MHZ_XTAL_ISEL_VALUE from 7U to 11U, BOARD_32KHZ_XTAL_CLOAD_DEFAULT from 8U to 4U, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT from 1U to 3U
    - MCX W72 (low-power reference design applications only): BOARD_32MHZ_XTAL_CDAC_VALUE from 12U to 10U, BOARD_32MHZ_XTAL_ISEL_VALUE from 7U to 11U, BOARD_32KHZ_XTAL_CLOAD_DEFAULT from 8U to 4U, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT from 1U to 3U
  - New PLATFORM_RegisterNbuTemperatureRequestEventCb() API: register a function callback when NBU request new temperature measurement. API provides the interval request for the temperature measurement
  - Update PLATFORM_IsNbuStarted() API to return true only if the NBU firmware has been started.
- [platform lowpower]
  - Move RAM layout values in fwk_platform_definition.h and update RAM retention API for KW47/MCXW72

Bugfixes

- [OtaSupport]
  - OTA_MakeHeadRoomForNextBlock(): fixed a case where the function could try to erase outside the OTA partition range.

6.2.4: KW45/K32W1x/MCXW71/RX61x SDK 2.16.100 This release does not contain the changes from 6.2.3 release.

This release contains changes from 6.2.2 release.

Main Change

- armgcc support for Cmake sdk2 support and VS code integration

Minor changes

- [NBU]
  - Optimize some critical sections on nbu firmware
- [Platform]
  - Optimize PLATFORM_RemoteActiveReq() execution time.

6.2.3: KW47 EAR1.0 Initial Connectivity Framework enablement for KW47 EAR1.0 support.

New features

- OpenNBU feature : nbu_ble project is available for modification and building

Supported features

- Deep sleep mode

Unsuported features

- Power down mode
- FRO32K support (XTAL32K less boards)

Main changes

- [NBU]
  - LPTMR2 available and TimerManager initialization with Compile Macro: gPlatformUseLptmr_d
  - NBU can now have access to GPIOD
  - SW RNG and SW SecLib ported to NBU (Software implementation only)
- [RNG]
  - Obsoleted API removed : FWK_RNG_DEPRECATED_API
  - RNG can be built without SecLib for NBU, using gRngUseSecLib_d in fwk_config.h
  - Some API updates:
    - RNG_IsReseedneeded() renamed to RNG_IsReseedNeeded,
    - RNG_TriggerReseed() renamed to RNG_NotifyReseedNeeded(),
    - RNG_SetSeed() and RNG_SetExternalSeed() return status code.
  - Optimized Linear Congruential modulus computation to reduce cycle count.

Minor changes

- [NVM]
  - Optimize NvIsRecordErased() procedure for faster garbage collection
  - MISRA fix : Remove externs and weaks from NVM module - Make RNG and timer manager dependencies conditional
- [Platform]
  - Allow the debugger to wakeup the KW47/MCXW72 target

6.2.2: KW45/K32W1 MR6 SDK 2.16.000 Experimental Features only:

- Power down on application power domain : Some tests have shown some failure. Power consumption higher than Deep Sleep. => This feature is not fully supported in this release
- XTAL32K less board with FRO32K support : Some additional stress tests are under progress.
- FRO32K notifications callback is for debug only and shall not be used for production. User shall not execute long processing (such as PRINTF) as it is executed in ISR context.

Changes

- [Board] Support for freedom board FRDM-MCX W7X
- [HWparams]
  - Support for location of HWParameters and Application Factory Data IFR in IFR1
  - Default is still to use HWparams in Flash to keep backward compatibility
- [RNG]: API updates:
  - New APIS RNG_IsReseedneeded(), RNG_SetSeed() to provide See to PRNG on NBU/App core - See BluetoothLEHost_ProcessIdleTask() in app_conn.c
  - New APIs RNG_SetExternalSeed() : User can provide external seed. Typically used on NBU firmrware for App core to set a seed to RNG. RNG_TriggerReseed() : Not required on App core. Used on NBU only.
- [NVS] Wear statistics counters added - Fix nvs_file_stat() function
- [NVM] fix Nv_Shutdown() API
- [SecLib] New feature AES MMO supported for Zigbee

6.2.2: RW61x RFP4 SDK 2.16.000

- [Platform] Support Zigbee stack
- [OTA] Add support for RW61x OTA with remap feature.
  - Required modifications to prevent direct access to flash logical addresses when remap is active.
  - Image trailers expected at different offset with remap enabled (see gPlatformMcuBootUseRemap_d in fwk_config.h)
  - fixed image state assessment procedure when in RunCandidate.
- [NVS] Wear statistics counters added
- [SecLib] New feature AES MMO supported for Zigbee
- [Misra] various fixes

6.2.1: KW45/K32W1 MR5 SDK 2.15.000 Experimental Features only:

- Power down on application power domain : Some tests have shown some failure. This feature is not fully supported in this release
- XTAL32K less board with FRO32K support : Some additional stress tests are under progress. Timing variation of the timebase are being analyzed

Major changes

- [RNG]: API updates
  - New compile flag to keep deprecated API: FWK_RNG_DEPRECATED_API
  - change return error code to int type for RNG_Init(), RNG_ReInit()
  - New APIs RNG_GetTrueRandomNumber(), RNG_GetPseudoRandomData()
- [Platform]
  - fwk_platform_sensors
    - Change default temperature value from -1 to 999999 when unknown
  - fwk_platform_genfsk
    - rename from platform_genfsk.c/h to fwk_platform_genfsk.c/h
  - platform family
    - Rename the framework platform folder from kw45_k32w1 to connected_mcu to support other platform from the same family
  - fwk_platform_intflash
    - Moved from fwk_platform files to the new fwk_platform_intflash files the internal flash dependant API
- [NBU]
  - BOARD_LL_32MHz_WAKEUP_ADVANCE_HSLOT changed from 2 to 3 by default
  - BOARD_RADIO_DOMAIN_WAKE_UP_DELAY changed from 0x10 to 0x0F
- [gcc linker]
  - Exclude k32w1_nbu_ble_15_4_dyn.bin from .data section

Minor Changes

- [Platform]
  - PLATFORM_GetTimeStamp(0 has an important fix for reading the Timestamp in TSTMR0
  - New API PLATFORM_TerminateCrypto(), PLATFORM_ResetCrypto() called from SecLib for lowpower exit
  - Fix when enable fro debug callback on nbu
- [DBG]
  - SWO
    - Add new files fwk_debug_swo.c/h to use SWO for debug purpose
    - Two new flags has been added:
      - BOARD_DBG_SWO_CORE_FUNNEL to chose on which core you want to use SWO

        · BOARD_DBG_SWO_PIN_ENABLE to enable SWO on a pin

- [NVS]
  - Add support of NVS and Settings in framework
- [NBU]
  - Fix power down issues and reduce critical section on NBU side:
    * new API PLATFORM_RemoteActiveReqWithoutDelay() called from NBU functions where waiting delay is not required
    * Increase delay needed in power down for OEM part to request the SOC to be active
    * Remove unnecessary code to PLATFORM_RemoteActiveReqWithoutDelay() from PLATFORM_HciRpmsgRxCallback()
    * Improve nbu memory allocation failure debug messages
- [SDK]
  - Multicore: remove critical section in HAL_RpmsgSendTimeout() (only required in FPGA HDI mode)
  - Flash drivers: update for ECC detection
- [Platform]
  - fwk_platform_sensors
    * Fix temperature reporting to NBU
  - fwk_platform_extflash
    * Align .c and .h prototype of PLATFORM_ExternalFlashAreaIsBlank() function
- [NVM]
  - Keep Mutex in NvModuleDeInit(). In Bare metal OS, Mutex can not be destroyed
  - New API NvRegisterEccFaultNotificationCb() to register Notification callback when Ecc error happens in FileSystem
- [MISRA] fixes
  - SecLib_sss.c: ECDH_P256_ComputeDhKey()
  - fwk_platform_extflash.c: PLATFORM_IsExternalFlashPageBlank()
  - fwk_fs_abstraction.c: Various fixes
- [HWparams]
  - Fix on if condition when gHwParamsProdDataPlacementLegacy2IfrMode_c mode is selected
- [OTA]
  - Enable gOtaCheckEccFaults_d by default to avoid bus in case of ECC error during OTA
  - Fix OTA partition overflow during OTA stop and resume transfer
- [BOARD]
  - Place code button or led specific under correct defines in board_comp.c/h
  - Bring back MACROs BOARD_INITRFSWITCHCONTROLPINS in pin_mux header file of the loc board
- [SecLib]
  - Add some undefinition in SecLib_mbedtls_config as new dependency has been added in mbedtls repo:

    * MBEDTLS_SSL_CBC_RECORD_SPLITTING, MBEDTLS_SSL_PROTO_TLS1, MBEDTLS_SSL_PROTO_TLS1_1

- [FRO32K]
  - FRO32K notification callback PLATFORM_FroDebugCallback_t() has new parameter to report he fro_trim value
  - maxCalibrationIntervalMs value can be provided to NBU using PLATFORM_FwkSrvSetRfSfcConfig()
- [Sensors]
  - fix: PLATFORM_GetTemperatureValue() shall have NBU started to send temperature to NBU

6.2.1: RW61x RFP3

- [NVS]
  - Add support of NVS and Settings in framework
- [MISRA] fixes
  - board_lp.c BOARD_UninitDebugConsole() and BOARD_ReinitDebugConsole()
  - fwk_platform_ble.c: Various fixes
- [OTA]
  - Fix OTA partition overflow during OTA stop and resume transfer

6.2.0: RT1060/RT1170 SDK2.15 Major

6.1.8: KW45/K32W1 MR4

- [BOARD PLATFORM]
  - Move gBoardUseFro32k_d to board_platform.h file
  - Offer the possibility to change the source clock accuracy to gain in power consumption
- [BOARD LP]
  - Move PLATFORM_SetRamBanksRetained() at end of BOARD_EnterLowPowerCb() in case a memory allocation is done previously in this function
  - fix low power, increase BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0 to 0x10 - Skip this delay when App requesting NBU wakeup
- [PLATFORM]
  - fwk_platform_ble.c/h: New timestamp API that returns the difference between the current value of the LL clock and the argument of the function
  - fwk_platform.c/h:
    * New PLATFORM_EnableEccFaultsAPI_d compile flag: Enable APIs for interception of ECC Fault in bus fault handler
    * New gInterceptEccBusFaults_d compile flag: Provide FaultRecovery() demo code for bus fault handler to Intercept bus fault from Flash Ecc error
- [LOC]
  - Incorrect behavior for set_dtest_page (DqTEST11 overridden)
  - Fix SW1 button wake able on Localization board

  - Fix yellow led not properly initialized
  - Format localization pin_mux.c/h files
- [Inter Core]
  - Affect values to enumeration giving the inter core service message ids
  - Shared memory settings shared between both cores
  - Add callback to register when NBU has unrecoverable Radio issue
- [NVM]
  - Add NV_STORAGE_MAX_SECTORS, NV_STORAGE_SIZE as linker symbol for alignment with other toolchain
  - ECC detection and recovery. New gNvSalvageFromEccFault_d and gNvVerifyReadBackAfterProgram_d compile flags. Please refer to ECC Fault detection section in README.md file located in NVM folder
- [OTA]
  - Prevent bus fault in case of ECC error when reading back OTA_CFR update status (disable by default)
- [SecLib]
  - Shared mutex for RNG and SecLib as they share same hardware resource
- [Key storage]
  - Fix to ignore the garbage at the end of buffers
  - Detect when buffers are too small in KS_AddKey() functions
- [FileCache]
  - Fix deadlock in Filecache FC_Process()
- [SDK]
  - Applications: remove definition of stack location and use default from linker script, fix warmboot stack in freertos at 0x20004000
  - Memory Manager Light:
    * fix Null pointer harfault when MEM_STATISTICS_INTERNAL enable
    * Fix MemReinitBank() on wakeup from lowpower when Ecc banks are turned off

6.1.7: KW45/K32W1 MR3

- [OTA]
  - New API OTA_SetNewImageFlagWithOffset()
  - Fix StorageBitmapSize calculation
  - OTA clean up: Removed OTA_ValidateImage()
- [Low Power]
  - New linker Symbol m_lowpower_flag_start in linker file.
    * Flag is used to indicate NBU that Application domain goes to power down mode. Keep this flag to 0 if only Deep sleep is supported
    * This flag will be set to 1 if Application domain goes to power down mode
  - Re-introduce PWR_AllowDeviceToSleep()/PWR_DisallowDeviceToSleep(), PWR_IsDeviceAllowedToSleep() API

  - Implement tick compensation mechanism for idle hook in a dedicated freertos utils file fwk_freertos_utils.[ch], new functions: FWK_PreIdleHookTickCompensation() and FWK_PostIdleHookTickCompensation
  - Rework timestamping on K4W1
    * PLATFORM_GetMaxTimeStamp() based on TSTMR
    * Rename PLATFORM_GetTimestamp() to PLATFORM_GetTimeStamp()
    * Update PLATFORM_Delay(): Rework to use TSTMR instead of LPTMR for platform_delay
    * Update PLATFORM_WaitTimeout(): Fixed a bug in PLATFORM_WaitTimeout() related to timer wrap
    * Add PLATFORM_IsTimeoutExpired() API
  - Fix race condition in PWR_EnterLowPower(), masking interrupts in case not done at upper layer
  - Low power timer split in new files fwk_platform_lowpower_timer.[ch]
  - New PWR_systicks_bm.c file for bare metal usage: implement SysTick suspend/resume functionality, New weak PWR_SysTicksLowPowerInit()
- [FRO32K]
  - Improve FRO32K calibration in NBU
  - create PLATFORM_InitFro32K() to initialize FRO32K instead of XTAL32K (to be called from hardware_init())
  - update FRO32K README.md file in SFC module
  - Debug:
  - Add Notification callback feature for SFC module FRO32K
  - Linker script update to support m_sfc_log_start in SMU2
- [SecLib]
  - Remove gSecLibSssUseEncryptedKeys_d compile option, split Secure/Unsecure APIs
  - RNG update to use same mutex than SecLib
  - Fix AES_128_CBC_Encrypt_And_Pad length
  - Implement RNG_ReInit() for lowpower
  - Fix issue in ECDH_P256_GenerateKeys() when waking up from power down
  - Call CRYPTO_ELEMU_reset() from SecLib_reInit() for power down support
- [BOARD]
  - Create new board_platform.h file for all Board characteristics settings (32Mhz XTAL, 32KHZ XTAL, etc..)
  - TM_EnterLowpower() TM_EnterLowpower() to be called from LP callbacks
  - Support Localization boards, Only BUTTON0 supported
    * New compile flag BOARD_LOCALIZATION_REVISION_SUPPORT
    * New pin_mux.[ch] files
  - Offer the possibility to override CDAC and ISEL 32MHz settings before the initialization of the crystal in board_platform.h
    * new BOARD_32MHZ_XTAL_CDAC_VALUE, BOARD_32MHZ_XTAL_ISEL_VALUE
    * BOARD_32MHZ_XTAL_TRIM_DEFAULT obsoleted

- [NVM file system]
  - Look ahead in pending save queue - Avoid consuming space to save outdated record
  - Fix NVM gNvDualImageSupport feature in NvIsRecordCopied
- [Inter Core]
  - Change PLATFORM_NbuApiReq() API return parameters granularity from uint32 to uint8
  - MAX_VARIANT_SZ change from 20 to 25
  - Set lp wakeup delay to 0 to reduce time of execution on host side, NBU waits XTAL to be ready before starting execution
  - Update inter core config rpmsg_config.h
  - Add timeout to while loops that relies on hardware in RemoteActiveReq(), Application can register Callbacks when timeout
  - Return non-0 status when calling PLATFORM_FwkSrvSendPacket when NBU non started
  - Let PLATFORM_GetNbuInfo return -10 if response not received on timeout - Doxygen platform_ics APIs
- [HW params]
  - New compile Macro for HW params placement in IFR - Save 8K in FLash: gHwParamsProdDataPlacement_c . 3 modes:
  - Legacy placement, move from legacy to IFR, IFR only placement
  - New compile Macro for Application data to be stored with HW params (in shared flash sector): gHwParamsAppFactoryDataExtension_d, New APIs:
    - Nv_WriteAppFactoryData(), Nv_GetAppFactoryData()
  - See HWParameter.h
- [Platform]
  - Implement PLATFORM_GetIeee802_15_4Addr() API in fwk_platform_ot.c - New gPlatformUseUniqueDeviceIdFor15_4Addr_d compile Macro
  - Wakeup NBU domain when reading RADIO_CTRL UID_LSB register in PLATFORM_GenerateNewBDAddr()
- [Reset]
  - New reset Implementations using Deep power down mode or LVD:
    - new files fwk_platform_reset.[ch]
    - new APIs: PLATFORM_ForceDeepPowerDownReset(), PLATFORM_ForceLvdReset() + reset on ext pins
    - new compile flags: gAppForceDeepPowerDownResetOnResetPinDet_d and gAppForceLvdResetOnResetPinDet_d to reset on external pins
- [FSCI]
  - fix when gFsciRxAck_c enabled
  - integrate new reset APIs

6.1.4: RW610/RW612 RFP1

- [Low Power]
  - Added support of low power for OpenThread stack.
  - Added PWR_AllowDeviceToSleep/PWR_DisallowDeviceToSleep/PWR_IsDeviceAllowedToSleep APIs.
- [platform]
  - Added PLATFORM_GetMaxTimeStamp API.
  - Fixed high impact Coverity.
- [FreeRTOS]
  - Created a new utilities module for FreeRTOS: fwk_freertos_utils.c/h.
  - Implemented a tick compensation mechanism to be used in FreeRTOS idle hook, likely around flash operations. This mechanism aims to estimate the number of ticks missed by FreeRTOS in case the interrupts are masked for a long time.

6.1.4: KW45/K32W1 MR2

- [Low power]
  - Powerdown mode tested and enabled on Low Power Reference Design applications
  - XTAL32K removal functionality using FRO32K, supported from NBU firmwares - limitation: Application domain supports Deep Sleep only (not power down)
  - NBU low power improvement: low power entry sequence improvement and system clock reduction to 16Mhz during WFI
  - Wake up time from cold boot, reset, power switch greatly improved. Device starts on FRO32K, switch to XTAL32K when ready if gBoardUseFro32k_d not set
  - Bug fixes:
    - Move PWR LowPower callback to PLATFORM layers
    - Fix wrong compensation of SysTicks
    - Reinit system clocks when exiting power down mode: BOARD_ExitPowerDownCb(), restore 96MHz clock is set before going to low power
    - Call Timermanager lowpower entry exit callbacks from PLATFORM_EnterLowPower()
    - Update PLATFORM_ShutdownRadio() function to force NBU for Deep power down mode
  - K32W1:
    - Support lowpower mode for 15.4 stacks
- [NVM]
  - New Compilation MACRO gNvDualImageSupport to support multiple firmware image with different register dataset
  - Change default configuration gNvStorageIncluded_d to 1, gNvFragmentation_Enabled_d to 1, gUnmirroredFeatureSet_d to TRUE
  - Some MISRA issues for this new configuration.
  - Remove deprecated functionality gNvUseFlexNVM_d
- [SecLib]

  - New NXP Ultrafast ecp256 security library:
    * New optimized API for ecdh DhKey/ecp256 key pair computation: Ecdh_ComputeDhKeyUltraFast(), ECP256_GenerateKeyPairUltraFast().
    * New macro gSecLibUseDspExtension_d.
    * Improved software version of Seclib with Ultrafast library for ECP256_LePointValid()
  - Bug fixes:
    * Share same mutex between Seclib and RNG to prevent concurrent access to S200
    * Optimized S200 re-initialization, restore ecdh key pair after power down
    * Fixed race condition when power down low power entry is aborted
    * Endianness function updates and clean up
- [OTA]
  - OTASupport improvements:
    * New API OTA_GetImgState(), OTA_UpdateImgState()
    * OTASupport and fwk_platform_extflash API updates for external flash: OTA_SelectExternalStoragePartition(), PLATFORM_IsExternalFlashSectorBlank(), PLATFORM_IsExternalFlashPageBlank(), PLATFORM_OtaGetOtaPartitionConfig()
    * Updated OtaExternalFlash.c, 2 new APIs in fwk_platform_extflash.c
    * Removed unused FLASH_op_type and FLASH_TransactionOpNode_t definitions from public API
    * Removed unused InternalFlash_EraseBlock() from OtaInternalFlash.c
- [NBU firmware]
  - Mechanism to set frequency constraint to controller from the host PLATFORM_SetNbuConstraintFrequency()
  - NbuInfo has one more digit in versionBuildNo field
- [Board]
  - Support Extflash low power mode, add BOARD_UninitExternalFlash(), PLATFORM_UninitExternalFlash(), PLATFORM_ReinitExternalFlash()
  - Support XTAL32K removal functionatity, use FRO32K instead by setting gBoardUseFro32k_d to 1 in board.h file
  - Support localization boards KW45B41Z-LOC Rev C
  - Low power improvement: New BOARD_InitPins() and BOARD_InitPinButtonBootConfig() called from hardware_init.c
  - Removed KW45_A0_SUPPORT support (dcdc)
  - Bug fixes:
    * Fixed glitches on the serial manager RX when exiting from power down
    * Fixed ADC not deinitialized in clock gated modes in BOARD_EnterLowPowerCb()
    * Fixed UART output flush when going to low power: BOARD_UninitAppConsole()
- [platform]
  - PLATFORM_InitBle(), PLATFORM_SendHci() can now block with timeout if NBU does not answer. Application can register callback function to be notified when it occurs: PLATFORM_RegisterBleErrorCallback()

  - Added API to set and get 32Khz XTAL capacitance values: PLATFORM_GetOscCap32KValue() and PLATFORM_SetOscCap32KValue()
  - Added new Service FWK call gFwkSrvNbuMemFullIndication_c to get NBU mem full indication, register with PLATFORM_RegisterNbuMemErrorCallback()
  - Added support negative value in platform intercore service
- [linker script]
  - Realigned gcc linker script with IAR linker script.
  - Added possibility to redefine cstack_start position
  - Added Possibility to change gNvmSectors in gcc linker script
  - Added dedicated reserved Section in shared memory for LL debugging
- [FreeRTOSConfig.h]
  - Removed unused MACRO configFRTOS_MEMORY_SCHEME and configTOTAL_HEAP_SIZE
- [HW Param]
  - Added xtalCap32K field to store XTAL32K triming value
- [fwk_hal_macros.h]
  - Added MACRO for KB, MB and set, clear bits in bit fields
- [Debug]
  - Added MACROs for performance measurement using DWT: DBG_PERF_MEAS

6.1.3 KW45 MR1 QP1

- [Initialization] Delay the switch to XTAL32K source clock until the BLE host stack is initialized
- [lowpower] NBU wakeup from lowpower: configuration can now be programmed with BOARD_NBU_WAKEUP_DELAY_LPO_CYCLE, BOARD_RADIO_DOMAIN_WAKE_UP_DELAY in board.h file
- [NBU firmware] Major fix for NBU system clock accuracy
- [clock_config]
  - Update SRAM margin and flash config when switching system frequency
  - Trim FIRC in HSRUN case
- [XTAL 32K trim] XTAL 32K configuration can be tuned in board.h file with BOARD_32MHZ_XTAL_TRIM_DEFAULT, BOARD_32KHZ_XTAL_CLOAD_DEFAULT, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT
- [MAC address] Add OUI field in PLATFORM_GenerateNewBDAddr() when using Unique Device Id

6.1.2: RW610/RW612 PRC1

- [Low Power]
  - Updates after SDK Power Manager files renaming.
  - Moved PWR LowPower callback to PLATFORM layers.
  - Bug fixes:
    * Fixed wrong compensation of SysTicks during tickless idle.

        * Reinit RTC bus clock after exit from PM3 (power down).
- [OTA]
  - Initial support for OTA using the external flash.
- [platform]
  - Implemented platform specific time stamp APIs over OSTIMER.
  - Implemented platform specific APIs for OTA and external flash support.
  - Removed PLATFORM_GetLowpowerMode API.
  - Added support of CPU2 wake up over Spinel for OpenThread stack.
  - Bug fixes:
    * Fixed issues related to handling CPU2 power state.
- [board]
  - Updated flash_config to support 64MB range.
- [linker script]
  - Fixed wrong assert.

6.1.1: KW45/K32W1 MR1

- [platform] Use new FLib_MemSet32Aligned() to write in ECC RAM bank to force ECC calculation in the MEM_ReinitRamBank() function
- [FunctionLib] Implement new API to set a word aligned
- [platform] Set coarse amplifier gain of the oscilattor 32k to 3
- [platform] Switch back to RNG for MAC Adress generation
- [SecLib] Get rid of the lowpower constraint of deep sleep in ECDH API
- [DCDC] Set DCDC output voltage to 1.35V in case LDO core is set to 1.1V to ensure a drop of 250mV between them
- [NVM] NvIdle() is now returning the number of operations that has been executed
- [documentation] Add markdown of each framework module by default on all package
- [LowPower] Add a delay advised by hardware team on exit of lowpower for SPC
- [SecLib] Rework of SecLib_mbedTLS ECDH functions
- [OTA] Make OTA_IsTransactionPending() public API
- [FunctionLib] Change prototype of FLib_MemCpyWord(), pDst is now a void* to permit more flexibility
- [NVM] Add an API to know if there is a pending operation in the queue
- [FSCI] Fix wrong error case handling in FSCI_Monitor()

6.1.0: KW45/K32W1 RFP

- [LowPower] Do not call PLATFORM_StopWakeUpTimer() in PWR_EnterLowPower() if PLATFORM_StartWakeUpTimer() was not previously called
- [boards] Add the possibility to wakeup on UART 0 even if it is not the default UART
- [boards] Add support for Hardware flow control for UART0, Enable with gBoardUseUart0HwFlowControl, Pin mux update with two additional API for RTS, CTS pins

- [Sensors] Improve ADC wakeup time from deep sleep state: use save and restore API for ADC context before/after deep sleep state.
- [linker script] update SMU2 shared memory region layout with NBU: increase sqram_btblebuf_size to support 24 connections. Shared memory region moved to the end
- [SecLib] SecLib_DeriveBluetoothSKD() API update to support if EdgeLock key shall be re-generated

6.0.11: KW45/K32W1 PRC3.1

FSCI: Framework Serial Communication Interface

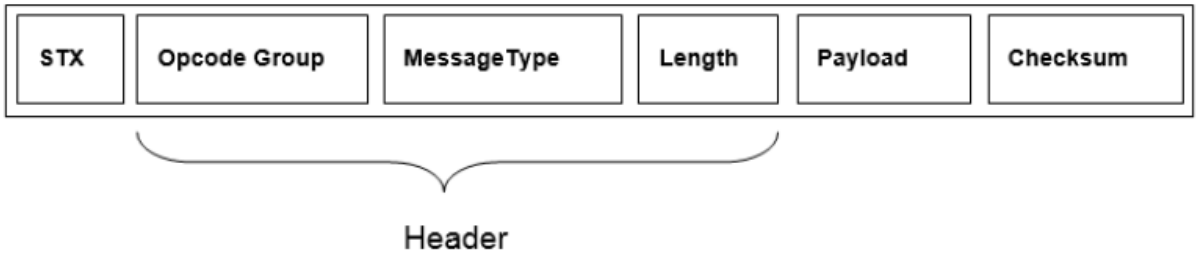
Overview The Framework Serial Communication Interface (FSCI) is both a software module and a protocol that allows monitoring and extensive testing of the protocol layers. It also allows separation of the protocol stack between two protocol layers in a two processing entities setup, the host processor (typically running the upper layers of a protocol stack) and the Black Box application (typically containing the lower layers of the stack, serving as a modem). The Test Tool software is an example of a host processor, which can interact with FSCI Black Boxes at various layers. In this setup, the user can run numerous commands to test the Black Box application services and interfaces.

The FSCI enables common service features for each device enables monitoring of specific interfaces and API calls. Additionally, the FSCI injects or calls specific events and commands into the interfaces between layers.

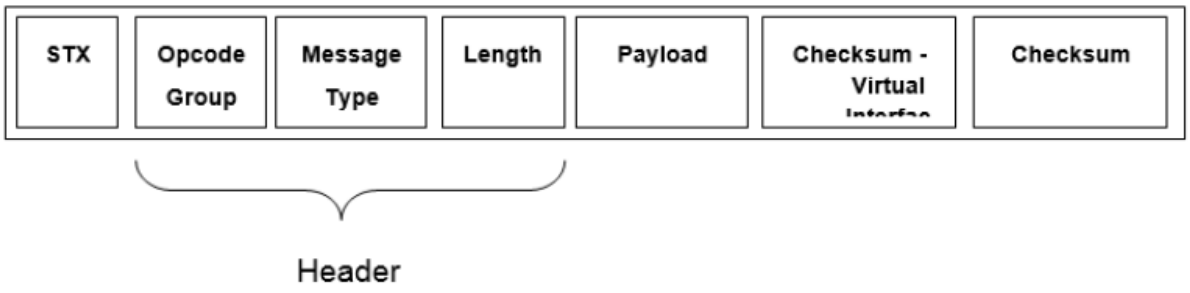
An entity which needs to be interfaced to the FSCI module can use the API to register opcodes to specific interfaces. After doing so, any packet coming from that interface with the same opcode triggers a callback execution. Two or more entities cannot register the same opcode on the same interface, but they can do so on different interfaces. For example, two MAC instances can register the same opcodes, one over UARTA, and the other over UARTB. This way, Test Tool can communicate with each MAC layer over two UART interfaces.

The FSCI module executes either in the context of the Serial Manager task or owns its dedicated task if the compilation Macro *gFsciUseDedicatedTask_c* is set to 1.

FSCI packet structure The FSCI module sends and receives messages as shown in the figure below. This structure is not specific to a serial interface and is designed to offer the best communication reliability. The Black Box device expects messages in little-endian format. It also responds with messages in little-endian format.



Below is an illustration of the FSCI packet structure when a virtual interface is used instead :



Field name	Length (bytes)	Description
STX	1	Used for synchronization over the serial interface. The value is always 0x02.
Opcode Group	1	Distinguishes between different Service Access Primitives (for example MLME or MCPS).
Message Type	1	Specifies the exact message opcode that is contained in the packet.
Length	1 or 2	The length of the packet payload, excluding the header and FCS. The length field content must be provided in little-endian format.
Payload	variable	Payload of the actual message.
Checksum	1	Checksum field used to check the data integrity of the packet.
Checksum2	0 or 1	The second CRC field appears only for virtual interfaces.

NOTE : When virtual interfaces are used, the first checksum is decremented with the ID of the interface. The second checksum is used for error detection.

constant definition The following Macro configurs the FSCI module

```
#define gFsciIncluded_c 0 /* Enable/Disable FSCI module */
#define gFsciUseDedicatedTask_c 1 /* Enable Fsci task to avoid recursivity in Fsci module (Misra␣
↪compliant) */
#define gFsciMaxOpGroups_c 8
#define gFsciMaxInterfaces_c 1
#define gFsciMaxVirtualInterfaces_c 0
#define gFsciMaxPayloadLen_c 245 /* bytes */
#define gFsciTimestampSize_c 0 /* bytes */
#define gFsciLenHas2Bytes_c 0 /* boolean */
#define gFsciUseEscapeSeq_c 0 /* boolean */
#define gFsciUseFmtLog_c 0 /* boolean */
#define gFsciUseFileDataLog_c 0 /* boolean */
#define gFsciLoggingInterface_c 1 /* [0..gFsciMaxInterfaces_c) */
#define gFsciHostMacSupport_c 0 /* Host support at MAC layer */
```

The following provides the OpGroups values reserved by MAC, application, and FSCI.

FSCI Host FSCI Host is a functionality that allows separation at a certain stack layer between two entities, usually two boards running separate layers of a stack.

Support is provided for functionality at the MAC layer, for example, MAC/PHY layers of a stack are running as a Black Box on a board, and MAC higher layers are running on another. The higher layers send and receive serial commands to and from the MAC Black Box using the FSCI set of operation codes and groups.

The protocol of communication between the two is the same. The current level of support is provided for:

- FSCI_MsgResetCPUReqFunc – sends a CPU reset request to black box
- FSCI_MsgWriteExtendedAdrReqFunc – configures MAC extended address to the Black Box
- FSCI_MsgReadExtendedAdrReqFunc – N/A

The approach on the Host interfacing a Black Box using synchronous primitives is by default the polling of the FSCI_receivePacket function, until the response is received from the Black Box. The calling task polls whenever the task is being scheduled. This is required because a stack synchronous primitive requires that the response of that request is available in the context of the caller right after the SAP call has been executed.

The other option, available for RTOS environments, is using an event mechanism. The calling task blocks waiting for the event that is sent from the Serial Manager task when the response is available from the Black Box. This option is disabled by default. The disadvantage of this option is that the primitive cannot be received from another Black Box through a serial interface because the blocked task is the Serial Manager task, which reaches a deadlock as cannot be released again.

FSCI ACK ACK transmission is enabled through the gFsciTxAck_c macro definition. Each FSCI valid packet received triggers an FSCI ACK packet transmission on the same FSCI interface that the packet was received on. The serial write call is performed synchronously to send the ACK packet before any other FSCI packet. Only then the registered handler is called to process the received packet. The ACK is represented by the gFSCI_CnfOpcodeGroup_c and mFsciMsgAck_c Opcode. An additional byte is left empty in the payload so that it can be used optionally as a packet identifier to correlate packets and ACKs. ACK reception is the other component that is enabled through gFsciRxAck_c. The behavior is such that every FSCI packet sent through a serial interface triggers an FSCI ACK packet reception on the same interface after the packet is sent. If an ACK packet is received, the transmission is considered successful. Otherwise, the packet is resent a number of times. The ACK wait period is configurable through mFsciRxAckTimeoutMs_c and the number of transmission retries through mFsciTxRetryCnt_c. The ACK mechanism described above can also be coupled with a FSCI packet reception timeout enabled through gFsciRxTimeout_c and configurable through mFsciRxRestartTimeoutMs_c. Whenever there are no more bytes to be read from a serial interface, a timeout is configured at the predefined value if no other bytes are received. If new bytes are received, the timer is stopped and eventually canceled at successful reception. However, if, for any reason, the timeout is triggered, the FSCI module considers that the current packet is invalid, drops it, and searches for a new start marker.

FSCI usage example Detailed data types and APIs are described in ConnFWK API documentation.

Initialization

```
/* Configure the number of interfaces and virtual interfaces used */
#define gFsciMaxInterfaces_c 4
#define gFsciMaxVirtualInterfaces_c 2
....
/* Define the interfaces used */
static const gFsciSerialConfig_t myFsciSerials[] = {
    /* Baudrate, interface type, channel No, virtual interface */ {gUARTBaudRate115200_c, gSerialMgrUart_
↪c, 1, 0}, {gUARTBaudRate115200_c, gSerialMgrUart_c, 1, 1}, {0 , gSerialMgrIICSlave_c, 1, 0}, {0 ,␣
↪gSerialMgrUSB_c, 0, 0},
};
....
/* Call init function to open all interfaces */
FSCI_Init( (void*)mFsciSerials );
```

Registering operation groups

```
myOpGroup = 0x12; // Operation Group used
myParam = NULL; // pointer to a parameter to be passed to the handler function (myHandlerFunc)
myInterface = 1; // index of entry from myFsciSerials
...
FSCI_RegisterOpGroup( myOpGroup, gFsciMonitorMode_c, myHandlerFunc, myParam, myInterface );
```

Implementing handler function

```
void fsciMcpsReqHandler(void *pData, void* param, uint32_t interfaceId)
{
   clientPacket_t *pClientPacket = ((clientPacket_t*)pData);
   fsciLen_t myNewLen;
   switch( pClientPacket->structured.header.opCode )
   {
      case 0x01:
      {
         /* Reuse packet received over the serial interface The OpCode remains the same. The length of the␣
↪response must be <= that the length of the received packet */
         pClientPacket->structured.header.opGroup = myResponseOpGroup;/* Process packet */
         ...
         pClientPacket->structured.header. len = myNewLen;
         FSCI_transmitFormatedPacket(pClientPacket, interfaceId);
         return;
      }
      case 0x02:
      {
         /* Alocate a new message for the response. The received packet is Freed */
         clientPacket_t *pResponsePkt = MEM_BufferAlloc( sizeof(clientPacketHdr_t) + myPayloadSize_d␣
↪+ sizeof(uint8_t) // CRC);
         if(pResponsePkt)
         {
            /* Process received data and fill the response packet */ ...
            pResponsePkt->structured.header. len = myPayloadSize_d;
            FSCI_transmitFormatedPacket(pClientPacket, interfaceId);
         }
         break;
      }
      default:
         MEM_BufferFree( pData );
         FSCI_Error( gFsciUnknownOpcode_c, interfaceId );
         return;
   }
   /* Free message received over the serial interface */
   MEM_BufferFree( pData );
}
```

Helper Functions Library

Overview This framework provides a collection of features commonly used in embedded software centered on memory manipulation.

HWParameter: Hardware parameter

Production Data Storage Hardware parameters provide production data storage

Overview Different platforms/boards need board/network node-specific settings to function according to the design. (Examples of such settings are IEEE® addresses and radio calibration values specific to the node.) For this purpose, the last flash sector is reserved and contains hardware-specific parameters for production data storage. These parameters pertain to the network node as a distinct entity. For example, a silicon mounted on a PCB in a specific configuration, rather than to just the silicon itself. This sector is reserved by the linker file, through the PROD_DATA section and it should be read/written only through the API described below.

> Note : This sector is not erased/written at code download time and it is not updated via over-the-air firmware update procedures to preserve the respective node-specific data, regardless of the firmware running on it.

Constant Definitions Name :

```
extern uint32_t PROD_DATA_BASE_ADDR[];
```

Description :

This symbol is defined in the linker script. It specifies the start address of the PROD_DATA section.

Name :

```
static const uint8_t mProdDataIdentifier[10] = {"PROD_DATA:"};
```

Description :

The value of this constant is copied as identification word (header) at the beginning of the PROD_DATA area and verified by the dedicated read function.

Note: the length of mProdDataIdentifier imposes the definition of PROD_DATA_ID_STRING_SZ as 10. The legacy HW parameters structure provides headroom for future usage. There are currently 63 bytes available.

Data type definitions Name :

```
typedef PACKED_STRUCT HwParameters_tag
{
    uint8_t identificationWord[PROD_DATA_ID_STRING_SZ]; /* internal usage only: valid data present */
    /*@{*/
    uint8_t bluetooth_address[BLE_MAC_ADDR_SZ];       /*!< Bluetooth address */
    uint8_t ieee_802_15_4_address[IEEE_802_15_4_SZ]; /*!< IEEE 802.15.4 MAC address - K32W1 only␣
↪*/
    uint8_t xtalTrim;                                 /*!< XTAL 32MHz Trim value */
    uint8_t xtalCap32K;                                 /*!< XTAL 32kHz capacitance value */
    /* For forward compatibility additional fields may be added here
       Existing data in flash will not be compatible after modifying the hardwareParameters_t typedef.
       In this case the size of the padding has to be adjusted.
    */
    uint8_t reserved[1];
    /* first byte of padding : actual size if 63 for legacy HwParameters but
      complement to 128 bytes in the new structure */
}
hardwareParameters_t;
```

Description:

Defines the structure of the hardware-dependent information.

> Note : Some members of this structure may be ignored on a specific board/silicon configuration. Also, new members may be added for implementation-specific purposes and the backward compatibility must be maintained.

The CRC calculation starts from the reserved field of the hardwareParameters_t and ends before the hardwareParamsCrc field. Additional members to this structure may be added using the following method :

Add new fields before the reserved field. This method does not cause a CRC fail, but you must keep in mind to subtract the total size of the new fields from the size of the reserved field. For example, if a field of uint8_t size is added using this method, the size of the reserved field shall be changed to 63.

Co-locating application factory data in HW Parameters flash sector. The sector containing the Hardware parameter structure may be located in the internal flash, usually at its last sector. The actual Hardware parameter structure has a size of 128 bytes - including padding reserved for future use. Since there is plenty of room available in a flash sector (4kB or 8kB), co-locating Application Factory Data in the same structure prevents from reserving another flash sector for these data. The application designer may adopt this solution by defining gHwParamsAppFactoryDataExtension_d as 1. A total of 2kB is alloted to this purpose.

If this option was chosen, whenever any of the Hardware parameter fields is modified, its CRC16 will change so the sector will need erasing. The gHwParamsAppFactoryDataPreserveOnHwParamUpdate_d compilation option deals with restoring the contents of the App Factory Data. Nonetheless this requires a temporary allocation a 2kB buffer to preserve the previous content and restore then on completion of the Hw Parameter update.

Special reserved area at start of IFR1 in range [0x02002000..0x02002600[On development boards a 1536 byte area is reserved and the actual Hardware parameter area begins at offset 0x600. Preserving this area on a HW parameter update also requires a temporary 1.5kB dynamic allocation (in addition to the App Factory 2kB allocation), to be able to restore on completion of update operation.

HW Parameters Production Data placement options The placement of production data (PROD_DATA) can be selected based on the definition of gHwParamsProdDataPlacement_c (see fwk_config.h). The productions data seldom need update for final products, once calibration data, MAC addresses or others have been programmed. Two cases exist, plus a transition mode :

1) gHwParamsProdDataMainFlashMode_c (0) :
   - PROD_DATA are located at top of Main Flash. Hardware parameters section is placed in the last sector of internal flash [0xfe000..0x100000[.
   - The linker script must reserve this area explicitly so as to prevent placement of NVM or text sections at that location by setting gUseProdInfoMainFlash_d.
2) gHwParamsProdDataMainFlash2IfrMode_c(1) : - PROD_DATA are located in IFR1, but MainFlash version still exists during interim period. - If the contents of the PROD_DATA section in MainFlash is valid (not blank and correct CRC) but the IFR PROD_DATA is still blank, copy the contents of MainFlash PROD_DATA to IFR location. - When done PROD_DATA in IFR are used. Once the transition is done, an application using (2: gHwParamsProdDataPlacementIfrMode_c) may be programmed.
3) gHwParamsProdDataIfrMode_c (2) :
   - PROD_DATA section dwells in the IFR1 sector [0x02002000..0x02004000[
   - in development phase the area comprised between [0x02002000..0x02002600[must be reserved for internal purposes.
   - This allows to free up the top sector of Main Flash by linking with gUseProdInfoMainFlash_d unset.

LowPower

Low Power reference user guide This Readme file describes the connectivity software architecture and provides the general low power enablement user guide.

1- Connectivity Low Power SW architecture The connectivity low power software architecture is composed of various components. These are described from the lower layer to the application layer:

1. The SDK power manager in component/power_manager. This component provides the basic low power framework. It is not specific to the connectivity but generic across devices. it covers:
   - gather the low power constraints for upper layer and take the decision on the best suitable low power state the device is allowed to go to fullfill the constraints.
   - call the low power entry and exit function callbacks
   - call the appropriate SW routines to switch the device into the suitable low power state
2. Connectivity Low power module in the connectivity framework. This module is composed of:
   - The low power service called PWR inside framework/LowPower (this folder), This module is generic to all connectivity devices.
   - The platform lowpower: fwk_platform_lowpower.[ch] located in framework\platform\<platform_name>. These files are a collection of low power routines functions for the PWR module and upper layer. These are specific to the device.

     Both PWR and platform lowpower files are detailed in section below.
3. Low power Application modules, it consists of 3 parts:
   - Application initialization file app_services_init.c where the application initializes the low power framework, see next section 'Demo example for typical usage of low power framework'
   - Application Idle task from application to call the main low power entry function PWR_EnterLowPower() to switch the device into lowpower. This function is application specific, one example is given in the section 1.3.3
   - Low power board files : board_lp.[ch] located in board/lowpower. These files implement the low power entry and exit functions related to the application and board. Customers shall modify these files for their own needs. Example code is given for the connectivity applications.

     User guide is provided in section 1.3 below.

   Note : Linker script may also be impacted for power down mode support in order to provide an RAM area for ROM warm boot (depends on the platform) and application warmboot stack

The Low power central and master reference design applications provide an example of Low power implementation for BLE. Customer can also refer to the associated document 'low power connectivity reference design user guide'.

1.1 - SDK power manager This module provides the main low power functionalities such as:

- Decide the best low-power mode dependent on the constraints set by upper layers by using PWR_SetLowPowerModeConstraints() API function.

- Handle the sequences to enter and exit low-power mode.
- Enable and configure wake up sources, call the application callbacks on low power entry/exit sequences.

The SDK power manager provides the capability for application and all components to receive low power constraints to the power. The Application does not set the low-power mode the device shall go into. When going to low power, the SDK power manager selects the best low-power mode that fits all the constraints.

As an example, if the low power constraint set from Application is Power Down mode, and no other constraint is set, the SDK power manager selects Power down mode, the next time the device enters low power. However, if a new constraint is set by another component, such as the SecLib module that operates Hardware encryption, the SecLib module would select WFI as additional low power constraint. Also, the SDK power manager selects this last low-power mode until the constraint is released by the SecLib module. It then reselects Power Down mode for further low power entry modes.

1.2 - PWR Low power module The PWR module in the connectivity framework provides additional services for the connectivity stacks and applications on top of the SDK power manager.

It also provides a simple API for Connectivity Stack and Connectivity applications.

However, more advanced features such as configuring the wake-up sources are only accessible from the SDK Power Manager API.

In addition to the SDK Power Manager, the PWR module uses the software resources from lower level drivers but is independent of the platform used.

1.2.1 - Functional description Initialization of the PWR module should be done through PWR_Init() function. This is mainly to initialize the SDK power manager and the platform for low power. It also registers PWR low power entry/exit callback PWR_LowpowerCb() to the SDK power manager. This function will be called back when entering and exiting low power to perform mandatory save/restore operations for connectivity stacks. The application can perform extra optional save/restore operations in the board_lp file where it can register to the SDK Power Manager its own callback. This is usually used to handle optional peripherals such as serial interfaces, GPIOs, and so on.The main entry function is PWR_EnterLowPower(). It should be called from Idle task when no SW activity is required. The maximum duration for lowpower is given as argument timeoutUs in useconds. This function will check the next Hardware event in the connectivity stack, typically the next Radio activity. A wakeup timer is programmed if the timeoutUs value is shorter than the next radio event timing. Passing a timeout of 0us will be interpreted as no timeout on the application side.

On device wakeup from low power state, the function will return the time duration the device has been in low power state.

Two APi are provided to set and release low power state constraints : PWR_SetLowPowerModeConstraint() and PWR_ReleaseLowPowerModeConstraint(). These are helper functions. User can use directly the SDK power manager if needed.

The PWR module also provides some API to be set as callbacks into other components to prevent from going to low power state. It can be used in following examples :

1. If a DMA is running, the module in charge of the DMA would need to set a constraint to avoid the system from going to a low power state when the RAM and system bus are no longer available.
2. If transfer is going on a peripheral, the drivers shall set a constraint to forbid low power mode.
3. If encryption is on going through an Hardware accelerator, the HW accelerator and the required ressources (clocks, etc), shall be kept active also by setting a constraints.

1.2.2 - Tickless mode support This module also provides some routines functions PWR_SysticksPreProcess() and PWR_SysticksPostProcess() from PWR_systicks.c in order to support the tickless mode when using FreeRTOS. The tickless mode is the capability to suspend the periodic system ticks from FreeRTOS and keep timebase tracking using another low power counter. In this implementation, the Timer Manager and time_stamp component are used for this purpose.

Idle task shall call these functions PWR_SysticksPreProcess() and PWR_SysticksPostProcess() before and after the call to the main low power entry function PWR_EnterLowPower().

Refer to framework/LowPower/PWR_systicks.c file or section 2.1 below for more information.

1.3 - Low power platform submodule Low power platform module file fwk_platform_lowpower.c provides the necessary helper functions to support low power device initialization, device entry, and exit routines. These are platform and device specific. Typically, the PWR module uses the low power platform submodule for all low power specific routines.

The low power platform submodule is documented in the Connectivity Framework Reference Manual document and in the Connectivity Framework API document.

1.4 - Low power board files Low power board files board_lp.[ch] are both application and board specific. Users should update this file to add new functions to include new used peripherals that require low power support. In the current SDK package, only Serial Manager over UART and button (IO toggle wake up source) are supported and demonstrated in the Bluetooth LE demo application.

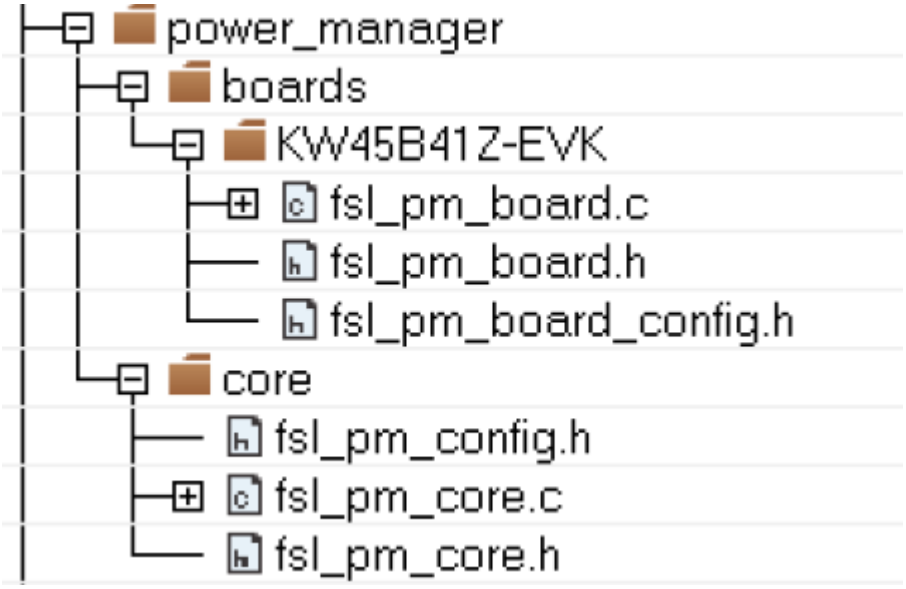
Other peripherals that require specific action on low power entry and restore on low power exit should be added to low power board files. For more details, refer to section Low power board file update

2 - Low power Application user guide This section provides a user guide to enable Low power on a connectivity application, It gives example of typical implementation for the initialization, Idle task function and low power entry/exit functions.

2.1 - Application Project updates It is recommended to reuse the low-power peripheral/central reference design application projects as a start. This ensures that everything is in place for the low-power optimization feature. Then, application files may be added to one of the two projects.

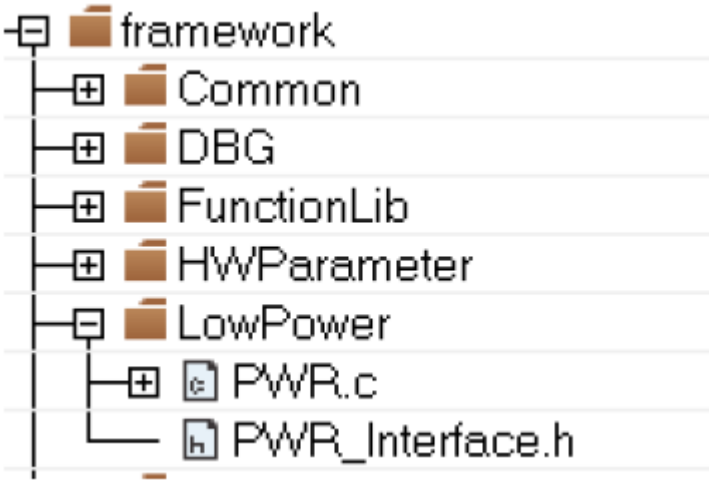
However, users can start directly from the application project and implement low power in it, by performing the steps described in the following sections.

2.1.1 - SDK Power Manager Most of the Low power functionality is implemented in the SDK Power Manager. The files to add into the project SDK power_manager module are listed in the figure below:



You need to use the files located in the folder that match your device.

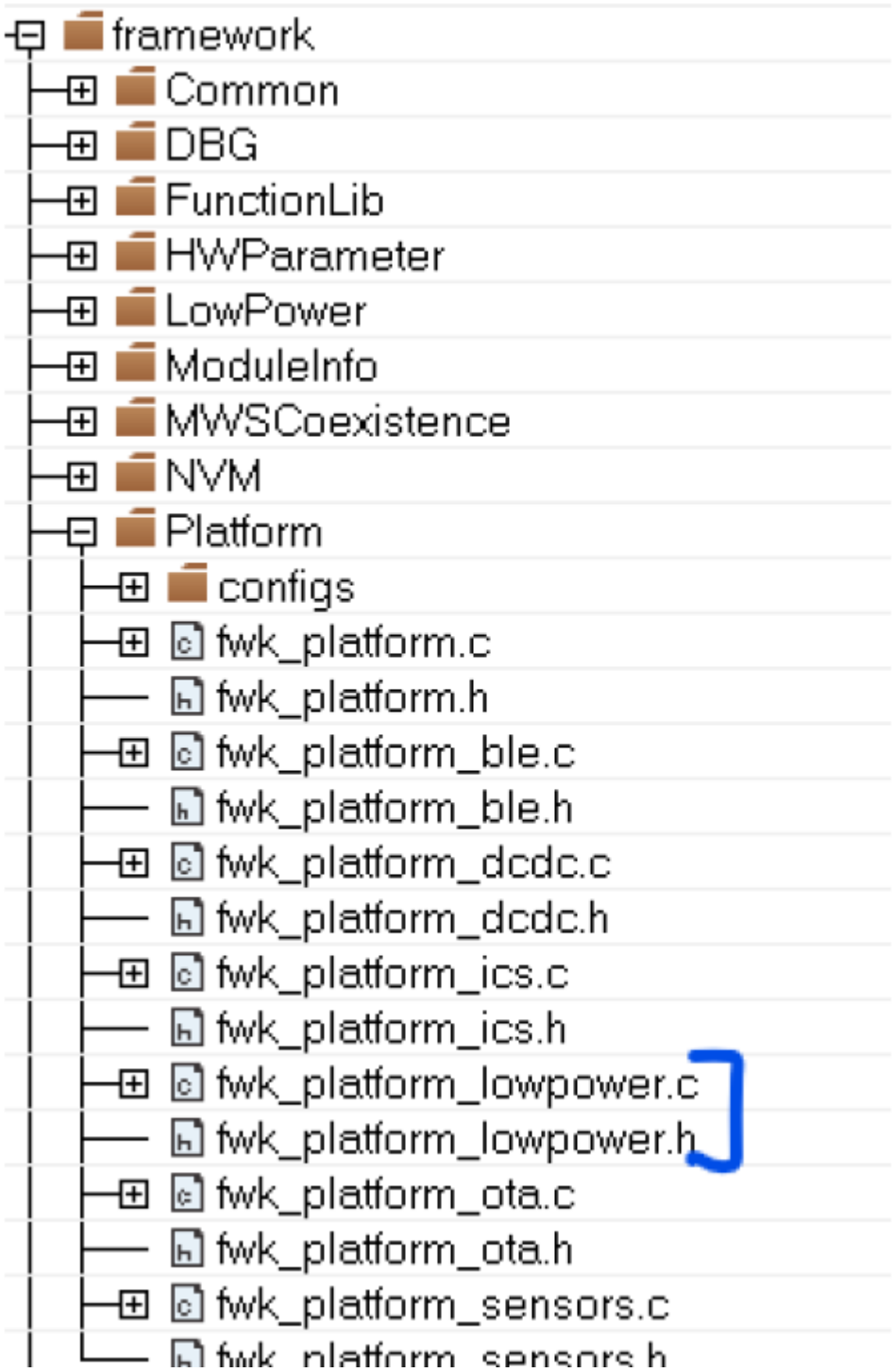
2.1.2 - PWR connectivity framework module PWR.c PWR_Interface.h shall be added to your application projects :



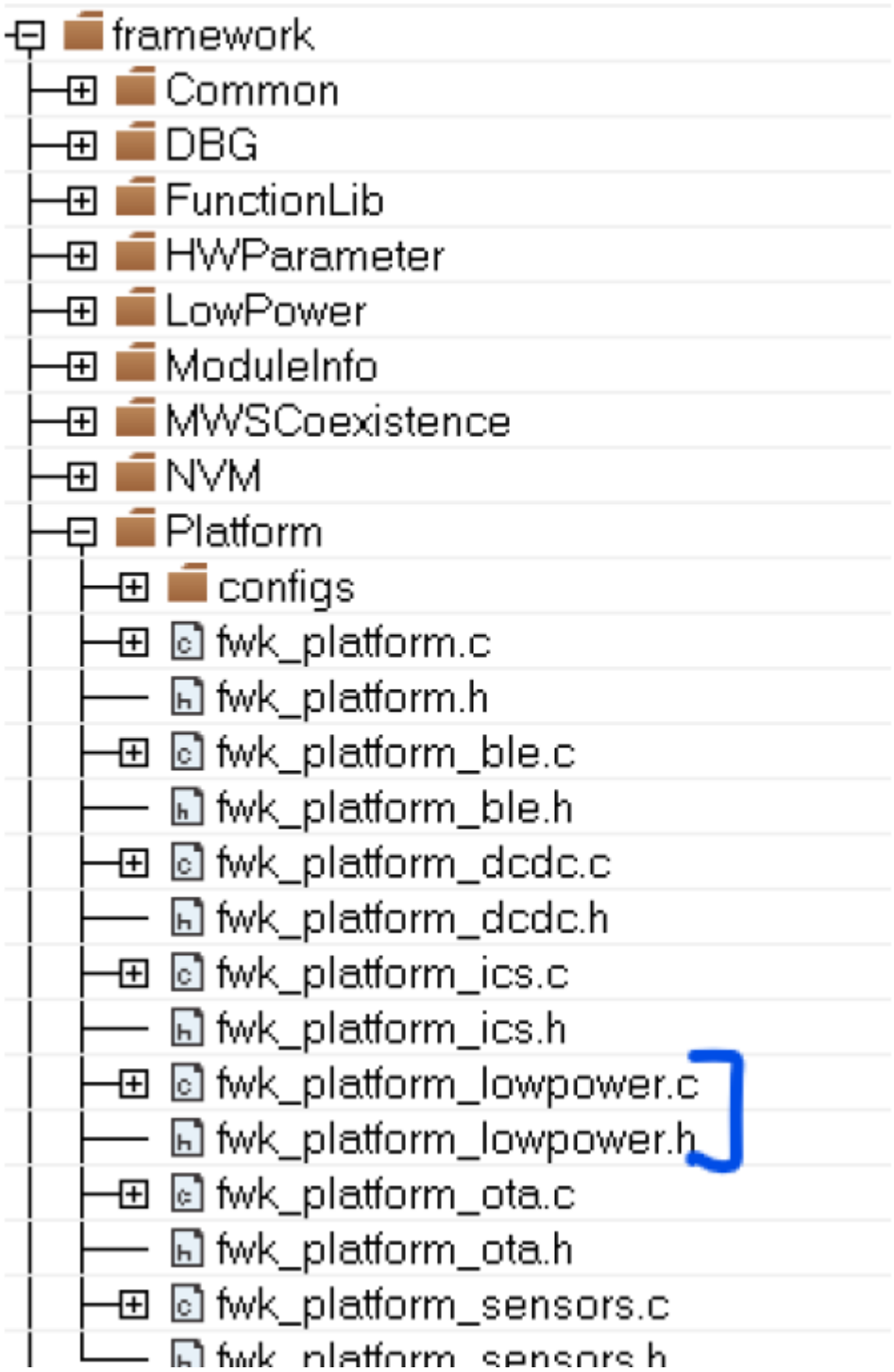
Optionally, in order to support Systick less mode, PWR_systicks.c or PWR_systicks_bm.c could also be added.

The include path to add is: middleware/wireless/framework/LowPower

2.1.3 -Low power platform submodule Low power platform files can be found in the 'Platform' module in the connectivity framework:



2.1.4 - Low power board files These files are located in the same folder that the other board files board.[ch]. Hence, it is not required to add any new include path at compiler command line.



2.1.5 - Application RTOS Idle hook and tickeless hook functions See section 2.4.3 Idle task implementation example

2.2 - Low power and wake up sources Initialization Low power initialization and configuration are performed in APP_ServiceInitLowpower()function. This is called from APP_InitServices() function called from the main() function so all is already set up when calling the main application entry point, typically BluetoothLEHost_AppInit() function in the Bluetooth LE demo applications.

The default Low Power mode configured in APP_InitServices() is Deep Sleep mode. In Bluetooth

LE, (or any other stack technology), Deep Sleep mode fits for all use cases. For instance, for Bluetooth LE states: Advertising, Connected, Scanning states. This mode already performs a very good level of power saving and likely, this is not required to optimize more if the device is powered from external supply.

APP_ServiceInitLowpower() function performs the following initialization and configuration:

- Initialize the Connectivity framework Low power module PWR_Init(), this function initialized the SDK power manager.
- Configure the wakeup sources such as serial manager wake up source for UART, or button for IO wake up configuration. These are typical wakeup sources used in the connectivity application. Developer may want to add additional wake up sources here specific for the application.

  Note : The low power timer wakeup source and wakeup from Radio domain are directly enabled from the Connectivity framework Low power module PWR as it is mandatory for the connectivity stack. If your application supports other peripherals (such as i2c, spi, and others) that require wake sources from low power, developer should add additional wake up sources setting in this function APP_ServiceInitLowpower(). The complete list of wakeup sources are available from the SDK power manager component, see file fsl_pm_board.h in component/boards/<device_name>/.
- Initialize and register the Low power board file used to register and implement low power entry and exit callback function used for peripheral. This is done by calling the BOARD_LowPowerInit() function.
- Register low power Enter and exit critical function to driver component to enable / disable low power when the Hardware is active. Example is given for serial manager that needs to disable low power when the TX ring buffer contains data so the device does not enter low power until the buffer is empty.

Finally, APP_ServiceInitLowpower() function configures the Deep Sleep mode as the default low power constraint for the application. It is recommended to keep this level of low power constraint during all the connectivity stack initialization.

Example of low power framework initialization can be found in app_services_init.c file. Below is some code example for initializing the low power framework and wake up sources:

```
static void APP_ServiceInitLowpower(void)
{
    PWR_ReturnStatus_t status = PWR_Success;

    /* It is required to initialize PWR module so the application
     * can call PWR API during its init (wake up sources...) */
    PWR_Init();

    /* Initialize board_lp module, likely to register the enter/exit
     * low power callback to Power Manager */
    BOARD_LowPowerInit();

    /* Set Deep Sleep constraint by default (works for All application)
     *   Application will be allowed to release the Deep Sleep constraint
     *   and set a deepest lowpower mode constraint such as Power down if it needs
     *   more optimization */
    status = PWR_SetLowPowerModeConstraint(PWR_DeepSleep);
    assert(status == PWR_Success);

#if (defined(gAppButtonCnt_c) && (gAppButtonCnt_c > 0))

    /* Init and enable button0 as wake up source
     * BOARD_WAKEUP_SOURCE_BUTTON0 can be customized based on board configuration
```

(continues on next page)

(continued from previous page)

```
    * On EVK we use the SW2 mapped to GPIOD */
   PM_InitWakeupSource(&button0WakeUpSource, BOARD_WAKEUP_SOURCE_BUTTON0, NULL,␣
↪true);
#endif

#if (gAppButtonCnt_c > 1)
   /* Init and enable button1 as wake up source
    * BOARD_WAKEUP_SOURCE_BUTTON1 can be customized based on board configuration
    * On EVK we use the SW3 mapped to PTC6 */
   PM_InitWakeupSource(&button1WakeUpSource, BOARD_WAKEUP_SOURCE_BUTTON1, NULL,␣
↪true);
#endif

#if (defined(gAppUseSerialManager_c) && (gAppUseSerialManager_c > 0))

#if defined(gAppLpuart0WakeUpSourceEnable_d) && (gAppLpuart0WakeUpSourceEnable_d > 0)
   /* To be able to wake up from LPUART0, we need to keep the FRO6M running
    * also, we need to keep the WAKE domain is SLEEP.
    * We can't put the WAKE domain in DEEP SLEEP because the LPUART0 is not mapped
    * to the WUU as wake up source */
   (void)PM_SetConstraints(PM_LP_STATE_NO_CONSTRAINT, APP_LPUART0_WAKEUP_
↪CONSTRAINTS);
#endif

   /* Register PWR functions into SerialManager module in order to disable device lowpower
      during SerialManager processing. Typically, allow only WFI instruction when
      uart data are processed by serail manager  */
   SerialManager_SetLowpowerCriticalCb(&gSerMgr_LowpowerCriticalCBs);
#endif

#if defined(gAppUseSensors_d) && (gAppUseSensors_d > 0)
   Sensors_SetLowpowerCriticalCb(&app_LowpowerSensorsCriticalCBs);
#endif

   (void)status;
}
```

2.3 - low power entry/exit sequences : board files updates Board Files that handles low-power are board_lp.[ch] files.

Low power board files implement the low-power callbacks of the peripherals to be notified when entering or exiting Low Power mode. This module also registers these low-power callbacks to the SDK Power Manager component to get the notifications when the device is about to enter low-power or exit Low Power mode. The Low-power callbacks are registered from BOARD_LowPowerInit() function. This function is called from app_services_init.c file after PWR module initialization.

The low power callback functions can be categorized in two groups:

- Entry Low power call back functions: These are usually used to prepare the peripherals to enter low-power. For example, they can be used for flushing FIFOs, switching off some clocks, and reconfiguring pin mux to avoid leakage on pins. In case of Power Down mode, these functions could be used to save the Hardware peripheral context.
- Exit Low power call back functions: These are typically used to restore the peripherals to functionality. Therefore, they perform the reverse of what is done by the entry call-back functions: restoring the pin mux, re-enabling the clock, in case of Power Down mode, restoring the Hardware peripheral context, and so on.

Note that distinction can be done between clock gating mode (Deep Sleep mode), and power gated mode (Power down mode) when entering and exiting Low Power mode. The

BOARD_EnterLowPowerCb() and BOARD_ExitLowPowerCb() functions provide the code to call the various peripheral entry and exit functions to go and exit Deep Sleep mode: serial manager, button, debug console, and others.

However, the processing to save and restore the Hardware peripheral is implemented in different functions BOARD_EnterPowerDownCb() and BOARD_ExitPowerDownCb(). These two functions should be called when exiting power gated modes of the power domain. These two should implement specific code for such case (likely the complete reinitialization of each peripheral). In order to know the Low Power mode that the wake up domain, or main domain has been entered, the low-power platform API PLATFORM_GetLowpowerMode() can be called.

> **Note :** BOARD_ExitPowerDownCb() is called before BOARD_ExitLowPowerCb() as it is generally required to restore the Hardware peripheral contexts before reconfiguring the pin mux to avoid any signal glitches on the pads

Also, It is important to know whether the location of the Hardware peripheral is in the main domain or wake up domain. The two power domains can go into different power modes with the limitation that the wakeup domain cannot go to a deepest Low Power mode than the main domain. Depending on the constraint set on SDK power manager, the wake up domain could remain in active while the main domain can go to deep sleep or power down modes. In this case, the peripherals in the wake up domain does not required to be restored, as explained in the section Power Down. Likely, only pin mux reconfiguration is required in this case.

example Low power entry and exit functions shall be registered to the SDK power manager so these functions will be called when the device will enter and exit low power mode. This is done by BOARD_LowPowerInit() typically called from application source code in app_services_init.c file

```
static pm_notify_element_t boardLpNotifyGroup = {
    .notifyCallback = BOARD_LowpowerCb,
    .data           = NULL,
};

void BOARD_LowPowerInit(void)
{
    status_t status;

    status = PM_RegisterNotify(kPM_NotifyGroup2, &boardLpNotifyGroup);
    assert(status == kStatus_Success);
    (void)status;
}
```

BOARD_LowpowerCb() callback function will handle both the entry and exit sequences. An argument is passed to the function to indicate the lowpower state the device enter/exit. Typical implementation is given below. Customer shall make sure to differentiate low power entry and exit, and the various low power states.

Typically, nothing is expected to be done if low power state is WFI or Sleep mode. These modes are some light low power states and the system can be woken up by interrupt trigger.

In Deep sleep mode, the clock tree and source clocks are off, the system needs to be woken up from an event from the WUU module.

In Power down mode, some peripherals are likely to be powered off, context save and restore may need to be done in these functions.

```
static status_t BOARD_LowpowerCb(pm_event_type_t eventType, uint8_t powerState, void *data)
{
    status_t ret = kStatus_Success;
    if (powerState < PLATFORM_DEEP_SLEEP_STATE)
    {
        /* Nothing to do when entering WFI or Sleep low power state
           NVIC fully functionnal to trigger upcoming interrupts */
```

(continues on next page)

(continued from previous page)

```
    }
    else
    {
        if (eventType == kPM_EventEnteringSleep)
        {
            BOARD_EnterLowPowerCb();

            if (powerState >= PLATFORM_POWER_DOWN_STATE)
            {
                /* Power gated low power modes often require extra specific
                 * entry/exit low power procedures, those should be implemented
                 * in the following BOARD API */
                BOARD_EnterPowerDownCb();
            }
        }
        else
        {
            /* Check if Main power domain domain really went to Power down,
             *   powerState variable is just an indication, Lowpower mode could have been skipped by an␣
↪immediate wakeup
             */
            PLATFORM_PowerDomainState_t main_pd_state = PLATFORM_NO_LOWPOWER;
            PLATFORM_status_t           status;

            status = PLATFORM_GetLowpowerMode(PLATFORM_MainDomain, &main_pd_state);
            assert(status == PLATFORM_Successful);
            (void)status;

            if (main_pd_state == PLATFORM_POWER_DOWN_MODE)
            {
                /* Process wake up from power down mode on Main domain
                 *  Note that Wake up domain has not been in power down mode */
                BOARD_ExitPowerDownCb();
            }

            BOARD_ExitLowPowerCb();
        }
    }
    return ret;
}
```

2.4 - Low power constraint updates and optimization Except for the board file update as seen in previous section, the application does not need any other changes for low-power support in Deep Sleep mode. It shall work as if no low-power is supported. However, If more aggressive power saving is required, this constraint can be changed in your application in order to further reduce the power consumption in Low Power mode.

2.4.1 - Changing the Default Application low power constraint after firmware initialization The Low power reference design applications (central or peripheral) provides demonstration on how to change the Application low power constraint. In the Application main entry point BluetoothLEHost_AppInit(), Deep Sleep mode is configured by default from APP_ServiceInitLowpower() function.

> **Note :** It is recommended to keep Deep Sleep mode as default during all the stack initialization phase until BluetoothLEHost_Initialized() and BleApp_StartInit() functions are called. In case of Bonded device with privacy, it is recommended to wait for gControllerPrivacyStateChanged_c event to be called.

BleApp_LowpowerInit() function provides an example of code on how to release the default Deep sleep low-power constraint and set a new constraint such as Power down mode for the application. This deeper low-power mode is used when no Bluetooth LE activity is on going, and if no other higher Low-power constraint is set by another components or layer. For instance, if some serial transmission is on going by the serial manager, or if the SecLib module has on going activity on the HW crypto accelerator, the low-power mode could less deep.

```
static void BleApp_LowpowerInit(void)
{
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
    PWR_ReturnStatus_t status;

    /*
     * Optionally, Allow now Deepest lowpower mode constraint given by gAPP_
↪LowPowerConstraintInNoBleActivity_c
     *    rather than DeepSleep mode.
     * Deep Sleep mode constraint has been set in APP_InitServices(), this is fine
     *    to keep this constraint for typical lowpower application but we want the
     *    lowpower reference design application to be more agressive in term of power saving.

     *   To apply a lower lowpower mode than Deep Sleep mode, we need to
     *    - 1) First, release the Deep sleep mode constraint previously set  by default in app_services_init()
     *    - 2) Apply new lowpower constraint when No BLE activity
     *   In the various BLE states (advertising, scanning, connected mode), a new Lowpower
     *    mode constraint will be applied depending of Application Compilation macro set in app_preinclude.
↪h :
     *    gAppPowerDownInAdvertising, gAppPowerDownInConnected, gAppPowerDownInScanning
     */

    /*  1) Release the Deep sleep mode constraint previously set  by default in app_services_init() */
    status = PWR_ReleaseLowPowerModeConstraint(PWR_DeepSleep);
    assert(status == PWR_Success);
    (void)status;

    /*  2) Apply new Lowpower mode constraint gAppLowPowerConstraintInNoBleActivity_c *
     *      The BleAppStart() call above has already set up the new lowpower constraint
     *       when Advertising request has been sent to controller        */
    BleApp_SetLowPowerModeConstraint(gAppLowPowerConstraintInNoBleActivity_c);
#endif
}
```

2.4.2 - Changing the Application lowest low power constraint during application execution

In the various application use cases, (in the various Bluetooth LE activity states, advertising, connected, scanning), some lower low-power constraint can be set, as Power down for advertising, Deep Sleep for connected, or Scanning. Customer can change the level of Low Power mode in the various use case mainly depending of the time duration the device is supposed to remain in low-power. The longer the time that the device remains in low power, the higher the benefit for a deeper Low Power mode such as Power down mode. However, please note that the wake up from power down mode takes significantly more time than deep sleep as ROM code is re executed and the hardware logic needs to be restored. Sections Deep Sleep and Power Down provide some guidance on when to use Deep Sleep mode or Power Down modes respectively.

In the low power reference design applications, four application compilations macros are defined to adjust the low-power mode into advertising, scanning, connected, or no Bluetooth LE activity. Other use cases can be added as desired. For instance, If application needs to run a DMA transfer, or if application needs to wakeup regularly to process data from external device, it may be useful to set WFI constraint (in case of DMA transfer), or Deep Sleep constraint (in case of regular wake up to process external data), rather than power down or a even lower low-power mode.

The 4 application compilation macros can be found in app_preinclude.h file of the project. See

app_preinclude.h for low power reference design peripheral application :

```
/*! Lowpower Constraint setting for various BLE states (Advertising, Scanning, connected mode)
   The value shall map with the type defintion PWR_LowpowerMode_t in PWR_Interface.h
     0 : no LowPower, WFI only
     1 : Reserved
     2 : Deep Sleep
     3 : Power Down
     4 : Deep Power Down
   Note that if a Ble State is configured to Power Down mode, please make sure
     gLowpowerPowerDownEnable_d variable is set to 1 in Linker Script
   The PowerDown mode will allow lowest power consumption but the wakeup time is longer
     and the first 16K in SRAM is reserved to ROM code (this section will be corrupted on
     each power down wakeup so only temporary data could be stored there.)
         Power down feature not supported. */

#define gAppLowPowerConstraintInAdvertising_c          3
/* Scanning not supported on peripheral */
//#define gAppLowPowerConstraintInScanning_c            2
#define gAppLowPowerConstraintInConnected_c           2
#define gAppLowPowerConstraintInNoBleActivity_c       4
```

In lowpower_central.c lowpower_preripheral.c files, the application sets and releases the low power constraint from BleApp_SetLowPowerModeConstraint() and BleApp_ReleaseLowPowerModeConstraint() functions. These functions are called with the macro value passed as argument.

> **Important Note :** Setting the application low power constraint shall be done on new Bluetooth LE state request so the new constraint is applied immediately, while the application low-power mode constraint shall be released when the Bluetooth LE state is exited. For example, setting the new low power constraint for Advertising shall be done when the application requests advertising to start. Releasing the low power constraint shall be done in the advertising stop callback (advertising has been stopped).

After releasing the low power constraint, the previous low power constraint, (likely the one that has been set during firmware initialization in APP_ServiceInitLowpower() function, or the updated low power constraint in BleApp_StartInit() function) applies again.

2.4.3 - Idle task implementation example

2.4.3.1 Tickless mode support and Low power entry function Idle task configuration from FreeRTOS shall be enabled by configUSE_TICKLESS_IDLE in FreeRTOSConfig.h. This will have the effect to have vPortSuppressTicksAndSleep() called from Idle task created by FreeRTOS. Here is a typical implementation of this function:

```
void vPortSuppressTicksAndSleep(TickType_t xExpectedIdleTime)
{

    bool abortIdle = false;
    uint64_t actualIdleTimeUs, expectedIdleTimeUs;

    /* The OSA_InterruptDisable() API will prevent us to wakeup so we use
     * OSA_DisableIRQGlobal() */
    OSA_DisableIRQGlobal();

    /* Disable and prepare systicks for low power */
    abortIdle = PWR_SysticksPreProcess((uint32_t)xExpectedIdleTime, &expectedIdleTimeUs);

    if (abortIdle == false)
    {
```

(continues on next page)

(continued from previous page)

```
        /* Enter low power with a maximal timeout */
        actualIdleTimeUs = PWR_EnterLowPower(expectedIdleTimeUs);

        /* Re enable systicks and compensate systick timebase */
        PWR_SysticksPostProcess(expectedIdleTimeUs, actualIdleTimeUs);
    }

    /* Exit from critical section */
    OSA_EnableIRQGlobal();
}
```

2.4.3.2 Connectivity background tasks and Idle hook function example

Some process needs to be run in background before going into low power. This is the case for writing in NVM, or firmware update OTA to be writen in Flash. If so, configUSE_IDLE_HOOK shall be enabled in FreeRTOSCOnfig.h so vApplicationIdleHook() will be called prior to vPortSuppressTicksAndSleep(). Typical implementation of vApplicationIdleHook() function can be found here :

```
void vApplicationIdleHook(void)
{
    /* call some background tasks required by connectivity */
#if ((gAppUseNvm_d) || \
    (defined gAppOtaASyncFlashTransactions_c && (gAppOtaASyncFlashTransactions_c > 0)))

    if (PLATFORM_CheckNextBleConnectivityActivity() == true)
    {
        BluetoothLEHost_ProcessIdleTask();
    }
#endif
}
```

PLATFORM_CheckNextBleConnectivityActivity() function implemented in low power platform file fwk_platform_lowpower.c typically checks the next connectivity event and returns true if there's enough time to perform time consuming tasks such as flash erase/write operations (can be defined by the compile macro depending on the platform).

2. Low power features

2.1 - FreeRTOS systicks

Low power module in framework supports the systick generation for FreeRTOS. Systicks in FreeRTOS are most of the time not required in the Bluetooth LE demos applications because the framework already supports timers by the timer manager component, so the application can use the timers from this module. The systicks in FreeRTOS are useful for all internal timer service provided by FreeRTOS (through OSA) like OSA_TimeDelay(), OSA_TimeGetMsec(), OSA_EventWait(). When systicks are enabled, an interrupt (systick interrupt) is triggered and executed on a periodic basis. In order to save power, periodic systick interrupts are undesirable and thus disabled when going to low-power mode. This feature is called low power FreeRTOS tickless mode. When entering the low power state, the system ticks shall be disabled and switch to a low power timer. On wake-up, the module retrieves the time passed in low power and compensate the ticks count accordingly. This feature does not apply on bare metal scheduler.

On FreeRTOS, the vPortSuppressTicksAndSleep() function implemented in the app_low_power.c file will be called when going to idle. FreeRTOS will give to this function the xExpectedIdleTime, time in tick periods before a task is due to be moved into the Ready state. This function will manage the systicks (disable/enable) through PWR_SysticksPreProcess() and PWR_SysticksPostProcess() calls. Then, when calling PWR_EnterLowPower(), a time out duration in micro seconds will be given and the function will set a timer before entering low power.

In addition, this function will return the low power period duration, used to compensate the ticks count.

In our example low power reference design peripheral application, an OSA_EventWait() has been added to demonstrate the tickless mode feature. You can adjust the timeout with the gAppTaskWaitTimeout_ms_c flag in the app_preinclude.h file, its value in our demo is 8000ms. So 8 seconds after stopping any activity we will wake up from low power. If the flag is not defined in the application its value will be osaWaitForever_c and there will be no OS wake up.

2.2 - Selective RAM bank retention To optimize the consumption in low power, the linker script specific function PLATFORM_GetDefaultRamBanksRetained() is implemented. This function obtains the RAM banks that need to be retained when the device goes in low power, in order to set them with PLATFORM_SetRamBanksRetained() function. The RAM banks that are not needed are set in power off state, when the device goes in low power mode.

The function PLATFORM_GetDefaultRamBanksRetained() is linker script specific. Hence, it cannot be adapted for a different application. If these functions are called from board_lp.c, it is possible to give to PLATFORM_SetRamBanksRetained() a different bank_mask adapted to your specific application.

In deep power down, this feature does not have any impact because in this power mode, all RAM banks are already powered off.

3 - Low power modes overview PWR module API provides the capability to set low power mode constraints from various components or from the application. These constraints are provided to the SDK power manager. Upper layer (all Application code, connectivity stacks, etc.) can call directly the SDK Power Manger if it requires more advanced tuning. The PWR API can be found in PWR_Interface.h.

> **Note :** 'Upper layer' signifies all layers, applications, components, or modules that are above the connectivity framework in the Software architecture.
>
> **Note :** Each power domain has its own Low Power mode capability. The Low Power modes described below are for the main domain and it is supposed that the wake up domain goes to the same Low Power mode. This is not always true as the wake up domain that contains some wake up peripheral can go a lower Low Power mode state than the main domain so the peripherals in the wake up domain can remain operational when the main domain is in Low Power mode (deep sleep or power down modes). In this case, the context of the Hardware peripheral located in the wake up domain does not need to be saved and restored as for the peripherals located in the main domain

3.1 Wait for Interrupt (WFI) Definition

In the Wait for Interrupt (WFI) state, the CPU core is powered on, but is in an idle mode with the clock turned OFF.

Wake up time and typical use case

The wakeup time from this Low Power mode is insignificant because the Fast clock from FRO is still running.

This Low Power mode is mainly used when there is an hardware activity while the Software runs the Idle task. This allows the code execution to be temporarily suspende, thus reducing a bit the power consumption of the device by switching off the processor clock. When an interrupt fires, the processor clock is instantaneously restored to process the Interrupt Service Routine (ISR).

Usage

In order to prevent the software from programming the device to go to a lower Low Power mode (such as Deep Sleep, Power Down mode or Deep Power Down mode), the component responsible for the hardware drivers shall call PWR_SetLowPowerModeConstraint(PWR_WFI) function. When the Hardware activity is completed, the component shall release the constraint by calling PWR_ReleaseLowPowerModeConstraint(PWR_WFI).

Alternatively, the component can call PWR_LowPowerEnterCritical() and then PWR_LowPowerExitCritical() functions.

For fine tuning of the Low Power mode allowing more power saving, the component can call directly the SDK power manager API with PM_SetConstraints() function using the appropriate Low Power mode and low power constraint. However, this is reserved for more advanced user that knows the device very well. It is not recommended to do so.

The PWR module has no external dependencies, so the low-power entry and exit callback functions must be defined by the user for each peripheral that has specific low power constraints It is consequently convenient to register to the component the low power callbacks structure that is used for entering and exit low power critical sections. In Bluetooth LE, you can take the example in the app_conn.c file as shown here :

```
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
static const Seclib_LowpowerCriticalCBs_t app_LowpowerCriticalCBs =
{
  .SeclibEnterLowpowerCriticalFunc = &PWR_LowPowerEnterCritical,
  .SeclibExitLowpowerCriticalFunc  = &PWR_LowPowerExitCritical,
};
#endif

void BluetoothLEHost_Init(..)
{
  ...
      /* Cryptographic hardware initialization */
      SecLib_Init();
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
      /* Register PWR functions into SecLib module in order to disable device lowpower
         during Seclib processing. Typically, allow only WFI instruction when
         commands (key generation, encryption) are processed by Seclib  */
      SecLib_SetLowpowerCriticalCb(&app_LowpowerCriticalCBs);
#endif
  ...
}
```

Limitations

No limitation when using the WFI mode.

3.2 Sleep mode Sleep mode is similar to WFI low power mode but with some additional clock gating. The Sleep mode is device specific, please consult the Hardware reference manuel of the device for more information.

3.2 Deep Sleep mode Definition

In Deep Sleep mode, the fast clock is turned off, and the CPU along with the main power domain are placed into a retention state, with the voltage being scaled down to support state retention only. Because no high frequency clock is running, the voltage applied on the power domain can be reduced to reduce leakage on the hardware logic. This reduces the overall power consumption in the Deep Sleep mode. When waking up from Deep sleep mode, the core voltage is increased back to nominal voltage and the fast clock (FRO) is turned back on, the peripheral in this domain can be reused as normal.

To same more additional power, Some unused RAM banks can be powered off. this prevents from having current leakage and consequently, allow to reduce even more the power consumption in Deep SLeep mode. This is achieved by calling PLATFORM_SetRamBanksRetained() from low power entry function from board_lp.c file.

Usage

All firmware is able to implement Deep Sleep mode transparently to the application thanks to the PWR module, low power platform submodule and low power board file. This is described in the section Low-power implementation.

When entering this mode, it is recommended to turn the output pins into input mode, or high impedance to reduce leakage on the pads. This is typically done in pin_mux.c file, called from board.c file and executed from the low power callback in board_lp.c file. As an example, the TX line of the UART peripheral can be turned to disabled so it prevents the current from being drawn by the pad in Low Power mode.

Wake up time and typical use case

The wake up time is very fast, it takes mostly the time for the Fast FRO to start up again (couple of hundreds of microseconds) so this mode is a very good balance between power consumption in low-power mode and wake up latency and shall be used extensively in most of the use cases of the application.

Limitations

In Deep Sleep mode, the clock is disabled to the CPU and the main peripheral domain, so peripheral activity (for example, an on-going DMA transfer) is not possible in Deep Sleep mode.

3.3 Power Down mode **Definition**

In Power Down mode, both the clock, and power are shut off to the CPU and the main peripheral domain. SRAM is retained, but register values are lost. The SDK power manager handles the restore of the processor registers and dependencies such as interrupt controller and similar ones transparently from the application.

Usage

The application, with the help of the low power board files, saves and restores the peripherals that were located in the power domain during the entry and exit of the power down mode. This is done from low power board_lp files in the entry/exit low power callbacks. Example is given for the serial manager and debug console in *board_lp.c* file in function *BOARD_ExitPowerDownCb()*.

If the device contains a dedicated wake up power domain where some wake up peripherals are located, if this wake up domain is not turned into power down mode but only Deep sleep mode or active mode, this peripheral does not need for a save and restore on low power entry/exit. For instance, on KW45, This is basically achieved when enabling the wakeup source of the peripheral *PWR_EnableWakeUpSource()* from *APP_ServiceInitLowpower()* function. Alternatively, this can be directly achieved by setting the constraint to the SDK power manager by calling *PM_SetConstraints()*, (use APP_LPUART0_WAKEUP_CONSTRAINTS for wakeup from UART constraint).

On exit from low power, The low power state of power domain can be retrieved by Platform API PLATFORM_GetLowpowerMode(). This API shall be called from low power exit callback function only.

As for Deep Sleep mode, software shall configure the output pins into input or high impedance during the Low Power mode to avoid leakage on the pads.

Wake up time and typical use case

The wake up time is significantly longer than wake up time from Deep Sleep (from several hundreds of micro-seconds to a couple of milliseconds depending on the platform). On some platform, it can takes longer, for instance, if ROM code is implemented and perform authentication checks for security and hardware logic in power domain needs to be restored (case for KW45).

However, After ROM code execution, the SDK power manager resumes the Idle task execution from where it left before entering low-power mode. Hence, the wakeup time from this mode is still significantly lower that the initialization time from a power on reset or any other reset.

Depending on the wakeup time of the platform and the low power time duration, This mode is recommended when no Software activity is expected to happen for the next several seconds. In Bluetooth LE, this mode is preferred in advertising or without Bluetooth LE activity. However, in scanning or connected mode, Regular wakes up happens regularly for instance to retrieve HCI message responses from the Link layer, the Deep Sleep mode is rather recommended.

Limitations

In addition to the Deep Sleep limitation (no Hardware processing on going when going to Power down mode) and the significant increase of the wake time, the Power Down mode requires the ROM code to execute and this last uses significant amount of memory in SRAM.

Typically, The first SRAM bank (16 KBytes) is used by the ROM code during execution so the Application firmware can use this section of SRAM for storing bss, rw data, or stacks. Only temporary data could be stored here and this location is overwritten on every Power Down exit sequence.

In order to avoid placing firmware data section (bss, rw, etc.) in the first SRAM bank, the linker script variable *gLowpowerPowerDownEnable_d* should be set to 1. Setting the linker script variable to avoid placing firmware data section in the first SRAM bank, The effect of setting this flag is to prevent the firmware from using the first 16 KB in SRAM.

> *Note* : This setting is ONLY required if the application implements Power Down mode.
> If Application uses other low-power mode, this is not required.

3.4 Deep Power-down mode **Definition**

In Deep Power Down mode, the SRAM is not retained. This power mode is the lowest disponible, it is exited through reset sequence.

Usage

In addition to the Power Down limitation, the Deep Power Down mode shut down all memory in SRAM. Because it is exited through reset sequence the wake time is also longer.

Wake up time and typical use case

As this low-power mode is exited through the reset sequence, the wake up time is longer than any other mode. In Bluetooth LE, this mode is possible in no Bluetooth LE activity, and is preferred if we know that there will be no Bluetooth LE activity before a several amount of time.

Limitations

All memory in SRAM will be shut down in deep power down, the main limitation in going in this low-power mode is that the context will not be saved.

ModuleInfo

Overview The ModuleInfo is a small Connectivity Framework module that provides a mechanism that allows stack components to register information about themselves.

The information comprises :

- Component or module name (for example: Bootloader, IEEE 802.15.4 MAC, and Bluetooth LE Host) and associated version string
- Component or module ID
- Version number
- Build number

The information can be retrieved using shell commands or FSCI commands.

Detailed data types and APIs used in ConnFWK_APIs_documentation.pdf.

NVM: Non-volatile memory module

Overview In a standard Harvard-architecture-based MCU, the flash memory is used to store the program code and program constant data. Modern processors have a built-in flash memory controller that can be used under user program execution to store non-volatile data. The flash memories have individually erasable segments (sectors) and each segment has a limited number of erase cycles. If the same segments are used to store various kinds of data all the time, those segments quickly become unreliable. Therefore, a wear-leveling mechanism is necessary to prolong the service life of the memory. The NVM module in the connectivity framework provides a file system with a wear-leveling mechanism, described in the subsequent sections. The *NvIdle()* function handles the program and erase memory operations. Before resetting the MCU, *NvShutdown()* must be called to ensure that all save operations have been processed.

NVM boundaries and linker script requirement Most of the MCUs have only a standard flash memory that the non-volatile (NV) storage system uses. The amount of memory that the NV system uses for permanent storage and its boundaries are defined in the linker configuration file though the following linker symbols :

- NV_STORAGE_START_ADDRESS
- NV_STORAGE_END_ADDRESS
- NV_STORAGE_MAX_SECTORS
- NV_STORAGE_SECTOR_SIZE

The reserved memory consists of two virtual pages. The virtual pages are equally sized and each page is using one or more physical flash sectors. Therefore, the smallest configuration is using two physical sectors, one sector per virtual page.

NVM Table The Flash Management and Non-Volatile Storage Module holds a pointer to a RAM table. The upper layers of this table register information about data that the storage system should save and restore. An example of NVM table entry list is given below.

pData	ElemCount	ElemSize	EntryId	EntryType
0x1FFF9000	3	8	0xF1F4	MirroredInRam
0x1FFF7640	5	4	0xA2A6	NotMirroredInRam
0x1FFF1502	6	1	0x4212	NotMirroredInRam AutoRestore
0x1FFFF200	2	6	0x118F	MirroredInRam

NVM Table entry As show above, A NVM table entry contains a generic pointer to a contiguous RAM data structure, the number of elements the structure contains, the size of a single element, a table entry ID, and an entry type.

A RAM table entry has the following structure:

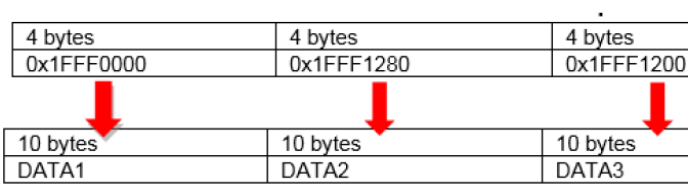
- pData (4 bytes) is a pointer to the RAM memory location where the dataset elements are stored.

- elemCnt (2 bytes) represents how many elements the dataset has.
- elemSz (2 bytes) is the size of a single element.
- entryID is a 16-bit unique ID of the dataset.
- dataEntryType is a 16-bit value representing the type of entry (mirrored/unmirrored/unmirrored auto restore).

For mirrored datasets, pData must point directly to the RAM data. For unmirrored datasets, it must be a double pointer to a vector of pointers. Each pointer in this table points to a RAM/FLASH area. Mirrored datasets require the data to be permanently kept in RAM, while unmirrored datasets have dataset entries either in flash or in RAM. If the unmirrored entries must be restored at the initialization, NotMirroredInRamAutoRestore should be used. The entryID gUnmirroredFeatureSet_d should be set to 1 for enabling unmirrored entries in the application. The last entry in the RAM table must have the entryID set to gNvEndOfTableId_c.

pData	0x1FFF8000
elemCnt	4
elemSz	10
entryID	1
dataEntryType	gNVM_NotM gNVM_NotM

The figure below provides an example of table entry :



When the data pointed to by the table entry pointer (pData) has changed (entirely or just a single element), the upper layers call the appropriate API function that requests the storage system to save the modified data. All the save operations (except for the synchronous save and atomic save) and the page erase and page copy operations are performed on system idle task. The application must create a task that calls NvIdle in an infinite loop. It should be created with OSA_PRIORITY_IDLE. However, the application may choose another priority. The save operations are done in one virtual page, which is the active page. After a save operation is performed on an unmirrored dataset, pData points to a flash location and the RAM pointer is freed. As a result, the effective data should always be allocated using the memory management module.

Active page The active page contains information about the records and the records. The storage system can save individual elements of a table entry or the entire table entry. Unmirrored datasets can only have individual saves. On mirrored datasets, the save/restore functions must receive the pointer to RAM data. For example, if the application must save the third element in the above vector, it should send 0x1FFF8000 + 2 * elemSz. For unmirrored datasets, the application must send the pointer that points to the area where the data is located. For example, if the application must save the third element in the above vector, it should send 0x1FFF8000 + 2 * sizeof(void*).

The page validity is guaranteed by the page counter. The page counter is a 32-bit value and is written at the beginning and at the end of the active page. The values need to be equal to consider the page a valid one. The value of the page counter is incremented after each page copy operation. A page erase operation is performed when the system is formatted. It is also performed when the page is full and a new record cannot be written into that page. Before being erased, the full page is first copied (only the most recent saves) and erased afterward.

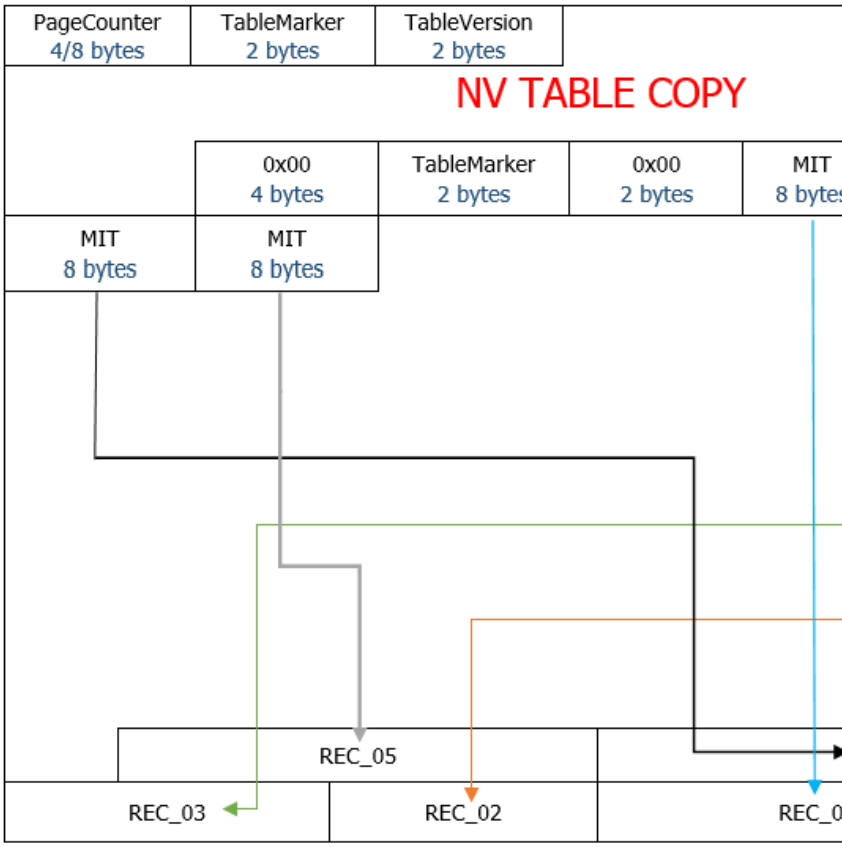
The validity of the Meta Information Tag (MIT), and, therefore, of a record, is guaranteed by the MIT start and stop validation bytes. These two bytes must be equal to consider the record

referred by the MIT valid. Furthermore, the value of these bytes indicates the type of the record, whether it is a single element or an entire table entry. The nonvolatile storage system allows dynamic changes of the table within the RAM memory, as follows:

- Remove table entry
- Register table entry

A new table entry can be successfully registered if there is at least one entry previously removed or if the NV table contains uninitialized table entries, declared explicitly to register new table entries at run time. A new table entry can also replace an existing one if the register table entry is called with an overwrite set to true. This functionality is disabled by default and must be enabled by the application by setting gNvUseExtendedFeatureSet_d to 1.

The layout of an active page is shown below:



As shown above, the table stored in the RAM memory is copied into the flash active page, just after the table version. The “table start” and “table end” are marked by the table markers. The data pointers from RAM are not copied. A flash copy of a RAM table entry has the following structure:



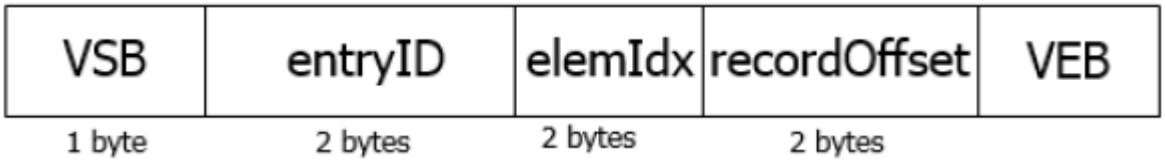
Where:

- entryID is the ID of the table entry
- entryType represents the type of the entry (mirrored/unmirrored/unmirrored auto restore)
- elemCnt is the elements count of that entry
- elemSz is the size of a single element

This copy of the RAM table in flash is used to determine whether the RAM table has changed. The table marker has a value of 0x4254 (“TB” if read as ASCII codes) and marks the beginning

and end of the NV table copy.

After the end of the RAM table copy, the Meta Information Tags (MITs) follow. Each MIT is used to store information related to one record. An MIT has the following structure:

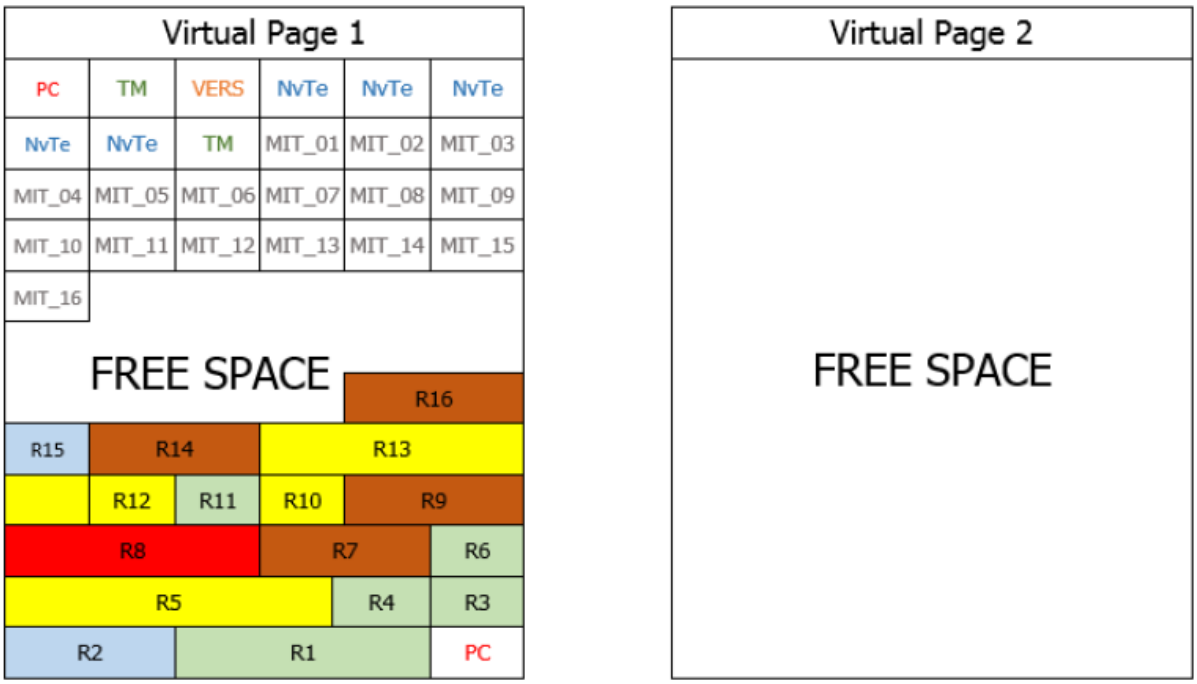


Where:

- VSB is the validation start byte.
- entryID is the ID of the NV table entry.
- elemIdx is the element index.
- recordOffset is the offset of the record related to the start address of the virtual page.
- VEB is the validation end byte.

A valid MIT has a VSB equal to a VEB. If the MIT refers to a single-element record type, VSB=VEB=0xAA. If the MIT refers to a full table entry record type (all elements from a table entry), VSB=VEB=0x55. Because the records are written to the flash page, the available page space decreases. As a result, the page becomes full and a new record does not have enough free space to be copied into that page.

In the example given below, the virtual page 1 is considered to be full if a new save request is pending and the page free space is not sufficient to copy the new record and the additional MIT. In this case, the latest saved datasets (table entries) are copied to virtual page 2.

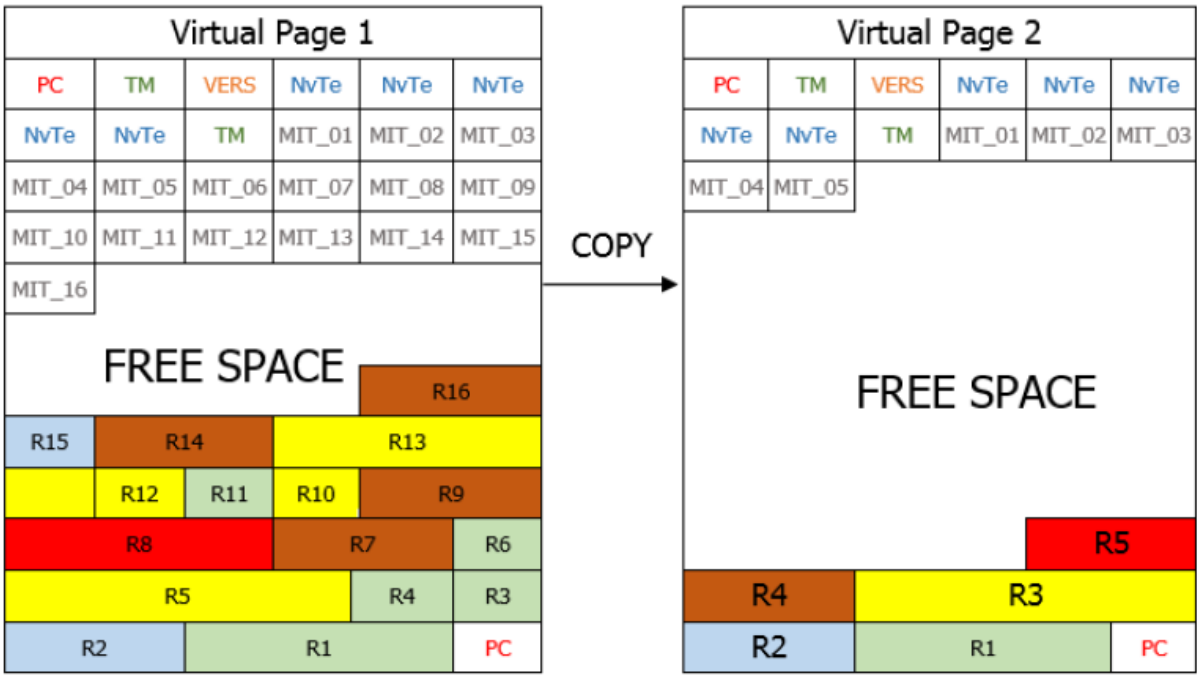


In this example, there are five datasets (one color for each dataset) with both 'full' and 'single' record types.

- R1 is a 'full' record type (contains all the NV table entry elements), whereas R3, R4, R6 and R11 are 'single' record types.
- R2 – full record type; R15 – single record type
- R5, R13 – full record type; R10, R12 – single record type

- R8 – full record type
- R7, R9, R14, R16 – full record type

As shown above, the R3, R4, R6, and R11 are 'single' record types, while R1 is a 'full' record type of the same dataset. When copied to virtual page 2, a defragmentation process takes place. As a result, the record copied to virtual page 2 has as much elements as R1, but individual elements are taken from R3, R4, R6, and R11. After the copy process completes, the virtual page 2 has five 'full' record types, one for each dataset. |This is illustrated below:



Finally, the virtual page 2 is validated by writing the PC value and a request to erase virtual page 1 is performed. The page is erased on an idle task, sector by sector where only one sector is erased at a time when idle task is executed.

If there is any difference between the RAM and flash tables, the application must call RecoverNvEntry for each entry that is different from its RAM copy to recover the entry data (ID, Type, ElemSz, ElemCnt) from flash before calling NvInit. The application must allocate the pData and change the RAM entry. It can choose to ignore the flash entry if the entry is not desired. If any entry from RAM differs from its flash equivalent at initialization, a page copy is triggered that ignores the entries that are different. In other words, data stored in those entries is lost.

The application can check if the RAM table was updated. In other words, if the MCU program was changed and the RAM table was updated, using the function GetFlashTableVersion and compare the result with the constant gNvFlashTableVersion_c. If the versions are different, NvInit detects the update and automatically upgrades the flash table. The upgrade process triggers a page copy that moves the flash data from the active page to the other one. It keeps the entries that were not modified intact and it moves the entries that had their elements count changed as follows:

- If the RAM element count is smaller than the flash element count, the upgrade only copies as many elements as are in RAM.
- If the RAM element count is larger than the flash element count, the upgrade copies all data from flash and fills the remaining space with data from RAM. If the entry size is changed, the entry is not copied. Any entryIds that are present in flash and not present in RAM are also not copied. This functionality is not supported if gNvUseExtendedFeatureSet_d is not set to 1.

ECC Fault detection The KW45/K32W1 internal flash is organized in 16 byte phrases and 8kB sectors (minimal erase unit). Its flash controller is synthesized so that it generates ECC information and an ECC generator / checker. During the programming of internal flash, errors may accidentally happen and cause ECC errors as a flash phrase is being written. These may happen due to multiple reasons:

- programmatic errors such as overwriting an already programmed phrase (transitioning bits from 0b to 1b). These are evitable by performing a blank check verification over phrase to be programmed, at the expense of processing power.
- occurrence of power drop or glitches during a programming operation.
- excessive wear of flash sector. The flash controller is capable of correcting one single ECC error but raises a bud fault whenever reading a phrase containing more than one ECC fault. Once an ECC error has 'infected' a flash phrase, the fault will remain and raise again at each read operation over the same phrase including blank check and prefetch. It can only be rid of by erasing the whole flash sector that contained the faulty phrase. In order to recover from situations where an ECC fault has occurred a gNvSalvageFromEccFault_d option has been added, which forces gNvVerifyReadBackAfterProgram_d to be defined to TRUE. If defined, the gNvVerifyReadBackAfterProgram_d option of the NVM module, causes the program to read back the programmed area after every flash programming operation. The verification is performed in safe mode if gNvSalvageFromEccFault_d is also defined. This is so as to detect ECC faults as early as possible as they appear, indeed when verifying a programming operation, one cannot be certain of the absence of ECC fault and avoid the bus fault. The safe API is thence used to perform the read back operation is performed using this safe API, so that we can tread in the flash and detect potential errors. The defects are detected on the fly whereas in the absence of safe read back, the error would cause a fault, potentially much later. During normal operation, assuming that no chip reset was provoked, this will consist in a single ECC fault either in the last record data or its meta information. Detecting such a fault calls for an immediate page copy to the other virtual page, so that the currently active page gets erased and the error gets cleared. Should the ECC fault occurs in the middle of a page copy operation, the switch of active page is postponed so that the fault page can be erased again and the copy can be restarted.

If the system underwent a power drop during a flash programming operation, sufficient to provoke a reset, at the ensuing reboot, ECC fault(s) may be present in the NVM area at the location that was being written. The detection is performed by an NVM sweeping mechanism, using the safe read API. That marks the faulty virtual page so that all subsequent reads within this virtual page are done with the safe API. If this case arises, a copy of the valid contents of the faulty page is attempted to the other virtual page. At NVM initialization, faults should be detected, either at the top of the meta data or at the bottom of the record area within the previous active page. This should guarantee that only the latest record write operation may be impaired. When the page copy has taken place, the faulty page is erased and the execution may resume. During NvCopyPage, when 'garbage collecting' occurs or whenever the current virtual active page needs to be transferred to the other virtual page, ECC errors are intercepted so that the operation can be attempted again in case of error. In case of NVM contents clobbering by programming errors, the salvage operation does its best to rescue as many records as possible but data will inevitably be lost.

An additional option -namely *gInterceptEccBusFaults_d* - was introduced in order to catch and correct ECC faults at Bus Fault handler level. Indeed, should an ECC bus fault fire, in spite of the precautions taken with NVM's gNvSalvageFromEccFault_d, we verify if the fault belongs to the NV storage. If so, a drastic policy can be adopted consisting in an erasure of the faulty sector. The corresponding Bus Fault handling is not part of the NVM, but dwells in the framework platform specific sources. Alternative handling could be implemented by the customer.

Save policy: Execution of program and erase operations on a flash an MCU core fetches code from cause perturbations of the core activity or requires to place critical code in RAM so that real-time ISR can still be served. The penalty of a sector erase is much higher than a simple program operation. The NVM is designed so as to limit the erase operations at 'garbage collecting' time,

so that flash wear is limited and no time is wasted. Several write policies are implemented to cope with the application constraints, one synchronous mode API and several posted write APIs. Among the posted write policies, the gNvmSaveOnIdleTimerPolicy_d compilation option selects a mode where flash write operations occur at time interval within the Idle task. Another option exists to 'randomize' the time interval with some jitter.

1) NvSyncSave performs a write synchronously with the disadvantage of stalling processor activity until comp
2) NvSaveOnCount posts a pending write operation and postpones the actual flash operation until number of record updates has reached a maximum. The actual write happens during Idle Task execution.see NvSetCountsBetweenSaves related API.
3) NvSaveOnInterval: posts a pending write operation and postpones the actual flash operation until the predefined number of ticks has elapsed. Optional mode - Active if (gNvmSaveOnIdleTimerPolicy_d & gNvmUseSaveOnTimerOn_c). see NvSetMinimumTicksBetweenSaves related API. Note that gNvmUseSaveIntervalJitter_c policy is a sub-option of gNvmSaveOnIdleTimerPolicy_d used to randomize slightly the time at which the write operation will happen.

Constant macro definition

- *gNvStorageIncluded_d* : If set to TRUE, it enables the whole functionality of the nonvolatile storage system. By default, it is set to FALSE (no code or data is generated for this module).
- *gNvUseFlexNVM_d* : If set to TRUE, it enables the FlexNVM functionality of the nonvolatile storage system. By default, it is set to FALSE. If FlexNVM is used, the standard nonvolatile storage system is disabled.
- *gNvFragmentation_Enabled_d* : Macro used to enable/disable the fragmented saves/restores (a particular element from a table entry can be saved or restored). It is set to FALSE by default.
- *gNvUseExtendedFeatureSet_d* : Macro used to enable/disable the extended feature set of the module:
  - Remove existing NV table entries
  - Register new NV table entries
  - Table upgrade

  It is set to FALSE by default.
- *gUnmirroredFeatureSet_d* : Macro used to enable unmirrored datasets. It is set to 0 by default.
- *gNvTableEntriesCountMax_c* : This constant defines the maximum count of the table entries (datasets) that the application is going to use. It is set to 32 by default.
- *gNvRecordsCopiedBufferSize_c* : This constant defines the size of the buffer used by the page copy function, when the copy operation performs defragmentation. The chosen value must be bigger than the maximum number of elements stored in any of the table entries. It is set by default to 64.
- *gNvCacheBufferSize_c* : This constant defines the size of the cache buffer used by the page copy function, when the copy operation does not perform defragmentation. The chosen value must be a multiple of 8. It is set by default to 64.
- *gNvMinimumTicksBetweenSaves_c* : This constant defines the minimum timer ticks between dataset saves (in seconds). It is set to 4 by default.
- *gNvCountsBetweenSaves_c* : This constant defines the number of calls to 'NvSaveOnCount' between dataset saves. It is set to 256 by default.

- *gNvInvalidDataEntry_c* : Macro used to mark a table entry as invalid in the NV table. The default value is 0xFFFFU.
- *gNvFormatRetryCount_c* : Macro used to define the maximum retries count value for the format operation. It is set to 3 by default.
- *gNvPendingSavesQueueSize_c* : Macro used to define the size of the pending saves queue. It is set to 32 by default.
- *gFifoOverwriteEnabled_c* : Macro used to enable overwriting older entries in the pending saves queue (if it is full). If it is FALSE and the queue is full, the module tries to process the oldest save in the queue to free a position. It is set to FALSE by default.
- *gNvMinimumFreeBytesCountStart_c* : Macro used to define the minimum free space at initialization. If the free space is smaller than this value, a page copy is triggered. It is set by default to 128.
- *gNvEndOfTableId_c* : Macro used to define the ID of the end-of-table entry. It is set to 0xFFFEU by default. No valid entry should use this ID.
- *gNvTableMarker_c* : Macro used to define the table marker value. The table marker is used to indicate the start and the end of the flash copy of the NV table. It is set to 0x4254U by default.
- *gNvFlashTableVersion_c* : Macro used to define the flash table version. It is used to determine if the NVM table was updated. It is set to 1 by default. The application should modify this every time the NVM table is updated and the data from NVM is still required.
- *gNvTableKeptInRam_d* : Set gNvTableKeptInRam_d to FALSE, if the NVM table is stored in FLASH memory (default). If the NVM table is stored in RAM memory, set the macro to TRUE.
- *gNvVerifyReadBackAfterProgram_d* : set by default force verification of NVM programming operations. Is forced implicitly when gNvSalvageFromEccFault_d is defined.
- *gNvSalvageFromEccFault_d* : use safe flash API to read from flash, and provide corrective action when ECC fault is met.

OtaSupport: Over-the-Air Programming Support

Overview This module includes APIs for the over-the-air image upgrade process. A Server device receives an image over the serial interface from a PC or other device thorough FSCI commands. If the Server has an image storage, the image is saved locally. If not, the image is requested chunk by chunk: With image storage

- OTA_RegisterToFsci()
- OTA_InitExternalMemory()
- OTA_WriteExternalMemory()
- ...
- OTA_WriteExternalMemory()

Without image storage:

- OTA_RegisterToFsci()
- OTA_QueryImageReq()
- OTA_ImageChunkReq()
- ...
- OTA_ImageChunkReq()

A Client device processes the received image by computing the CRC and filter unused data and stores the received image into a non-volatile storage. After the entire image has been transferred and verified, the Client device informs the Bootloader application that a new image is available, and that the MCU must be reset to start the upgrade process. See the following command sequence:

- OTA_StartImage()
- OTA_PushImageChunk() and OTA_CrcCompute ()
- ...
- OTA_PushImageChunk() and OTA_CrcCompute ()
- OTA_CommitImage()
- OTA_SetNewImageFlag()
- ResetMCU()

SecLib_RNG: Security library and random number generator

SecLib Implementation Flavors

Overview The NXP Connectivity Framework provides multiple SecLib implementations (flavors) to support different hardware capabilities, security requirements, and performance needs. Each flavor offers the same high-level cryptographic APIs but with different underlying implementations optimized for specific use cases.

Available SecLib Flavors

1. Software SecLib (SecLib.c**)(**RNG.c**)** **Description**: Pure software implementation providing cryptographic operations if no hardware is available.

Platforms: all platforms

Characteristics:

- Platform-independent software implementation
- No hardware acceleration requirements
- Consistent behavior across all platforms
- Higher CPU usage and power consumption
- Suitable for platforms without crypto hardware

Files: SecLib.c, SecLib_aes_mmo.c, RNG.c

2. EdgeLock SecLib (SecLib_sss.c**)(**RNG.c**)** **Description**: Hardware-accelerated implementation using NXP's EdgeLock secure subsystem s200.

Platforms: devices with S200: KW45, KW47, MCXW71, MCXW72

Characteristics:

- Hardware-backed security operations
- Secure key storage in hardware
- Hardware random number generation

- Lower CPU usage, better performance
- Enhanced security through hardware isolation
- Platform-specific (requires Secure Subsystem API)

Files: SecLib_sss.c, RNG.c

3. PSA Crypto SecLib (SecLib_psa.c**)(**RNG_psa.c**)** **Description**: Implementation based on ARM's Platform Security Architecture (PSA) Crypto API standard.

Platforms: MCXW23, (MCXW71 and MCXW72 for evaluation)

Characteristics:

- Industry-standard API
- Hardware acceleration when available
- Secure key management
- Future-proof and portable
- Compliance-ready

Files: SecLib_psa.c, SecLib_psa_config.h RNG_psa.c

4. MbedTLS SecLib (SecLib_mbedtls.c**)(**RNG_mbedTLS.c**)** **Description**: Implementation using the MbedTLS cryptographic library.

Platforms: all platforms, RW61

Characteristics:

- Well-tested and widely adopted
- Software and hardware acceleration support
- Extensive algorithm support
- Good performance optimization
- Large memory footprint

Files: SecLib_mbedTLS.c, SecLib_mbedtls_config.h, RNG_mbedTLS.c

Flavor Selection SecLib flavor is described in *'Kconfig'* and selected inside prj.conf or defconfig. Define one of this line to override the default configuration for your board:

```
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.sw=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.secure_subsystem=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.psa=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.mbedtls=y
```

The necessary configuration files are automatically added inside of CmakeLists.txt

Random number generator

Overview The RNG module is part of the framework used for random number generation. It uses hardware RNG peripherals as entropy sources (TRNG, Secure Subsystem, ...) to provide a true random number generator interface. A Pseudo-Random number generator (PRNG) implementation is available. The PRNG may depend of SecLib services (thus requiring a common mutex) to perform HMAC-SHA256, SHA256, AES-CTR, or alternateively a Lehmer Linear Congruential generator. A prerequisite for the PRNG to function with desired randomness is to be seeded using a proper source of entropy. If no hardware acceleration is present, the RNG may fallback to lesser quality ad-hoc source e.g if present SIM_UID registers, the UIDL is used as the initial seed for the random number generator.

Initialization The RNG module requires an initialization via a call to RNG_Init. The RNG initialization involves a call to RNG_SetSeed.

In the case of a dual core system consisting of a Host core and an NBU core, the Secure Subsystem is owned by the Host core. The Host core then has a direct access to its TRNG embedded in its secure subssystem. On the NBU code side, a request is emitted via RPMSG to the Host to provide a seed. On receipt of this request, the Host sets a 'reseed needed' flag (from the ISR context) If the core running the RNG service owns the TRNG entropy hardware (if any), it can get the seed directly form this hardware synchronously. In the case of an NBU that does not control the devices entropy source, that is owned by the Host, it request a seed from the Host processor via RPMSG exchange. On receipt of this request the Host sets a flag notifying of this request from the RPMSG ISR context.

RNG_ReInit API is to be used at wake up time in the context of LowPower. RNG_DeInit is used for unit tests and coverage purposes but has no useful role in a real application.

Asynchronous Seed Handling Enhancement The RNG module has been enhanced to use the framework's work queue system instead of polling from the IDLE thread.

Key Changes:

- **Work Queue Integration**: $\mathrm{RNG_NotifyReseedNeeded()}$ now submits seed operations to the work queue on Host cores
- **Non-blocking**: Replaces previous IDLE thread polling with asynchronous work queue scheduling
- **ISR-Safe**: Seed requests from ISR context are safely deferred to work queue execution

Benefits:

- Improved system responsiveness by removing IDLE thread dependency
- Better resource management through work queue scheduling

Seed handling RNG_SetSeed: RNG_SetExternalSeed may be used to inject application entropy to RNG context seed using a supplied array of bytes. RNG_IsReseedNeeded used from task in Host core to check whether seed must be sent to NBU core.

RNG_GetTrueRandomNumber is the API used to generate a Random 32 bit number from a HW source of entropy. It is essential if only to seed the pseudo random number generator.

RNG_GetPseudoRandomData is used to generate arrays of random bytes.

RNG_NotifyReseedNeeded Enhanced to use work queue submission on Host cores instead of requiring IDLE thread polling. Sets reseed flag and schedules asynchronous seed generation.

Security Library

Overview The framework provides support for cryptography in the security module. It supports both software and hardware encryption. Depending on the device, the hardware encryption uses either the S200, MMCAU, LTC, or CAU3 module instruction set or dedicated AES and SHA hardware blocks.

Software implementation is provided in a library format.

Support for security algorithms

	SW Seclib : SecLib.c	EdgeLock SecLib_sss.c	Seclib_p	Mbedtls SecLib_mbec	nccl (part of SecLib.c)	Usage example
AES_128	SecLib_aes.c	x	x	x		
AES_128_ECB		x	x	x		
AES_128_CBC	x	x		x		
AES_128_CTR encryption	x	x				
AES_128_OFB Encryption	x					
AES_128_CMAC	x	x	x	x		BLE connection, ieee 15.4
AES_128_EAX	x					
AES_128_CCM	x	x	x	x		BLE, ieee 15.4
SHA1	SecLib_sha.c	x		x		
SHA256	x	x	x	x		
HMAC_SHA256	x	x		x		PRNG, Digest for Matter
ECDH_P256 shared secret generation	x (by 15 incremental steps) -> SecLib_ecdh.c	x with MACRO SecLibECDHUseSSS	x	x	x	BLE pairing,
EC_P256 key pair generation	x	x	x	x	x	
EC_P256 public key generation from private key				x	x	Matter (ECDSA)
ECDSA_P256 hash and msg signature generation / verification		only if owner of the key pair		x	x	Matter
SPAKE2+ P256 arithmetics				x	x	Matter

SecLib_psa The framework provides PSA (Platform Security Architecture) Crypto API support through the SecLib_psa.c implementation. This provides a standardized cryptographic interface

that can leverage hardware acceleration when available.

Configuration PSA support is configured through the SecLib_psa_config.h file, which defines:

- Hardware acceleration capabilities via MBEDTLS_PSA_ACCEL_* macros
- Memory management integration with the framework's memory manager
- External RNG integration

Advantages of PSA Crypto API

Standardization and Portability

- **Vendor Independence**: Applications using PSA APIs can be easily ported between different hardware platforms without changing cryptographic code
- **Industry Standard**: PSA Crypto API is an ARM-defined industry standard (PSA Certified), ensuring consistent behavior across different platforms and vendors
- **Future-Proof**: As an evolving standard, PSA provides long-term compatibility and support for emerging cryptographic requirements

Security Benefits

- **Secure Key Management**: PSA provides built-in secure key storage and lifecycle management, reducing the risk of key exposure
- **Hardware Security Integration**: Seamless integration with hardware security modules (HSMs) and secure elements when available
- **Isolation**: Keys and cryptographic operations can be isolated from application code, reducing attack surface
- **Tamper Resistance**: Hardware-backed implementations provide protection against physical attacks

BLE advanced secure mode

New elements in existing structures: computeDhKeyParam_t::keepInternalBlob - boolean telling if the shared blob is kept in this structure(in .outpoint) after ECDH_P256_ComputeDhKey() or ECDH_P256_ComputeDhKeySeg() call.

New arguments in existing functions: ECDH_P256_ComputeDhKey keepBlobDhKey - boolean telling ECDH_P256_ComputeDhKey() or ECDH_P256_ComputeDhKeySeg() to keep the shared object after computation for later use (it is required by the SecLib_GenerateBluetoothF5KeysSecure).

New macros: gSecLibSssUseEncryptedKeys_d - Enable or disable S200 blobs SecLib support. 0 - the Bluetooth Keys are available in plaintext, 1 - the Bluetooth Keys are not available in plaintext, but in secured blobs. Default is disabled.

New functions:

LE Secure connections pairing:

void ECDH_P256_FreeDhKeyDataSecure This is a function used to free the shared object stored in computeDhKeyParam_t. When user calls ECDH_P256_ComputeDhKeySeg() with keepBlobDhKey set to 1, it should also call **ECDH_P256_FreeDhKeyDataSecure** .

SecLib_GenerateBluetoothF5Keys This function is extracted from the Bluetooth LE Host Stack implementation. This corresponds to the legacy implementation without key blobs.

SecLib_GenerateBluetoothF5KeysSecure Similar to **SecLib_GenerateBluetoothF5Keys** this function is modified to work with key blobs, the reason is to not use SSS inside the Bluetooth LE Host Stack.

SecLib_DeriveBluetoothSKD This is a helper function used by the Bluetooth LE Host Stack in the pairing procedure, when receiving the vendor HCI command specifying that the ESK needs to be provided to LL.

ELKE_BLE_SM_F5_DeriveKeys This is a private function, helper for **SecLib_GenerateBluetoothF5KeysSecure**. It was provided by the STEC team.

Privacy:

SecLib_ObfuscateKeySecure This is a function used by the Bluetooth LE Host Stack to obfuscate the IRK before setting it to Bluetooth LE Controller or before saving it to NVM

SecLib_DeobfuscateKeySecure This is a function used by the Bluetooth LE Host Stack to extract the plaintext IRK key from the saved NVM blob.

SecLib_VerifyBluetoothAh This function is extracted from the legacy Bluetooth LE Host Stack implementation using plaintext keys.

SecLib_VerifyBluetoothAhSecure Similar to **SecLib_VerifyBluetoothAh** with modification to work with S200 key blob.

SecLib_GenerateSymmetricKey This is a function used by the application to generate the local IRK and local CSRK.

SecLib_GenerateBluetoothEIRKBlobSecure This is a function used by the application to generate the EIRK needed by Bluetooth LE Controller from the IRK blob.

A2B feature

ECDH_P256_ComputeA2BKey This function is used to compute the EdgeLock to EdgeLock key. pInPeerPublicKey points to the peer public key, pOutE2EKey is the pointer to where the E2E key object will be stored, this will be freed by the application when it is no longer required by calling ECDH_P256_FreeE2EKeyData().

ECDH_P256_FreeE2EKeyData This function is used to free the key object given as a parameter. It is used by the application to free the E2E key when is no longer needed.

SecLib_ExportA2BBlobSecure This function is used to import an ELKE blob or plain text symmetric key in s200 and export an E2E key blob. The input type is identified by the keyType parameter.

SecLib_ImportA2BBlobSecure This function is used to import an E2E key blob in s200 and export an ELKE blob or plain text symmetric key. The output type is identified by the keyType parameter.

LE Secure connections Pairing flow and SecLib usage:

1. Each device needs to generate locally the public+private keypair. This is done using **ECDH_P256_GenerateKeys**.
2. Devices exchange their public keys.
3. Upon receiving the peer device's public key, local device is computing DH key using **ECDH_P256_ComputeDhKey**.
4. Each device sends DHKeyCheck packet
5. Upon receiving DhKeyCheck each device computes LTK blob using **SecLib_GenerateBluetoothF5Keys**
6. After computing the each device sends HCI_LeStartEnc (on initiator), HCI_Le_Provide_Long_Term_Key (on responder)
7. Bluetooth LE Controller sends back SKD report custom event
8. Bluetooth LE Host Stack computes ESKD based on LTK blob using **SecLib_DeriveBluetoothSKD** and sends it to Bluetooth LE Controller
9. Bluetooth LE Controller encrypts the link

IRK flow and SecLib usage:

1. At startup, when gInitializationComplete_c event is received:

- the local IRK is generated using **SecLib_GenerateSymmetricKey**
- the local EIRK is generated using **SecLib_GenerateBluetoothEIRKBlobSecure**
- local CSRK is generated using **SecLib_GenerateSymmetricKey**

2. During legacy pairing when receiving bonding keys, IRK is obfuscated using **SecLib_ObfuscateKeySecure** and stored
3. When app wants to set the OOB keys using Gap_SaveKeys the IRK is obfuscated using **SecLib_ObfuscateKeySecure**
4. When application calls API Gap_VerifyPrivateResolvableAddress IRK is obfuscated using **SecLib_ObfuscateKeySecure** and verified using **SecLib_VerifyBluetoothAhSecure**
5. When a new connection is received in Host with RPA address not resolved by the Bluetooth LE Controller, the Host tries to resolve it by obfuscating it using **SecLib_ObfuscateKeySecure** and verifying it using **SecLib_VerifyBluetoothAhSecure**
6. When adding a peer in Bluetooth LE Controller resolving list, the peer's IRK is obfuscated using **SecLib_ObfuscateKeySecure** before setting it using **HCI_Le_Add_Device_To_Resolving_List**.

7. When an IRK plaintext is requested by the application using Gap_LoadKeys it is obtained using **SecLib_DeobfuscateKeySecure**
8. When legacy pairing completes and LTK needs to be send in the pairing complete event (gConnEvtPairingComplete_c) the **SecLib_DeobfuscateKey** is used to extract the plaintext.

A2B flow and SecLib usage:

1. At startup, when gInitializationComplete_c event is received, the application will call **ECDH_P256_GenerateKeys** to generate the public/private key pair required for the E2E key derivation and send the public key to the peer device.
2. When the public key is received from the peer device, the application will call **ECDH_P256_ComputeA2BKeySecure** to generate the EdgeLock to EdgeLock key.
3. The application will obtain an E2E IRK blob by calling **SecLib_ExportA2BBlobSecure** with key type gSecElkeBlob_c. The obtained blob is sent to the peer anchor. The peer anchor will call **SecLib_ImportA2BBlob** with keyType gSecElkeBlob_c and save the resulting ELKE blob in NVM, for Digital Key both anchors must have the same IRK.
4. After pairing, in order to send the LTK and IRK contained in the bonding data securely, the application will call **SecLib_ExportA2BBlobSecure** with keyType gSecLtkElkeBlob_c for the LTK, and **SecLib_ExportA2BBlobSecure** with keyType gSecPlainText_c for the IRK. The E2E blobs obtained are sent along with the rest of the bonding data to the peer anchor device.
5. After the bonding data is trasfered the E2E key is no longer needed and **ECDH_P256_FreeE2EKeyData** is called with the key object obtained at step 2 when **ECDH_P256_ComputeA2BKeySecure** was called.

Sensors

Overview The Sensors module provides an API to communicate with the ADC. Two values can be obtained by this module :

- Temperature value
- Battery level

The temperature is given in tenths of degrees Celsius and the battery in percentage.

This module is multi-caller, the ADC is protected by a mutex on the resource and by pre-vententing lowpower (only WFI) during its processing. Platform specific code can be find in fwk_platform_sensors.c/h.

Features

Manual Measurement Calling “SENSORS_GetTemperature()” or “SENSORS_GetBatteryLevel()” function will give the last measured value that was copied to RAM. If you want to refresh this value you must first call the trigger function (SENSORS_TriggerTemperatureMeasurement() / SENSORS_TriggerBatteryMeasurement()) that configures the ADC to perform temperature or battery measurement, then a call to the refresh function (SENSORS_RefreshTemperatureValue() / SENSORS_RefreshBatteryLevel()) will read the returned value from the ADC and save it in RAM. You can run code between these two functions because the ADC won’t give you the value until it’s fully ready.

Important: When a measurement is ongoing, new measurement requests will be **dropped**. Measurement triggers are only accepted when the measurement state is **IDLE**. This applies to both temperature and battery measurements to prevent ADC resource conflicts.

Periodic Temperature Measurement The Sensors module supports automatic periodic temperature measurements that can be requested from two sources:

1. HOST Requests Applications can request periodic temperature measurements using:

```
SENSORS_TriggerPeriodicTemperatureMeasurement(uint32_t temperature_meas_interval_ms);
```

2. NBU Requests (wireless_mcu platforms only) The NBU (Narrow Band Unit) can request periodic temperature measurements. For more details about NBU temperature requests check NBU Integration Features section bellow.

Periodic Measurement Behavior **Immediate Trigger**: Every periodic request (HOST or NBU) triggers an **immediate measurement** before starting the interval timer.

Interval Management:

- **Minimum Interval Selection**: When both HOST and NBU request periodic measurements, the system uses the **shorter interval**
- **Dynamic Updates**: Changing the interval from either source immediately triggers a new measurement and restarts the timer with the new effective interval
- **Independent Control**: HOST and NBU can independently start/stop their periodic requests

One-shot Mode: Setting interval to 0 triggers a single measurement and stops periodic operation for that source.

Usage Examples

```
// HOST requests periodic measurement every 100ms
SENSORS_TriggerPeriodicTemperatureMeasurement(100);

// HOST requests one-shot measurement and stops periodic operation
SENSORS_TriggerPeriodicTemperatureMeasurement(0);
```

Concurrent Requests Behavior

HOST Interval	NBU Interval	Effective Interval	Notes
200ms	100ms	100ms	NBU interval is shorter
100ms	300ms	100ms	HOST interval is shorter
0 (stopped)	150ms	150ms	Only NBU active
200ms	0 (stopped)	200ms	Only HOST active
0 (stopped)	0 (stopped)	No periodic	Both stopped

Measurement State Management The Sensors module uses a state machine to manage ADC resource access:

States:

- **MEASUREMENT_IDLE**: No measurement in progress, ready to accept new requests
- **TMP_MEASUREMENT_ONGOING**: Temperature measurement in progress
- **BAT_MEASUREMENT_ONGOING**: Battery measurement in progress

Behavior:

- **Request Acceptance**: New measurement requests are only processed when state is MEASUREMENT_IDLE
- **Request Dropping**: If a measurement request is made while another measurement is ongoing, the new request is silently dropped
- **State Transitions**: State changes from IDLE ONGOING when measurement starts, and back to IDLE when measurement completes
- **Recovery Mechanism**: For periodic measurements, if a trigger fails due to ongoing measurement, a 1ms retry timer is started

NBU Integration Features

Platform Support: NBU integration features are only supported on **wireless_mcu** platforms. These features are **not available** on RT, mcxw23, and rw61x platforms.

Periodic Temperature Measurement from NBU The Sensors module supports automatic periodic temperature measurements triggered by NBU requests. This feature allows the system to automatically measure temperature at regular intervals without manual intervention. This temperature data is used for XTAL32M calibration and Channel Sounding calibration.

How it works:

1. **NBU Request**: The NBU can request periodic temperature measurements by calling the registered callback with a measurement interval
2. **Timer-based Scheduling**: A timer manager is used to schedule periodic measurements based on the requested interval
3. **Workqueue Processing**: Temperature measurement triggers are processed through the system workqueue to avoid blocking operations
4. **Automatic Refresh**: Once a measurement is complete, the next measurement is automatically scheduled

Key Components:

- **Temperature Request Callback**: Sensors_TemperatureReqCb() - Handles NBU requests for periodic measurements
- **Timer Callback**: Sensors_TempMeasTimerCallback() - Triggers measurements on timer expiration
- **Work Handler**: Sensors_PeriodicTempWorkHandler() - Processes temperature measurement requests in workqueue context
- **Ready Callback**: Sensors_TemperatureReadyCb() - Called when temperature measurement is complete

Usage:

- **One-shot measurement**: NBU requests with interval = 0
- **Periodic measurement**: NBU requests with interval > 0 (in milliseconds)
- The periodic measurement continues until a new request changes the interval or stops it

Temperature Filtering and NBU Communication

Modified Moving Average Filter On wireless_mcu platforms, temperature readings are processed through a Modified Moving Average (MMA) filter to reduce noise and prevent unnecessary NBU notifications. This filter helps to:

- **Limit XTAL32M calibration changes**: Prevents excessive calibration adjustments that could degrade RF performances
- **Optimize power consumption**: Reduces NBU wake-up events by limiting notifications to significant temperature changes
- **Reduce noise**: Smooths out temperature reading fluctuations for more stable system behavior

The XTAL32M calibration process uses a LUT (Look-Up Table) that contains pre-characterized calibration values mapping temperature ranges to optimal XTAL32M frequency adjustments, enabling precise crystal oscillator tuning based on thermal conditions.

The LUT is registered using `PLATFORM_RegisterXtal32MTempCompLut()` and this is currently done in `examples/_common/project_segments/wireless/wireless_mcu/app_common/hardware_init.c` and requires `gBoardUseXtal32MTempComp` to be enabled.

Configuration: Filter size is controlled by `PLATFORM_TEMPERATURE_FILTER_SIZE` (default: 4 samples)

NBU Temperature Threshold Temperature values are only sent to the NBU when they exceed a configurable threshold:

Threshold is Controlled by `PLATFORM_TEMP_SENT_NBU_THRESHOLD` (default: 10 tenths of degrees Celsius = 1°C)

Temperature Processing Flow:

1. **Raw Measurement**: ADC interrupt captures temperature reading
2. **Sensors Module Informed** Sensors referesh the temperature value
3. **Filtering**: Raw value is processed through MMA filter
4. **Threshold Check**: Filtered value is compared against last sent value ± threshold
5. **NBU Communication**: If threshold is exceeded:
    - XTAL32M calibration is performed (if LUT is available)
    - Temperature value is sent to NBU
    - Last sent value is updated

```
Temperature Request
        ↓
ADC Interrupt - Measurement is ready
        ↓
Raw Temperature Reading
```

(continues on next page)

(continued from previous page)

```
          ↓
Sensors Module Informed - Temperature refresh
          ↓
MMA Filter Processing
          ↓
Threshold Check: |filtered - last_sent| > threshold?
     ↓                    ↓
    YES                   NO
     ↓
LUT Available?
  ↓       ↓
 YES      NO
  ↓       ↓
XTAL32M     ↓
Calibration ↓
  ↓       ↓


     ↓
Send Temperature to NBU
     ↓
Update Last Value Sent to NBU
```

Platform Configuration Macros:

```
// Temperature filter size (number of samples for averaging)
#define PLATFORM_TEMPERATURE_FILTER_SIZE 4U

// Threshold for sending temperature to NBU (in tenths of degrees Celsius)
#define PLATFORM_TEMP_SENT_NBU_THRESHOLD 10  // 1°C
```

API Usage Examples

Manual Temperature Measurement

```
int32_t temperature1, temperature2;
SENSORS_TriggerTemperatureMeasurement();
...
temperature1 = (SENSORS_RefreshTemperatureValue())/10;
temperature2 = (SENSORS_GetTemperature())/10;
```

In this case temperature1 is equal to temperature2, both are in degree celsius.

Battery Level Measurement

```
uint8_t batteryLevel;
SENSORS_TriggerBatteryMeasurement();
...
batteryLevel = SENSORS_RefreshBatteryLevel();
// batteryLevel is now in percentage (0-100%)
```

Handling Concurrent Requests

```
// First request - will be accepted
SENSORS_TriggerTemperatureMeasurement();

// Second request while first is ongoing - will be dropped silently
```

(continues on next page)

(continued from previous page)

```
SENSORS_TriggerBatteryMeasurement();  // This will be ignored

// Wait for first measurement to complete
int32_t temp = SENSORS_RefreshTemperatureValue();

// Now battery measurement can be triggered
SENSORS_TriggerBatteryMeasurement();  // This will be accepted
```

Low Power Optimization - Wake-up Temperature Measurement

```
// In low power exit callback (interrupts masked, scheduler disabled)
/* Trig temperature measurement on wake-up.
 * If a temperature change is detected,
 * the new value will be sent to the NBU for appropriate XTAL32M trimming.
 * During channel sounding measurements, this temperature is also used for RTT compensation.
 * Since this function runs with interrupts masked and the scheduler disabled,
 * the unsafe variant must be used to avoid mutex acquisition.
 */
SENSORS_TriggerTemperatureMeasurementUnsafe();
```

This approach optimizes power consumption and provides additional benefits:

- **XTAL32M Trimming**: Temperature changes are sent to NBU for crystal oscillator calibration only if they exceed the configured threshold after filtering
- **RTT Compensation**: This temperature data is used by the NBU for RTT compensation during channel sounding measurements
- **Safe Execution**: Uses the unsafe variant to avoid mutex acquisition in interrupt context with scheduler disabled In case of periodic measurement:
- **Power Optimization**: Triggers temperature measurements when the system is already awake, avoiding dedicated timer wake-ups
- **Timer Reset**: Resets the periodic measurement timer to prevent unnecessary wake-ups

  Note: This optimization is implemented by default in wireless platform examples when gAppUseSensors_d is enabled. The temperature measurement is automatically triggered in the low power exit callback (BOARD_ExitLowPowerCb()) in examples/_common/project_segments/wireless/wireless_mcu/board_lp.c.

Constant macro definitions Name :

```
#define VALUE_NOT_AVAILABLE_8 0xFFu
#define VALUE_NOT_AVAILABLE_32 0xFFFFFFFFu
```

Description :

Defines the error value that can be compared to the value obtain on the ADC.

SFC : Smart Frequency Calibration

Overview The Smart Frequency Calibration module provides operations and calibration for the FRO32K source clock. This module is split between main core and Radio core:

- fwk_rf_sfc.[ch]: RF_SFC module on Radio core that provides Main FRO32K measurement/calibration and state machine in synchornization with Radio domain activities. See details below.

- fwk_sfc.h: SFC module on host core that provides type definition for usage with fwk_platform_ics.[ch] with PLATFORM_FwkSrvSetRfSfcConfig() API and fwk_platform_ble.c for received callback from the NBU core

Host SFC Module

Algorithm parametrization This module provides ability to configure the RF_SFC module by sending message to Radio core through fwk_platform_ics.c PLATFORM_FwkSrvSetRfSfcConfig():

- Filter size
- Maximum ppm threshold
- Maximum calibration interval
- Number of sample in filter to swicth from convergence to monitor mode

Ppm target The ppm target is the deviation from the target clock accepted by the algorithm. When the deviation is larger than the ppm target. The algorithm will update the trimming value and reset the filter. The ppm target cannot be more aggressive RF_SFC_MAXIMAL_PPM_TARGET in order to avoid having to update trimming value at each measurement.

Filter size Filter size must be included between RF_SFC_MINIMAL_FILTER_SIZE and RF_SFC_MAXIMAL_FILTER_SIZE. See *Filtering and Frequency estimation* section for more details on the parameter.

Maximum calibration interval In monitor mode, new measurement are triggered by low-power entry/exit. If the NBU core has a lot of radio activity it could never enter lowpower. The maximum calibration interval is here to ensure a measurement is done regularly. When executing idle the SFC module checks when the last measurement has been done, if it has been too long, it reset the filter and forces a new measurement

Trig sample number The trig sample number is the number of samples needed by the algorithm in its filter to switch from convergence to monitor mode. Having more than one sample in convergence mode allows to confirm the trimming value that we have set.

SFC debug information On the other way, the RF_SFC from Radio core sends back notifications to SFC module on main core using RX callback PLATFORM_RegisterFroNotificationCallback() from fwk_platform_ics.h and such information:

- last measured frequency
- average ppm from 32768Khz frequency
- last ppm measured from 32768Khz frequency
- FRO trimming value

RF_SFC module The RF_SFC module provides the functionality to calibrate the FRO32K source clock during Initialization and radio activity.

The RF_SFC is mostly used on XTAL32K less solution when no 32Khz crystal is soldered on the board. It allows to calibrate the FRO32K source clock to the desired frequency to keep Radio time base within the allowed tolerance given by the connectivity standards. However, even on a XTAL32K solution, the RF_SFC is also used during Initialization until the XTAL32K is up and

running in the system. The system firstly runs on the FRO32K clock source then switch to the XTAL32K clock source when it is ready with enough accuracy. This allows to save significant boot time as the FRO32K start up (including calibration) is much faster compared to XTAL32K .

This module will handle:

- FRO32K clock frequency measurement against 32Mhz crystal. It schedules appropriately the start of the measurement and gets the result when completed,
- Filter and estimate the 32Khz frequency value and error by averaging from the last measurements,
- FRO32K calibration in order to update the trimming value to reduce the frequency error on the clock.

The targeted frequency offset shall be within 200ppm. The RF_SFC will handle two modes of operation:

- Convergence mode: when frequency estimation is above 200pm,
- Monitor mode: when frequency estimation is below 200pm.

The RF_SFC module works in active and all low power modes on NBU domain, or on host application domain except power down mode. Power down mode on host application domain is not supported with the FRO32K configuration as clock source.

Feature enablement Enabling the FRO32K is done by calling the PLATFORM_InitFro32K() function during application initialization in hardware_init.c file, in BOARD_InitHardware() function. If FRO32K is not enabled, Oscillator XTAL32K shall be called instead by calling PLATFORM_InitOsc32K() function. The call to PLATFORM_InitFro32K() from BOARD_InitHardware() can be done by setting the Compilation flag gBoardUseFro32k_d to 1 in hardware_init.c or any header files included from this file.

```
#define gBoardUseFro32k_d       1
```

Detailed description

Frequency measurements When NBU low power is enabled, the frequency measurements are triggered on Low power wake-up by HW signal. The SFC process called from Idle task will check regularly the completion of the frequency measurement. When the measurement is done, it goes to filtering and frequency estimation process. The frequency measurement duration depends on monitor mode or convergence mode: In convergence mode, the frequency measurement duration is 0.5ms while it is 2ms in monitor mode. In monitor mode, the duration value remains less than the minimal radio activity duration so it does not impact the low power consumption in monitoring mode.

Filtering and Frequency estimation The FRO32KHz frequency measurement values are noisy because of thermal noise on the FRO32K itself. Also, the frequency measurement can introduce some error. In monitoring mode, it is required to filter the measurements by applying an exponential filter. new_estimation = (new_measurement + ((1 « n) -1) * last_estimation) » n

Default value for n is 7 (meaning 128 samples in the averaging window).

Frequency calibration When the frequency estimation gets higher than the targeted 200ppm target, the RF_SFC updates the trimming value for one positive or negative increment. For this purpose, it requires to:

- wake up the host application domain and keep the domain active,

- update the trim register of the FRO32K , this register is used to trim the capacitance value of the FRO32K,
- re-allow the host application domain to enter low power.

A slight power impact is expected during a calibration update due to host domain wake-up.

Operational modes When the low power mode is enabled on NBU power domain, RF SFC handles two modes of operation: convergence and monitor modes. However, when low power is disabled on NBU power domain, only convergence mode is supported.

Convergence mode Convergence mode is used when the estimated FRO32K frequency is above 200ppm or when the filter has been reset. Typically this occurs :

- During Power ON reset or other reset when NBU is switched OFF
- When temperature varies and FRO32K frequency deviates outside 200ppm threshold target
- When no calibration has been done during some time as we discard old values that could influence the algorithm

The convergence mode process typically starts with a FRO32K trim register update, performs a frequency measurement and the FRO32K trim register is updated until the measured frequency gets below 200ppm. These operations are repeated in a loop until the estimated frequency value gets below 200ppm. When below 200ppm during multiple measurements, the RC SFC switches to Monitoring mode. The convergence mode is only a transition mode to monitoring mode. In convergence mode, the NBU power domain does not go to low power. The convergence mode time duration depends on the initial frequency error of the FR032K. Default frequency measurement duration is 0.5ms so 20 measurements (given as example only) will require less than 10 ms to converge.

Monitoring mode Monitoring mode is used when the estimated FRO32K frequency is below 200ppm. In this mode, the measurement is triggered on NBU domain wake up from low power mode using an internal hardware signal. The exponential filter is applied to compute the frequency estimation. If the frequency estimation value is still within 200ppm, the NBU power domain is allowed to go to low power. If the estimated value gets above the 200ppm threshold, the RF SFC switch back to convergence mode. The trim register is updated by one increment (positive or negative) and because the frequency has been adjusted and changed, the estimated filtered frequency is reset to discard all previous measurements. Going back to convergence mode typically happens during a temperature gradient. If the temperature is constant, it is not expected to have the estimated value to go beyond 200ppm so no calibration should be required.

Initialization and configuration During initialization, the RF SFC module will block the Radio Software until monitoring mode is reached. This is to prevent the radio from running with an inaccurate time base due to an important 32k clock frequency error.

Initialization and configuration is done by the NBU core. The configuration parameters can set up:

- The 200ppm target threshold. This value shall be 200ppm or higher.
- The filtering number n (see section above), It shall be between 0 and 8. Default is 7 which is similar to an averaging filter of 128 samples. A higher value will be more robust against noise. A lower value will track temperature variation more faster.

In order to prevent the host application domain from going into power down mode (power down mode not supported with FRO32K as clock source), the fwkSrvLowPowerConstraintCallbacks functions structure is registered to the Framework service on host application core from fwk_platform_lowpower.c file, PLATFORM_LowPowerInit() function. The NBU code applies a

low power Deep Sleep constraint to the application core. This constraint is released when the NBU firmware has no activity to do and re-applied when a new activity starts.

Lowpower impact

Power impact during active mode: In monitoring mode (this should be 99.9% of the time if temperature does not vary), the FRO32KHz frequency measurements are performed during a Radio activity so it does not increase the active current as the sources clocks are already active. Also, it does not increase the active time as the measurement takes less time than an advertising event or connection event so no impact on power consumption.

The main power impact will be in convergence mode. In this case, measurements/calibrations are done in loop until the monitoring mode is reached (frequency error less than 200ppm). This could happen:

- During power ON reset,
- When temperature varies: The frequency will deviate from 32768Hz and FRO32K trimming register correction will need to be updated for that,
- When no measurement has been done during some time as we cannot predict if the FRO has drifted, so we discard older values and start convergence mode.

When FRO32K frequency needs to be adjusted, the NBU core will wake-up the main power domain and will update the FRO32K trimming register.

Power impact during low power mode: The power consumption in low power mode will increase slightly due to running FRO32K compared to XTAL32K. The power consumption of FRO32K typically consumes 350nA while it is only 100nA with XTAL32K. Refer to the product datasheet for the exact numbers.

Chapter 2

RTOS

2.1 FreeRTOS

2.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

FreeRTOS kernel for MCUXpresso SDK ChangeLog

FreeRTOS kernel Readme

2.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

2.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

2.1.4 corehttp

C language HTTP client library designed for embedded platforms.

2.1.5 corejson

JSON parser.

Readme

2.1.6 coremqtt

MQTT publish/subscribe messaging library.

2.1.7 corepkcs11

PKCS #11 key management library.

Readme

2.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme

Index

Non-alphabetical

___noreturn (*C macro*), 600

A

addDifferentSecurityModes (*C macro*), 447
addMode1AndMode2 (*C macro*), 447
addSameSecurityModes (*C macro*), 447
attErrorCode_t (*C enum*), 488
attErrorCode_t.gAttErrCodeAttributeNotFound_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeAttributeNotLong_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeCccdImproperlyConfigured_c (*C enumerator*), 490
attErrorCode_t.gAttErrCodeDatabaseOutOfSync_c (*C enumerator*), 490
attErrorCode_t.gAttErrCodeInsufficientAuthentication_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInsufficientAuthorization_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInsufficientEncryption_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInsufficientEncryptionKeySize_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInsufficientResources_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInvalidAttributeValueLength_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInvalidHandle_c (*C enumerator*), 488
attErrorCode_t.gAttErrCodeInvalidOffset_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeInvalidPdu_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeNoError_c (*C enumerator*), 488
attErrorCode_t.gAttErrCodeOutOfRange_c (*C enumerator*), 490
attErrorCode_t.gAttErrCodePrepareQueueFull_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeProcedureAlreadyInProgress_c (*C enumerator*), 490
attErrorCode_t.gAttErrCodeReadNotPermitted_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeRequestNotSupported_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeUnlikelyError_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeUnsupportedGroupType_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeValueNotAllowed_c (*C enumerator*), 490
attErrorCode_t.gAttErrCodeWriteNotPermitted_c (*C enumerator*), 489
attErrorCode_t.gAttErrCodeWriteRequestRejected_c (*C enumerator*), 490

B

bearerId_t (*C type*), 582
bearerStatus_t (*C type*), 582
bearerStatus_tag (*C enum*), 554
bearerStatus_tag.gEattBearerActive_c (*C enumerator*), 554
bearerStatus_tag.gEattBearerAllocated_c (*C enumerator*), 554
bearerStatus_tag.gEattBearerAlreadyAllocated_c (*C enumerator*), 554
bearerStatus_tag.gEattBearerFree_c (*C enumerator*), 554
bearerStatus_tag.gEattBearerReconfInProgress_c (*C enumerator*), 554
bearerStatus_tag.gEattBearerSuspended_c (*C enumerator*), 554
Ble_CheckMemoryStorage (*C function*), 587
Ble_CopyDeviceAddress (*C macro*), 588
Ble_DeviceAddressesMatch (*C macro*), 588

Ble_HciRecv (*C function*), 586
Ble_HciRecvFromIsr (*C function*), 586
Ble_HostDeInitialize (*C function*), 586
Ble_HostInitialize (*C function*), 585
Ble_IsPrivateNonresolvableDeviceAddress (*C macro*), 588
Ble_IsPrivateResolvableDeviceAddress (*C macro*), 588
Ble_IsRandomStaticDeviceAddress (*C macro*), 588
bleAddressType_t (*C type*), 582
bleAdvertiserClockAccuracy_t (*C type*), 582
bleAdvertiserClockAccuracy_tag (*C enum*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc20ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc30ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc50ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc75ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc100ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc150ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc250ppm_c (*C enumerator*), 557
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc500ppm_c (*C enumerator*), 557
bleAdvertisingChannelMap_t (*C struct*), 601, 635
bleAdvertisingChannelMap_t.enableChannel37 (*C var*), 602, 636
bleAdvertisingChannelMap_t.enableChannel38 (*C var*), 602, 636
bleAdvertisingChannelMap_t.enableChannel39 (*C var*), 602, 636
bleAdvertisingChannelMap_t.reserved (*C var*), 602, 636
bleAdvertisingFilterPolicy_t (*C enum*), 556
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromAnyAllowConnFromAny_c (*C enumerator*), 556
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromAnyAllowConnFromWL_c (*C enumerator*), 556
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromWLAllowConnFromAny_c (*C enumerator*), 556
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromWLAllowConnFromWL_c (*C enumerator*), 556
bleAdvertisingReportEventType_t (*C enum*), 557
bleAdvertisingReportEventType_t.gBleAdvRepAdvDirectInd_c (*C enumerator*), 557
bleAdvertisingReportEventType_t.gBleAdvRepAdvInd_c (*C enumerator*), 557
bleAdvertisingReportEventType_t.gBleAdvRepAdvNonconnInd_c (*C enumerator*), 557
bleAdvertisingReportEventType_t.gBleAdvRepAdvScanInd_c (*C enumerator*), 557
bleAdvertisingReportEventType_t.gBleAdvRepScanRsp_c (*C enumerator*), 557
bleAdvertisingType_t (*C enum*), 554
bleAdvertisingType_t.gAdvConnectableUndirected_c (*C enumerator*), 554
bleAdvertisingType_t.gAdvDirectedHighDutyCycle_c (*C enumerator*), 554
bleAdvertisingType_t.gAdvDirectedLowDutyCycle_c (*C enumerator*), 554
bleAdvertisingType_t.gAdvNonConnectable_c (*C enumerator*), 554
bleAdvertisingType_t.gAdvScannable_c (*C enumerator*), 554
bleAdvIndexType_t (*C enum*), 581
bleAdvIndexType_t.gAdvIndexAscend_c (*C enumerator*), 581
bleAdvIndexType_t.gAdvIndexDescend_c (*C enumerator*), 582
bleAdvIndexType_t.gAdvIndexRandom_c (*C enumerator*), 582
bleAdvIndexType_t.gAdvIndexUser_c (*C enumerator*), 582
bleAdvReportEventProperties_t (*C type*), 582
bleAdvReportEventProperties_tag (*C enum*), 554
bleAdvReportEventProperties_tag.gAdvEventAnonymous_c (*C enumerator*), 555
bleAdvReportEventProperties_tag.gAdvEventConnectable_c (*C enumerator*), 555
bleAdvReportEventProperties_tag.gAdvEventDirected_c (*C enumerator*), 555
bleAdvReportEventProperties_tag.gAdvEventLegacy_c (*C enumerator*), 555
bleAdvReportEventProperties_tag.gAdvEventScannable_c (*C enumerator*), 555
bleAdvReportEventProperties_tag.gAdvEventScanResponse_c (*C enumerator*), 555
bleAdvRequestProperties_t (*C type*), 582

bleAdvRequestProperties_tag (*C enum*), 555
bleAdvRequestProperties_tag.gAdvIncludeADIinDecisionPDU_c (*C enumerator*), 556
bleAdvRequestProperties_tag.gAdvIncludeAdvAinDecisionPDU_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvIncludeTxPower_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqAnonymous_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqConnectable_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqDirected_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqHighDutyCycle_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqLegacy_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvReqScannable_c (*C enumerator*), 555
bleAdvRequestProperties_tag.gAdvUseDecisionPDU_c (*C enumerator*), 555
bleAntennaInformation_t (*C type*), 583
bleAntennaInformation_tag (*C struct*), 604, 638
bleAntennaInformation_tag.maxCteLength (*C var*), 638
bleAntennaInformation_tag.maxSwitchingPatternLength (*C var*), 638
bleAntennaInformation_tag.numAntennae (*C var*), 638
bleAntennaInformation_tag.supportedSwitchingSamplingRates (*C var*), 638
bleBondCreatedEvent_t (*C type*), 583
bleBondCreatedEvent_tag (*C struct*), 603, 637
bleBondCreatedEvent_tag.address (*C var*), 604, 638
bleBondCreatedEvent_tag.addressType (*C var*), 603, 638
bleBondCreatedEvent_tag.nvmIndex (*C var*), 603, 638
bleBondDataBlob_t (*C struct*), 616, 649
bleBondDataBlob_t.bondDataBlobDeviceInfo (*C var*), 649
bleBondDataBlob_t.bondDataBlobDynamic (*C var*), 649
bleBondDataBlob_t.bondDataBlobLegacy (*C var*), 649
bleBondDataBlob_t.bondDataBlobStatic (*C var*), 649
bleBondDataBlob_t.bondDataDescriptors (*C var*), 649
bleBondDataBlob_t.bondHeader (*C var*), 649
bleBondDataDescriptorBlob_t (*C struct*), 616, 648
bleBondDataDescriptorBlob_t.raw (*C var*), 649
bleBondDataDeviceInfoBlob_t (*C struct*), 616, 648
bleBondDataDeviceInfoBlob_t.raw (*C var*), 648
bleBondDataDynamicBlob_t (*C struct*), 616, 648
bleBondDataDynamicBlob_t.raw (*C var*), 648
bleBondDataLegacyBlob_t (*C struct*), 616, 648
bleBondDataLegacyBlob_t.raw (*C var*), 648
bleBondDataRam_t (*C struct*), 616, 649
bleBondDataRam_t.bondDataBlobDeviceInfo (*C var*), 649
bleBondDataRam_t.bondDataBlobDynamic (*C var*), 649
bleBondDataRam_t.bondDataBlobLegacy (*C var*), 649
bleBondDataRam_t.bondDataBlobStatic (*C var*), 649
bleBondDataRam_t.bondDataDescriptors (*C var*), 649
bleBondDataRam_t.bondHeader (*C var*), 649
bleBondDataStaticBlob_t (*C struct*), 616, 648
bleBondDataStaticBlob_t.raw (*C var*), 648
bleBondIdentityHeaderBlob_t (*C struct*), 616, 648
bleBondIdentityHeaderBlob_t.raw (*C var*), 648
bleCentralClockAccuracy_t (*C type*), 582
bleCentralClockAccuracy_tag (*C enum*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc20ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc30ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc50ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc75ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc100ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc150ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc250ppm_c (*C enumerator*), 556
bleCentralClockAccuracy_tag.gBleCentralClkAcc500ppm_c (*C enumerator*), 556

bleChannelFrequency_t (*C enum*), 558
bleChannelFrequency_t.gBleFreq2402MHz_c (*C enumerator*), 558
bleChannelFrequency_t.gBleFreq2404MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2406MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2408MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2410MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2412MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2414MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2416MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2418MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2420MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2422MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2424MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2426MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2428MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2430MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2432MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2434MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2436MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2438MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2440MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2442MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2444MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2446MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2448MHz_c (*C enumerator*), 559
bleChannelFrequency_t.gBleFreq2450MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2452MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2454MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2456MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2458MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2460MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2462MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2464MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2466MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2468MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2470MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2472MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2474MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2476MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2478MHz_c (*C enumerator*), 560
bleChannelFrequency_t.gBleFreq2480MHz_c (*C enumerator*), 560
bleChannelMap_t (*C type*), 583
bleChannelOverrideMode_t (*C enum*), 558
bleChannelOverrideMode_t.gOverrideAdvChannels_c (*C enumerator*), 558
bleChannelOverrideMode_t.gOverrideInitChannels_c (*C enumerator*), 558
bleChannelOverrideMode_t.gOverrideScanChannels_c (*C enumerator*), 558
bleCteAllowedTypesMap_t (*C struct*), 616, 649
bleCteAllowedTypesMap_t.allowAoA (*C var*), 616, 650
bleCteAllowedTypesMap_t.allowAoD1us (*C var*), 616, 650
bleCteAllowedTypesMap_t.allowAoD2us (*C var*), 616, 650
bleCteAllowedTypesMap_t.reserved (*C var*), 616, 650
bleCteReqEnable_t (*C enum*), 578
bleCteReqEnable_t.gCteReqDisable_c (*C enumerator*), 578
bleCteReqEnable_t.gCteReqEnable_c (*C enumerator*), 578
bleCteRspEnable_t (*C enum*), 578
bleCteRspEnable_t.gCteRspDisable_c (*C enumerator*), 578
bleCteRspEnable_t.gCteRspEnable_c (*C enumerator*), 578
bleCteTransmitEnable_t (*C enum*), 578

bleCteTransmitEnable_t.gCteTransmitDisable_c (*C enumerator*), 578
bleCteTransmitEnable_t.gCteTransmitEnable_c (*C enumerator*), 578
bleCteType_t (*C type*), 584
bleCteType_tag (*C enum*), 578
bleCteType_tag.gCteTypeAoA_c (*C enumerator*), 578
bleCteType_tag.gCteTypeAoD1us_c (*C enumerator*), 579
bleCteType_tag.gCteTypeAoD2us_c (*C enumerator*), 579
bleCteType_tag.gCteTypeNoCte_c (*C enumerator*), 579
bleDeviceAddress_t (*C type*), 582
bleGetConnParamsMode_t (*C enum*), 581
bleGetConnParamsMode_t.gSendEventAfterLLProcUpdate_c (*C enumerator*), 581
bleGetConnParamsMode_t.gSendEventAfterRemoteSNChange_c (*C enumerator*), 581
bleHandoverAnchorNotificationEnable_t (*C enum*), 581
bleHandoverAnchorNotificationEnable_t.gAnchorNotificationDisable_c (*C enumerator*), 581
bleHandoverAnchorNotificationEnable_t.gAnchorNotificationEnable_c (*C enumerator*), 581
bleHandoverAnchorSearchMode_t (*C enum*), 581
bleHandoverAnchorSearchMode_t.gPacketMode_c (*C enumerator*), 581
bleHandoverAnchorSearchMode_t.gRssiSniffingMode_c (*C enumerator*), 581
bleHandoverAnchorSearchMode_t.gSuspendTxMode_c (*C enumerator*), 581
bleHandoverSuspendTransmitMode_t (*C enum*), 580
bleHandoverSuspendTransmitMode_t.gDoNotUseEventCounter_c (*C enumerator*), 580
bleHandoverSuspendTransmitMode_t.gSafeStopHostLlTx_c (*C enumerator*), 581
bleHandoverSuspendTransmitMode_t.gSafeStopLlTx_c (*C enumerator*), 581
bleHandoverSuspendTransmitMode_t.gUseEventCounter_c (*C enumerator*), 580
bleHandoverTimeSyncEnable_t (*C enum*), 581
bleHandoverTimeSyncEnable_t.gTimeSyncDisable_c (*C enumerator*), 581
bleHandoverTimeSyncEnable_t.gTimeSyncEnable_c (*C enumerator*), 581
bleHandoverTimeSyncStopWhenFound_t (*C enum*), 581
bleHandoverTimeSyncStopWhenFound_t.gTimeSyncDoNotStopWhenFound_c (*C enumerator*), 581
bleHandoverTimeSyncStopWhenFound_t.gTimeSyncStopWhenFound_c (*C enumerator*), 581
bleHardwareErrorCode_t (*C enum*), 561
bleHardwareErrorCode_t.bleHwErrCodeNoError_c (*C enumerator*), 561
bleIdentityAddress_t (*C struct*), 601, 635
bleIdentityAddress_t.idAddress (*C var*), 601, 635
bleIdentityAddress_t.idAddressType (*C var*), 601, 635
bleInitiatorFilterPolicy_t (*C type*), 582
bleIqReportPacketStatus_t (*C type*), 585
bleIqReportPacketStatus_tag (*C enum*), 579
bleIqReportPacketStatus_tag.gIqReportPacketStatusCorrectCrc_c (*C enumerator*), 579
bleIqReportPacketStatus_tag.gIqReportPacketStatusCrcIncorrectUsedLength_c (*C enumerator*), 579
bleIqReportPacketStatus_tag.gIqReportPacketStatusCrcIncorrectUsedOther_c (*C enumerator*), 579
bleIqReportPacketStatus_tag.gIqReportPacketStatusInsufficientResources_c (*C enumerator*), 579
bleIqSamplingEnable_t (*C enum*), 578
bleIqSamplingEnable_t.gIqSamplingDisable_c (*C enumerator*), 578
bleIqSamplingEnable_t.gIqSamplingEnable_c (*C enumerator*), 578
bleLlConnectionRole_t (*C enum*), 556
bleLlConnectionRole_t.gBleLlConnectionCentral_c (*C enumerator*), 556
bleLlConnectionRole_t.gBleLlConnectionPeripheral_c (*C enumerator*), 556
bleLocalKeysBlob_t (*C struct*), 616, 649
bleLocalKeysBlob_t.___unnamed2___ (*C union*), 621, 653
bleLocalKeysBlob_t.___unnamed2___.pKey (*C var*), 621, 653
bleLocalKeysBlob_t.___unnamed2___.raw (*C var*), 621, 653
bleLocalKeysBlob_t.keyGenerated (*C var*), 649
bleMonAdvCondition_t (*C enum*), 381
bleMonAdvCondition_t.gBleMonAdvConditionRssiHighThreshold_c (*C enumerator*), 381
bleMonAdvCondition_t.gBleMonAdvConditionRssiLowThreshold_c (*C enumerator*), 381
bleNotificationEvent_t (*C struct*), 603, 637
bleNotificationEvent_t.adv_handle (*C var*), 603, 637

bleNotificationEvent_t.ce_counter (*C var*), 603, 637
bleNotificationEvent_t.channel (*C var*), 603, 637
bleNotificationEvent_t.deviceId (*C var*), 603, 637
bleNotificationEvent_t.eventType (*C var*), 603, 637
bleNotificationEvent_t.rssi (*C var*), 603, 637
bleNotificationEvent_t.scanned_addr (*C var*), 603, 637
bleNotificationEvent_t.status (*C var*), 603, 637
bleNotificationEvent_t.timestamp (*C var*), 603, 637
bleNotificationEventType_t (*C type*), 583
bleNotificationEventType_tag (*C enum*), 577
bleNotificationEventType_tag.gNotifAdvConnReqRx_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifAdvEventOver_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifAdvScanReqRx_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifAdvTx_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifChannelMatrix_c (*C enumerator*), 578
bleNotificationEventType_tag.gNotifConnChannelMapUpdate_c (*C enumerator*), 578
bleNotificationEventType_tag.gNotifConnCreated_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifConnEventOver_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifConnInd_c (*C enumerator*), 578
bleNotificationEventType_tag.gNotifConnRxPdu_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifEventNone_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifPhyReq_c (*C enumerator*), 578
bleNotificationEventType_tag.gNotifPhyUpdateInd_c (*C enumerator*), 578
bleNotificationEventType_tag.gNotifScanAdvPktRx_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifScanEventOver_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifScanReqTx_c (*C enumerator*), 577
bleNotificationEventType_tag.gNotifScanRspRx_c (*C enumerator*), 577
blePathLossReportingEnable_t (*C enum*), 580
blePathLossReportingEnable_t.gPathLossReportingDisable_c (*C enumerator*), 580
blePathLossReportingEnable_t.gPathLossReportingEnable_c (*C enumerator*), 580
blePathLossThresholdZoneEntered_t (*C type*), 585
blePathLossThresholdZoneEntered_tag (*C enum*), 579
blePathLossThresholdZoneEntered_tag.gPathLossThresholdHighZone_c (*C enumerator*), 579
blePathLossThresholdZoneEntered_tag.gPathLossThresholdLowZone_c (*C enumerator*), 579
blePathLossThresholdZoneEntered_tag.gPathLossThresholdMiddleZone_c (*C enumerator*), 579
blePowerControlPhyType_t (*C type*), 585
blePowerControlPhyType_tag (*C enum*), 580
blePowerControlPhyType_tag.gPowerControlLePhy1M_c (*C enumerator*), 580
blePowerControlPhyType_tag.gPowerControlLePhy2M_c (*C enumerator*), 580
blePowerControlPhyType_tag.gPowerControlLePhyCodedS2_c (*C enumerator*), 580
blePowerControlPhyType_tag.gPowerControlLePhyCodedS8_c (*C enumerator*), 580
blePrivacyMode_t (*C type*), 582
bleResult_t (*C type*), 582
bleResult_tag (*C enum*), 540
bleResult_tag.gAttStatusBase_c (*C enumerator*), 551
bleResult_tag.gAttSuccess_c (*C enumerator*), 551
bleResult_tag.gBleAlreadyInitialized_c (*C enumerator*), 541
bleResult_tag.gBleFeatureNotSupported_c (*C enumerator*), 540
bleResult_tag.gBleInvalidParameter_c (*C enumerator*), 540
bleResult_tag.gBleInvalidState_c (*C enumerator*), 541
bleResult_tag.gBleNVMError_c (*C enumerator*), 541
bleResult_tag.gBleOsError_c (*C enumerator*), 541
bleResult_tag.gBleOutOfMemory_c (*C enumerator*), 541
bleResult_tag.gBleOverflow_c (*C enumerator*), 540
bleResult_tag.gBleReassemblyInProgress_c (*C enumerator*), 541
bleResult_tag.gBleRngError_c (*C enumerator*), 541
bleResult_tag.gBleSecLibError_c (*C enumerator*), 541
bleResult_tag.gBleStatusBase_c (*C enumerator*), 540

bleResult_tag.gBleSuccess_c (*C enumerator*), 540
bleResult_tag.gBleTimerError_c (*C enumerator*), 541
bleResult_tag.gBleUnavailable_c (*C enumerator*), 540
bleResult_tag.gBleUnexpectedError_c (*C enumerator*), 541
bleResult_tag.gCsCallbackAlreadyInstalled_c (*C enumerator*), 554
bleResult_tag.gCsSecurityCheckFail_c (*C enumerator*), 554
bleResult_tag.gCsStatusBase_c (*C enumerator*), 553
bleResult_tag.gCsSuccess_c (*C enumerator*), 554
bleResult_tag.gCtrlStatusBase_c (*C enumerator*), 546
bleResult_tag.gCtrlSuccess_c (*C enumerator*), 546
bleResult_tag.gDevDbCccdLimitReached_c (*C enumerator*), 553
bleResult_tag.gDevDbCccdNotFound_c (*C enumerator*), 553
bleResult_tag.gDevDbStatusBase_c (*C enumerator*), 553
bleResult_tag.gDevDbSuccess_c (*C enumerator*), 553
bleResult_tag.gGapAdvDataTooLong_c (*C enumerator*), 552
bleResult_tag.gGapAnotherProcedureInProgress_c (*C enumerator*), 553
bleResult_tag.gGapDeviceNotBonded_c (*C enumerator*), 553
bleResult_tag.gGapScanRspDataTooLong_c (*C enumerator*), 553
bleResult_tag.gGapStatusBase_c (*C enumerator*), 552
bleResult_tag.gGapSuccess_c (*C enumerator*), 552
bleResult_tag.gGattAnotherProcedureInProgress_c (*C enumerator*), 551
bleResult_tag.gGattClientConfirmationTimeout_c (*C enumerator*), 552
bleResult_tag.gGattConnectionSecurityRequirementsNotMet_c (*C enumerator*), 552
bleResult_tag.gGattDbCccdNotFound_c (*C enumerator*), 553
bleResult_tag.gGattDbCharacteristicNotFound_c (*C enumerator*), 553
bleResult_tag.gGattDbDescriptorNotFound_c (*C enumerator*), 553
bleResult_tag.gGattDbInvalidHandle_c (*C enumerator*), 553
bleResult_tag.gGattDbServiceNotFound_c (*C enumerator*), 553
bleResult_tag.gGattDbServiceOrCharAlreadyDeclared_c (*C enumerator*), 553
bleResult_tag.gGattDbStatusBase_c (*C enumerator*), 553
bleResult_tag.gGattDbSuccess_c (*C enumerator*), 553
bleResult_tag.gGattIndicationAlreadyInProgress_c (*C enumerator*), 552
bleResult_tag.gGattInvalidPduReceived_c (*C enumerator*), 552
bleResult_tag.gGattInvalidValueLength_c (*C enumerator*), 552
bleResult_tag.gGattLongAttributePacketsCorrupted_c (*C enumerator*), 552
bleResult_tag.gGattMtuExchangeInProgress_c (*C enumerator*), 552
bleResult_tag.gGattMultipleAttributesOverflow_c (*C enumerator*), 552
bleResult_tag.gGattOutOfSyncProceduresOngoing_c (*C enumerator*), 552
bleResult_tag.gGattPeerDisconnected_c (*C enumerator*), 552
bleResult_tag.gGattServerTimeout_c (*C enumerator*), 552
bleResult_tag.gGattStatusBase_c (*C enumerator*), 551
bleResult_tag.gGattSuccess_c (*C enumerator*), 551
bleResult_tag.gGattUnexpectedReadMultipleResponseLength_c (*C enumerator*), 552
bleResult_tag.gHciAclConnectionAlreadyExists_c (*C enumerator*), 542
bleResult_tag.gHciAlreadyInit_c (*C enumerator*), 546
bleResult_tag.gHciAuthenticationFailure_c (*C enumerator*), 542
bleResult_tag.gHciCallbackAlreadyInstalled_c (*C enumerator*), 546
bleResult_tag.gHciCallbackNotInstalled_c (*C enumerator*), 546
bleResult_tag.gHciChannelClassificationNotSupported_c (*C enumerator*), 544
bleResult_tag.gHciCoarseClockAdjustmentRejected_c (*C enumerator*), 545
bleResult_tag.gHciCommandDisallowed_c (*C enumerator*), 542
bleResult_tag.gHciCommandNotSupported_c (*C enumerator*), 546
bleResult_tag.gHciConnectionAcceptTimeoutExceeded_c (*C enumerator*), 542
bleResult_tag.gHciConnectionFailedToBeEstablishedOrSyncTimeout_c (*C enumerator*), 545
bleResult_tag.gHciConnectionLimitExceeded_c (*C enumerator*), 542
bleResult_tag.gHciConnectionRejectedDueToLimitedResources_c (*C enumerator*), 542
bleResult_tag.gHciConnectionRejectedDueToNoSuitableChannelFound_c (*C enumerator*), 545
bleResult_tag.gHciConnectionRejectedDueToSecurityReasons_c (*C enumerator*), 542

bleResult_tag.gHciConnectionRejectedDueToUnacceptableBdAddr_c (*C enumerator*), 542
bleResult_tag.gHciConnectionTerminatedByLocalHost_c (*C enumerator*), 543
bleResult_tag.gHciConnectionTerminatedDueToMicFailure_c (*C enumerator*), 545
bleResult_tag.gHciConnectionTimeout_c (*C enumerator*), 542
bleResult_tag.gHciControllerBusy_c (*C enumerator*), 545
bleResult_tag.gHciDifferentTransactionCollision_c (*C enumerator*), 544
bleResult_tag.gHciDirectedAdvertisingTimeout_c (*C enumerator*), 545
bleResult_tag.gHciEncryptionModeNotAcceptable_c (*C enumerator*), 544
bleResult_tag.gHciEventNotSupported_c (*C enumerator*), 546
bleResult_tag.gHciExtendedInquiryResponseTooLarge_c (*C enumerator*), 545
bleResult_tag.gHciHardwareFailure_c (*C enumerator*), 541
bleResult_tag.gHciHostBusyPairing_c (*C enumerator*), 545
bleResult_tag.gHciInstantPassed_c (*C enumerator*), 544
bleResult_tag.gHciInsufficientSecurity_c (*C enumerator*), 544
bleResult_tag.gHciInvalidHciCommandParameters_c (*C enumerator*), 542
bleResult_tag.gHciInvalidLpmParameters_c (*C enumerator*), 543
bleResult_tag.gHciInvalidParameter_c (*C enumerator*), 546
bleResult_tag.gHciLimitReached_c (*C enumerator*), 546
bleResult_tag.gHciLinkKeyCannotBeChanged_c (*C enumerator*), 544
bleResult_tag.gHciLLResponseTimeout_c (*C enumerator*), 543
bleResult_tag.gHciLmpErrorTransactionCollision_c (*C enumerator*), 543
bleResult_tag.gHciLmpPduNotAllowed_c (*C enumerator*), 544
bleResult_tag.gHciMacConnectionFailed_c (*C enumerator*), 545
bleResult_tag.gHciMemoryCapacityExceeded_c (*C enumerator*), 542
bleResult_tag.gHciOperationCancelledByHost_c (*C enumerator*), 546
bleResult_tag.gHciPacketTooEarly_c (*C enumerator*), 546
bleResult_tag.gHciPacketTooLate_c (*C enumerator*), 546
bleResult_tag.gHciPacketTooLong_c (*C enumerator*), 546
bleResult_tag.gHciPageTimeout_c (*C enumerator*), 541
bleResult_tag.gHciPairingNotAllowed_c (*C enumerator*), 543
bleResult_tag.gHciPairingWithUnitKeyNotSupported_c (*C enumerator*), 544
bleResult_tag.gHciParameterOutOfMandatoryRange_c (*C enumerator*), 544
bleResult_tag.gHciPinOrKeyMissing_c (*C enumerator*), 542
bleResult_tag.gHciQosNotAcceptableParameter_c (*C enumerator*), 544
bleResult_tag.gHciQosRejected_c (*C enumerator*), 544
bleResult_tag.gHciRemoteDeviceTerminatedConnectionLowResources_c (*C enumerator*), 542
bleResult_tag.gHciRemoteDeviceTerminatedConnectionPowerOff_c (*C enumerator*), 543
bleResult_tag.gHciRemoteUserTerminatedConnection_c (*C enumerator*), 542
bleResult_tag.gHciRepeatedAttempts_c (*C enumerator*), 543
bleResult_tag.gHciRequestedQosNotSupported_c (*C enumerator*), 544
bleResult_tag.gHciReserved_0x2B_c (*C enumerator*), 544
bleResult_tag.gHciReserved_0x31_c (*C enumerator*), 544
bleResult_tag.gHciReserved_0x33_c (*C enumerator*), 544
bleResult_tag.gHciReservedSlotViolation_c (*C enumerator*), 545
bleResult_tag.gHciRoleChangeNotAllowed_c (*C enumerator*), 543
bleResult_tag.gHciRoleSwitchFailed_c (*C enumerator*), 545
bleResult_tag.gHciRoleSwitchPending_c (*C enumerator*), 544
bleResult_tag.gHciScoAirModeRejected_c (*C enumerator*), 543
bleResult_tag.gHciScoIntervalRejected_c (*C enumerator*), 543
bleResult_tag.gHciScoOffsetRejected_c (*C enumerator*), 543
bleResult_tag.gHciSecureSimplePairingNotSupportedByHost_c (*C enumerator*), 545
bleResult_tag.gHciStatusBase_c (*C enumerator*), 541
bleResult_tag.gHciSuccess_c (*C enumerator*), 541
bleResult_tag.gHciSynchronousConnectionLimitToADeviceExceeded_c (*C enumerator*), 542
bleResult_tag.gHciTransportError_c (*C enumerator*), 546
bleResult_tag.gHciType0SubmapNotDefined_c (*C enumerator*), 545
bleResult_tag.gHciUnacceptableConnectionParameters_c (*C enumerator*), 545
bleResult_tag.gHciUnknownAdvertisingIdentifier_c (*C enumerator*), 546

bleResult_tag.gHciUnknownConnectionIdentifier_c (*C enumerator*), 541
bleResult_tag.gHciUnknownHciCommand_c (*C enumerator*), 541
bleResult_tag.gHciUnknownLpmPdu_c (*C enumerator*), 543
bleResult_tag.gHciUnspecifiedError_c (*C enumerator*), 543
bleResult_tag.gHciUnsupportedFeatureOrParameterValue_c (*C enumerator*), 542
bleResult_tag.gHciUnsupportedLpmParameterValue_c (*C enumerator*), 543
bleResult_tag.gHciUnsupportedRemoteFeature_c (*C enumerator*), 543
bleResult_tag.gL2caAlreadyInit_c (*C enumerator*), 547
bleResult_tag.gL2caCallbackAlreadyInstalled_c (*C enumerator*), 547
bleResult_tag.gL2caCallbackNotInstalled_c (*C enumerator*), 547
bleResult_tag.gL2caChannelAlreadyConnected_c (*C enumerator*), 547
bleResult_tag.gL2caChannelBusy_c (*C enumerator*), 547
bleResult_tag.gL2caChannelClosed_c (*C enumerator*), 547
bleResult_tag.gL2caChannelInvalid_c (*C enumerator*), 547
bleResult_tag.gL2caConnectionParametersRejected_c (*C enumerator*), 547
bleResult_tag.gL2caInsufficientResources_c (*C enumerator*), 547
bleResult_tag.gL2caInternalError_c (*C enumerator*), 548
bleResult_tag.gL2caInvalidParameter_c (*C enumerator*), 547
bleResult_tag.gL2caLePsmAlreadyRegistered_c (*C enumerator*), 547
bleResult_tag.gL2caLePsmInsufficientResources_c (*C enumerator*), 547
bleResult_tag.gL2caLePsmInvalid_c (*C enumerator*), 547
bleResult_tag.gL2caLePsmNotRegistered_c (*C enumerator*), 547
bleResult_tag.gL2caStatusBase_c (*C enumerator*), 547
bleResult_tag.gL2caSuccess_c (*C enumerator*), 547
bleResult_tag.gSmCommandNotSupported_c (*C enumerator*), 548
bleResult_tag.gSmInproperKeyDistributionField_c (*C enumerator*), 548
bleResult_tag.gSmInsufficientResources_c (*C enumerator*), 549
bleResult_tag.gSmInvalidCommandCode_c (*C enumerator*), 548
bleResult_tag.gSmInvalidCommandLength_c (*C enumerator*), 548
bleResult_tag.gSmInvalidCommandParameter_c (*C enumerator*), 548
bleResult_tag.gSmInvalidConnectionHandle_c (*C enumerator*), 548
bleResult_tag.gSmInvalidDeviceId_c (*C enumerator*), 548
bleResult_tag.gSmInvalidHciEventParameter_c (*C enumerator*), 549
bleResult_tag.gSmInvalidInternalOperation_c (*C enumerator*), 548
bleResult_tag.gSmInvalidPeerPublicKey_c (*C enumerator*), 550
bleResult_tag.gSmInvalidSmpPacketLength_c (*C enumerator*), 549
bleResult_tag.gSmInvalidSmpPacketParameter_c (*C enumerator*), 549
bleResult_tag.gSmKeySessionKeyDerivationFailed_c (*C enumerator*), 551
bleResult_tag.gSmLlConnectionEncryptionFailure_c (*C enumerator*), 549
bleResult_tag.gSmLlConnectionEncryptionInProgress_c (*C enumerator*), 549
bleResult_tag.gSmNullCBFunction_c (*C enumerator*), 548
bleResult_tag.gSmOobDataAddressMismatch_c (*C enumerator*), 549
bleResult_tag.gSmPairingAlreadyStarted_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorAuthenticationRequirements_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorBusy_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorCommandNotSupported_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorConfirmValueFailed_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorDhKeyCheckFailed_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorEncryptionKeySize_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorInvalidParameters_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorKeyRejected_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorNumericComparisonFailed_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorOobNotAvailable_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorPairingNotSupported_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorPasskeyEntryFailed_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorRepeatedAttempts_c (*C enumerator*), 551
bleResult_tag.gSmPairingErrorTimeout_c (*C enumerator*), 550
bleResult_tag.gSmPairingErrorUnknownReason_c (*C enumerator*), 550

bleResult_tag.gSmReceivedHciEventFromUnknownDevice_c (*C enumerator*), 549
bleResult_tag.gSmReceivedSmpPacketFromUnknownDevice_c (*C enumerator*), 549
bleResult_tag.gSmReceivedTimerEventForUnknownDevice_c (*C enumerator*), 550
bleResult_tag.gSmReceivedUnexpectedHciEvent_c (*C enumerator*), 549
bleResult_tag.gSmReceivedUnexpectedSmpPacket_c (*C enumerator*), 549
bleResult_tag.gSmSmpPacketReceivedAfterTimeoutOccurred_c (*C enumerator*), 550
bleResult_tag.gSmSmpTimeoutOccurred_c (*C enumerator*), 549
bleResult_tag.gSmStatusBase_c (*C enumerator*), 548
bleResult_tag.gSmSuccess_c (*C enumerator*), 548
bleResult_tag.gSmTbInvalidDataSignature_c (*C enumerator*), 551
bleResult_tag.gSmTbResolvableAddressDoesNotMatchIrk_c (*C enumerator*), 551
bleResult_tag.gSmUnattainableLocalDeviceMinKeySize_c (*C enumerator*), 550
bleResult_tag.gSmUnattainableLocalDeviceSecRequirements_c (*C enumerator*), 550
bleResult_tag.gSmUnattainablePeripheralSecReqRequirements_c (*C enumerator*), 550
bleResult_tag.gSmUnexpectedCommand_c (*C enumerator*), 548
bleResult_tag.gSmUnexpectedKeyset_c (*C enumerator*), 549
bleResult_tag.gSmUnexpectedKeyType_c (*C enumerator*), 548
bleResult_tag.gSmUnexpectedPairingTerminationReason_c (*C enumerator*), 548
bleResult_tag.gSmUnknownSmpPacketType_c (*C enumerator*), 549
bleScanningFilterPolicy_t (*C type*), 582
bleScanType_t (*C enum*), 558
bleScanType_t.gScanTypeActive_c (*C enumerator*), 558
bleScanType_t.gScanTypePassive_c (*C enumerator*), 558
BleSig_IsGroupingAttributeUuid16 (*C macro*), 600
BleSig_IsServiceDeclarationUuid16 (*C macro*), 600
bleSlotDurations_t (*C type*), 585
bleSlotDurations_tag (*C enum*), 579
bleSlotDurations_tag.gSlotDurations1us_c (*C enumerator*), 579
bleSlotDurations_tag.gSlotDurations2us_c (*C enumerator*), 579
bleSupportedSwitchingSamplingRates_t (*C struct*), 604, 638
bleSupportedSwitchingSamplingRates_t.reserved (*C var*), 604, 638
bleSupportedSwitchingSamplingRates_t.samplingSupportedAodReception (*C var*), 604, 638
bleSupportedSwitchingSamplingRates_t.switchingSamplingSupportedAoaReception (*C var*), 604, 638
bleSupportedSwitchingSamplingRates_t.switchingSupportedAodTransmission (*C var*), 604, 638
bleSyncCteType_t (*C type*), 585
bleSyncCteType_tag (*C struct*), 616, 650
bleSyncCteType_tag.doNotSyncWithAoA (*C var*), 617, 650
bleSyncCteType_tag.doNotSyncWithAoD1us (*C var*), 617, 650
bleSyncCteType_tag.doNotSyncWithAoD2us (*C var*), 617, 650
bleSyncCteType_tag.doNotSyncWithoutCte (*C var*), 617, 650
bleSyncCteType_tag.doNotSyncWithType3 (*C var*), 617, 650
bleSyncCteType_tag.reserved (*C var*), 617, 650
bleTransmitPowerChannelType_t (*C enum*), 558
bleTransmitPowerChannelType_t.gTxPowerAdvChannel_c (*C enumerator*), 558
bleTransmitPowerChannelType_t.gTxPowerConnChannel_c (*C enumerator*), 558
bleTransmitPowerLevelType_t (*C enum*), 558
bleTransmitPowerLevelType_t.gReadCurrentTxPowerLevel_c (*C enumerator*), 558
bleTransmitPowerLevelType_t.gReadMaximumTxPowerLevel_c (*C enumerator*), 558
bleTxPowerLevelFlags_t (*C struct*), 617, 650
bleTxPowerLevelFlags_t.maximum (*C var*), 617, 650
bleTxPowerLevelFlags_t.minimum (*C var*), 617, 650
bleTxPowerLevelFlags_t.reserved (*C var*), 617, 650
bleTxPowerReportingEnable_t (*C enum*), 580
bleTxPowerReportingEnable_t.gTxPowerReportingDisable_c (*C enumerator*), 580
bleTxPowerReportingEnable_t.gTxPowerReportingEnable_c (*C enumerator*), 580
bleTxPowerReportingReason_t (*C type*), 585
bleTxPowerReportingReason_tag (*C enum*), 580
bleTxPowerReportingReason_tag.gLocalTxPowerChanged_c (*C enumerator*), 580

bleTxPowerReportingReason_tag.gReadRemoteTxPowerLevelCommandCompleted_c (*C enumerator*), 580
bleTxPowerReportingReason_tag.gRemoteTxPowerChanged_c (*C enumerator*), 580
bleTxTestPacketPayload_t (*C enum*), 560
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern00001111_c (*C enumerator*), 561
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern01010101_c (*C enumerator*), 561
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern10101010_c (*C enumerator*), 560
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern11110000_c (*C enumerator*), 560
bleTxTestPacketPayload_t.gBleTestPacketPayloadPatternAllBits0_c (*C enumerator*), 561
bleTxTestPacketPayload_t.gBleTestPacketPayloadPatternAllBits1_c (*C enumerator*), 560
bleTxTestPacketPayload_t.gBleTestPacketPayloadPrbs9_c (*C enumerator*), 560
bleTxTestPacketPayload_t.gBleTestPacketPayloadPrbs15_c (*C enumerator*), 560
bleUuid_t (*C union*), 601, 635
bleUuid_t.uuid16 (*C var*), 601, 635
bleUuid_t.uuid32 (*C var*), 601, 635
bleUuid_t.uuid128 (*C var*), 601, 635
bleUuidType_t (*C type*), 582

D

deviceId_t (*C type*), 582

G

gAdvDataChange_c (*C macro*), 588
gaGattDynamicAttrBlob (*C var*), 533
gaGattDynamicValBlob (*C var*), 534
Gap_AcceptPairingRequest (*C function*), 397
Gap_AddDeviceToFilterAcceptList (*C function*), 411
Gap_AddDeviceToMonAdvList (*C function*), 404
Gap_AddSecurityModesAndLevels (*C macro*), 445
Gap_Authorize (*C function*), 392
Gap_BleAdvIndexChange (*C function*), 423
Gap_CancelInitiatingConnection (*C macro*), 446
Gap_CheckIfBonded (*C function*), 410
Gap_CheckIfConnected (*C function*), 410
Gap_CheckIndicationStatus (*C function*), 394
Gap_CheckNotificationStatus (*C function*), 394
Gap_CheckNvmIndex (*C function*), 410
Gap_ClearFilterAcceptList (*C function*), 411
Gap_ClearMonAdvList (*C function*), 405
Gap_Connect (*C function*), 406
Gap_ConnectFromPawr (*C function*), 407
Gap_ControllerEnhancedNotification (*C function*), 423
Gap_ControllerTest (*C function*), 421
Gap_CreateRandomDeviceAddress (*C function*), 412
Gap_DecryptAdvertisingData (*C function*), 445
Gap_DenyLongTermKey (*C function*), 401
Gap_Disconnect (*C function*), 407
Gap_EattConnectionAccept (*C function*), 440
Gap_EattConnectionRequest (*C function*), 440
Gap_EattDisconnect (*C function*), 441
Gap_EattReconfigureRequest (*C function*), 441
Gap_EattSendCredits (*C function*), 441
Gap_EnableConnectionCteRequest (*C function*), 437
Gap_EnableConnectionCteResponse (*C function*), 437
Gap_EnableConnectionlessCteTransmit (*C function*), 436
Gap_EnableConnectionlessIqSampling (*C function*), 436
Gap_EnableControllerPrivacy (*C function*), 420
Gap_EnableHostPrivacy (*C function*), 419

Gap_EnableMonAdv (*C function*), 405
Gap_EnablePathLossReporting (*C function*), 439
Gap_EnableTransmitPowerReporting (*C function*), 439
Gap_EnableUpdateConnectionParameters (*C function*), 418
Gap_EncryptAdvertisingData (*C function*), 444
Gap_EncryptLink (*C function*), 397
Gap_EnhancedReadTransmitPowerLevel (*C function*), 438
Gap_EnterPasskey (*C function*), 398
Gap_GenerateDhKeyV2 (*C function*), 439
Gap_GetBondedDeviceName (*C function*), 413
Gap_GetBondedDevicesCount (*C function*), 413
Gap_GetBondedDevicesIdentityInformation (*C function*), 395
Gap_GetConnectionHandleFromDeviceId (*C function*), 442
Gap_GetConnParams (*C macro*), 447
Gap_GetConnParamsMonitoring (*C function*), 442
Gap_GetDeviceIdFromConnHandle (*C function*), 442
Gap_GetHostVersion (*C function*), 442
Gap_InitMonitoringAdvertisers (*C function*), 404
Gap_LeChannelOverride (*C function*), 443
Gap_LePeriodicAdvUpdateSync (*C function*), 444
Gap_LeReadPhy (*C function*), 422
Gap_LeScGetLocalOobData (*C function*), 400
Gap_LeScRegeneratePublicKey (*C function*), 399
Gap_LeScSendKeypressNotification (*C function*), 400
Gap_LeScSetPeerOobData (*C function*), 400
Gap_LeScValidateNumericValue (*C function*), 399
Gap_LeSetHostFeature (*C function*), 444
Gap_LeSetPhy (*C function*), 422
Gap_LeSetSchedulerPriority (*C function*), 444
Gap_LoadCustomBondedDeviceInformation (*C function*), 409
Gap_LoadCustomPeerInformation (*C function*), 408
Gap_LoadEncryptionInformation (*C function*), 402
Gap_LoadKeys (*C function*), 423
Gap_ModifySleepClockAccuracy (*C function*), 421
Gap_Pair (*C function*), 396
Gap_PeriodicAdvCreateSync (*C function*), 433
Gap_PeriodicAdvReceiveDisable (*C function*), 434
Gap_PeriodicAdvReceiveEnable (*C function*), 433
Gap_PeriodicAdvSetInfoTransfer (*C function*), 434
Gap_PeriodicAdvSyncTransfer (*C function*), 434
Gap_PeriodicAdvTerminateSync (*C function*), 433
Gap_ProvideLongTermKey (*C function*), 401
Gap_ProvideOob (*C function*), 398
Gap_ReadAdvertisingTxPowerLevel (*C macro*), 446
Gap_ReadAntennaInformation (*C function*), 437
Gap_ReadChannelMap (*C function*), 425
Gap_ReadControllerLocalRPA (*C function*), 417
Gap_ReadFilterAcceptListSize (*C function*), 411
Gap_ReadMonAdvListSize (*C function*), 405
Gap_ReadPublicDeviceAddress (*C function*), 412
Gap_ReadRadioPowerLevel (*C function*), 415
Gap_ReadRemoteTransmitPowerLevel (*C function*), 438
Gap_ReadRemoteVersionInformation (*C function*), 442
Gap_ReadRssi (*C macro*), 446
Gap_ReadTxPowerLevelInConnection (*C macro*), 447
Gap_RegisterDeviceSecurityRequirements (*C function*), 390
Gap_RejectKeyExchangeRequest (*C function*), 399
Gap_RejectPairing (*C function*), 397

Gap_RejectPasskeyRequest (*C function*), 398
Gap_RemoveAdvSet (*C function*), 429
Gap_RemoveAllBonds (*C function*), 415
Gap_RemoveBond (*C function*), 414
Gap_RemoveDeviceFromFilterAcceptList (*C function*), 411
Gap_RemoveDeviceFromMonAdvList (*C function*), 405
Gap_ResumeLeScStateMachine (*C function*), 435
Gap_SaveCccd (*C function*), 393
Gap_SaveCustomPeerInformation (*C function*), 408
Gap_SaveDeviceName (*C function*), 412
Gap_SaveKeys (*C function*), 424
Gap_SendPeripheralSecurityRequest (*C function*), 396
Gap_SendSmpKeys (*C function*), 399
Gap_SetAdvertisingData (*C function*), 391
Gap_SetAdvertisingParameters (*C function*), 390
Gap_SetBondedDeviceName (*C function*), 414
Gap_SetChannelMap (*C function*), 425
Gap_SetConnectionCallback (*C function*), 445
Gap_SetConnectionCteReceiveParameters (*C function*), 436
Gap_SetConnectionCteTransmitParameters (*C function*), 437
Gap_SetConnectionlessCteTransmitParameters (*C function*), 435
Gap_SetDataRelatedAddressChanges (*C function*), 390
Gap_SetDecisionInstructions (*C function*), 404
Gap_SetDefaultPairingParameters (*C function*), 417
Gap_SetDefaultPeriodicAdvSyncTransferParams (*C function*), 435
Gap_SetExtAdvertisingData (*C function*), 426
Gap_SetExtAdvertisingDecisionData (*C function*), 427
Gap_SetExtAdvertisingParameters (*C function*), 426
Gap_SetLocalPasskey (*C function*), 402
Gap_SetPathLossReportingParameters (*C function*), 438
Gap_SetPeriodicAdvertisingData (*C function*), 431
Gap_SetPeriodicAdvParameters (*C function*), 429
Gap_SetPeriodicAdvResponseData (*C function*), 430
Gap_SetPeriodicAdvSubeventData (*C function*), 430
Gap_SetPeriodicAdvSyncTransferParams (*C function*), 435
Gap_SetPeriodicSyncSubevent (*C function*), 430
Gap_SetPrivacyMode (*C function*), 420
Gap_SetRandomAddress (*C function*), 417
Gap_SetScanMode (*C function*), 403
Gap_SetScanningCallback (*C function*), 445
Gap_SetTxPowerLevel (*C function*), 416
Gap_StartAdvertising (*C function*), 391
Gap_StartExtAdvertising (*C function*), 428
Gap_StartPeriodicAdvertising (*C function*), 431
Gap_StartScanning (*C function*), 405
Gap_StopAdvertising (*C function*), 392
Gap_StopExtAdvertising (*C function*), 428
Gap_StopPeriodicAdvertising (*C function*), 432
Gap_StopScanning (*C function*), 406
Gap_UpdateConnectionParameters (*C function*), 418
Gap_UpdateLeDataLength (*C function*), 419
Gap_UpdatePeriodicAdvList (*C function*), 432
Gap_VerifyPrivateResolvableAddress (*C function*), 416
gapAddrReadyEvent_t (*C type*), 583
gapAddrReadyEvent_t_tag (*C struct*), 604, 638
gapAddrReadyEvent_t_tag.aAddress (*C var*), 604, 638
gapAddrReadyEvent_t_tag.advHandle (*C var*), 604, 638
gapAdStructure_t (*C struct*), 467, 663

gapAdStructure_t.aData (*C var*), 467, 663
gapAdStructure_t.adType (*C var*), 467, 663
gapAdStructure_t.length (*C var*), 467, 663
gapAdType_t (*C enum*), 374
gapAdType_t.gAd3dInformationData_c (*C enumerator*), 376
gapAdType_t.gAdAdvertisingInterval_c (*C enumerator*), 376
gapAdType_t.gAdAdvertisingIntervalLong_c (*C enumerator*), 376
gapAdType_t.gAdAppearance_c (*C enumerator*), 375
gapAdType_t.gAdChannelMapUpdateIndication_c (*C enumerator*), 376
gapAdType_t.gAdClassOfDevice_c (*C enumerator*), 375
gapAdType_t.gAdComplete16bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdComplete32bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdComplete128bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdCompleteLocalName_c (*C enumerator*), 374
gapAdType_t.gAdEncryptedAdvertisingData_c (*C enumerator*), 376
gapAdType_t.gAdFlags_c (*C enumerator*), 374
gapAdType_t.gAdIncomplete16bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdIncomplete32bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdIncomplete128bitServiceList_c (*C enumerator*), 374
gapAdType_t.gAdLeDeviceAddress_c (*C enumerator*), 376
gapAdType_t.gAdLeRole_c (*C enumerator*), 376
gapAdType_t.gAdLeSupportedFeatures_c (*C enumerator*), 376
gapAdType_t.gAdManufacturerSpecificData_c (*C enumerator*), 376
gapAdType_t.gAdPeripheralConnectionIntervalRange_c (*C enumerator*), 375
gapAdType_t.gAdPublicTargetAddress_c (*C enumerator*), 375
gapAdType_t.gAdRandomTargetAddress_c (*C enumerator*), 375
gapAdType_t.gAdSecurityManagerOobFlags_c (*C enumerator*), 375
gapAdType_t.gAdSecurityManagerTkValue_c (*C enumerator*), 375
gapAdType_t.gAdServiceData16bit_c (*C enumerator*), 375
gapAdType_t.gAdServiceData32bit_c (*C enumerator*), 375
gapAdType_t.gAdServiceData128bit_c (*C enumerator*), 375
gapAdType_t.gAdServiceSolicitationList16bit_c (*C enumerator*), 375
gapAdType_t.gAdServiceSolicitationList32bit_c (*C enumerator*), 375
gapAdType_t.gAdServiceSolicitationList128bit_c (*C enumerator*), 375
gapAdType_t.gAdShortenedLocalName_c (*C enumerator*), 374
gapAdType_t.gAdSimplePairingHashC192_c (*C enumerator*), 375
gapAdType_t.gAdSimplePairingHashC256_c (*C enumerator*), 376
gapAdType_t.gAdSimplePairingRandomizerR192_c (*C enumerator*), 375
gapAdType_t.gAdSimplePairingRandomizerR256_c (*C enumerator*), 376
gapAdType_t.gAdTxPowerLevel_c (*C enumerator*), 375
gapAdType_t.gAdUniformResourceIdentifier_c (*C enumerator*), 376
gapAdTypeFlags_t (*C type*), 389
gapAdvertisingCallback_t (*C type*), 389
gapAdvertisingChannelMapFlags_t (*C enum*), 372
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag37_c (*C enumerator*), 372
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag38_c (*C enumerator*), 372
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag39_c (*C enumerator*), 372
gapAdvertisingData_t (*C struct*), 468, 663
gapAdvertisingData_t.aAdStructures (*C var*), 468, 663
gapAdvertisingData_t.cNumAdStructures (*C var*), 468, 663
gapAdvertisingDecisionData_t (*C struct*), 468, 663
gapAdvertisingDecisionData_t.dataLength (*C var*), 468, 664
gapAdvertisingDecisionData_t.pDecisionData (*C var*), 468, 663
gapAdvertisingDecisionData_t.pKey (*C var*), 468, 663
gapAdvertisingDecisionData_t.pPrand (*C var*), 468, 663
gapAdvertisingDecisionData_t.resolvableTagPresent (*C var*), 468, 664
gapAdvertisingEvent_t (*C struct*), 471, 666
gapAdvertisingEvent_t.eventData (*C union*), 485, 678

gapAdvertisingEvent_t.eventData (*C var*), 471, 666
gapAdvertisingEvent_t.eventData.advHandle (*C var*), 485, 678
gapAdvertisingEvent_t.eventData.advSetTerminated (*C var*), 485, 678
gapAdvertisingEvent_t.eventData.failReason (*C var*), 485, 678
gapAdvertisingEvent_t.eventData.perAdvResponse (*C var*), 485, 678
gapAdvertisingEvent_t.eventData.scanNotification (*C var*), 485, 678
gapAdvertisingEvent_t.eventData.subeventDataRequest (*C var*), 485, 678
gapAdvertisingEvent_t.eventType (*C var*), 471, 666
gapAdvertisingEventType_t (*C enum*), 380
gapAdvertisingEventType_t.gAdvertisingCommandFailed_c (*C enumerator*), 380
gapAdvertisingEventType_t.gAdvertisingSetTerminated_c (*C enumerator*), 380
gapAdvertisingEventType_t.gAdvertisingStateChanged_c (*C enumerator*), 380
gapAdvertisingEventType_t.gExtAdvertisingStateChanged_c (*C enumerator*), 380
gapAdvertisingEventType_t.gExtScanNotification_c (*C enumerator*), 380
gapAdvertisingEventType_t.gPerAdvResponse_c (*C enumerator*), 380
gapAdvertisingEventType_t.gPerAdvSubeventDataRequest_c (*C enumerator*), 380
gapAdvertisingFilterPolicy_t (*C enum*), 372
gapAdvertisingFilterPolicy_t.gProcessAll_c (*C enumerator*), 373
gapAdvertisingFilterPolicy_t.gProcessConnAllScanWL_c (*C enumerator*), 373
gapAdvertisingFilterPolicy_t.gProcessFilterAcceptListOnly_c (*C enumerator*), 373
gapAdvertisingFilterPolicy_t.gProcessScanAllConnWL_c (*C enumerator*), 373
gapAdvertisingParameters_t (*C struct*), 456, 655
gapAdvertisingParameters_t.advertisingType (*C var*), 457, 656
gapAdvertisingParameters_t.channelMap (*C var*), 457, 656
gapAdvertisingParameters_t.filterPolicy (*C var*), 457, 656
gapAdvertisingParameters_t.maxInterval (*C var*), 456, 656
gapAdvertisingParameters_t.minInterval (*C var*), 456, 656
gapAdvertisingParameters_t.ownAddressType (*C var*), 457, 656
gapAdvertisingParameters_t.peerAddress (*C var*), 457, 656
gapAdvertisingParameters_t.peerAddressType (*C var*), 457, 656
gapAdvertisingSetTerminated_t (*C struct*), 470, 665
gapAdvertisingSetTerminated_t.deviceId (*C var*), 470, 665
gapAdvertisingSetTerminated_t.handle (*C var*), 470, 665
gapAdvertisingSetTerminated_t.numCompletedExtAdvEvents (*C var*), 470, 665
gapAdvertisingSetTerminated_t.status (*C var*), 470, 665
gapAppearance_t (*C type*), 390
gapAppearance_tag (*C enum*), 386
gapAppearance_tag.gBarcodeScanner_c (*C enumerator*), 387
gapAppearance_tag.gBloodPressureArm_c (*C enumerator*), 387
gapAppearance_tag.gBloodPressureWrist_c (*C enumerator*), 387
gapAppearance_tag.gCardReader_c (*C enumerator*), 387
gapAppearance_tag.gCyclingCadenceSensor_c (*C enumerator*), 387
gapAppearance_tag.gCyclingComputer_c (*C enumerator*), 387
gapAppearance_tag.gCyclingPowerSensor_c (*C enumerator*), 387
gapAppearance_tag.gCyclingSpeedandCadenceSensor_c (*C enumerator*), 388
gapAppearance_tag.gCyclingSpeedSensor_c (*C enumerator*), 387
gapAppearance_tag.gDigitalPen_c (*C enumerator*), 387
gapAppearance_tag.gDigitizerTablet_c (*C enumerator*), 387
gapAppearance_tag.gFingertip_c (*C enumerator*), 388
gapAppearance_tag.gGamepad_c (*C enumerator*), 387
gapAppearance_tag.gGenericBarcodeScanner_c (*C enumerator*), 386
gapAppearance_tag.gGenericBloodPressure_c (*C enumerator*), 387
gapAppearance_tag.gGenericClock_c (*C enumerator*), 386
gapAppearance_tag.gGenericComputer_c (*C enumerator*), 386
gapAppearance_tag.gGenericCycling_c (*C enumerator*), 387
gapAppearance_tag.gGenericDisplay_c (*C enumerator*), 386
gapAppearance_tag.gGenericEyeglasses_c (*C enumerator*), 386
gapAppearance_tag.gGenericGlucoseMeter_c (*C enumerator*), 387

gapAppearance_tag.gGenericHeartrateSensor_c (*C enumerator*), 386
gapAppearance_tag.gGenericKeyring_c (*C enumerator*), 386
gapAppearance_tag.gGenericMediaPlayer_c (*C enumerator*), 386
gapAppearance_tag.gGenericOutdoorSportsActivity_c (*C enumerator*), 388
gapAppearance_tag.gGenericPhone_c (*C enumerator*), 386
gapAppearance_tag.gGenericPulseOximeter_c (*C enumerator*), 388
gapAppearance_tag.gGenericRemoteControl_c (*C enumerator*), 386
gapAppearance_tag.gGenericRunningWalkingSensor_c (*C enumerator*), 387
gapAppearance_tag.gGenericTag_c (*C enumerator*), 386
gapAppearance_tag.gGenericThermometer_c (*C enumerator*), 386
gapAppearance_tag.gGenericWatch_c (*C enumerator*), 386
gapAppearance_tag.gGenericWeightScale_c (*C enumerator*), 388
gapAppearance_tag.gHeartRateSensorHeartRateBelt_c (*C enumerator*), 387
gapAppearance_tag.gHumanInterfaceDevice_c (*C enumerator*), 387
gapAppearance_tag.gJoystick_c (*C enumerator*), 387
gapAppearance_tag.gKeyboard_c (*C enumerator*), 387
gapAppearance_tag.gLocationandNavigationDisplayDevice_c (*C enumerator*), 388
gapAppearance_tag.gLocationAndNavigationPod_c (*C enumerator*), 388
gapAppearance_tag.gLocationDisplayDevice_c (*C enumerator*), 388
gapAppearance_tag.gLocationPod_c (*C enumerator*), 388
gapAppearance_tag.gMouse_c (*C enumerator*), 387
gapAppearance_tag.gRunningWalkingSensorInShoe_c (*C enumerator*), 387
gapAppearance_tag.gRunningWalkingSensorOnHip_c (*C enumerator*), 387
gapAppearance_tag.gRunningWalkingSensorOnShoe_c (*C enumerator*), 387
gapAppearance_tag.gSportsWatch_c (*C enumerator*), 386
gapAppearance_tag.gThermometerEar_c (*C enumerator*), 386
gapAppearance_tag.gUnknown_c (*C enumerator*), 386
gapAppearance_tag.gWristWorn_c (*C enumerator*), 388
gapAuthenticationRejectedEvent_t (*C struct*), 478, 671
gapAuthenticationRejectedEvent_t.rejectReason (*C var*), 478, 671
gapAuthenticationRejectReason_t (*C type*), 388
gapAutoConnectParams_t (*C struct*), 484, 676
gapAutoConnectParams_t.aAutoConnectData (*C var*), 484, 676
gapAutoConnectParams_t.cNumAddresses (*C var*), 484, 676
gapAutoConnectParams_t.writeInFilterAcceptList (*C var*), 484, 676
gapCarSupport_t (*C enum*), 385
gapCarSupport_t.CAR_Supported (*C enumerator*), 385
gapCarSupport_t.CAR_Unavailable (*C enumerator*), 385
gapCarSupport_t.CAR_Unknown (*C enumerator*), 385
gapCarSupport_t.CAR_Unsupported (*C enumerator*), 385
gapConnCteRequestFailed_t (*C struct*), 481, 674
gapConnCteRequestFailed_t.status (*C var*), 481, 674
gapConnectedEvent_t (*C struct*), 477, 670
gapConnectedEvent_t.advHandle (*C var*), 477, 671
gapConnectedEvent_t.connectionRole (*C var*), 477, 671
gapConnectedEvent_t.connParameters (*C var*), 477, 671
gapConnectedEvent_t.localRpa (*C var*), 477, 671
gapConnectedEvent_t.localRpaUsed (*C var*), 477, 671
gapConnectedEvent_t.peerAddress (*C var*), 477, 671
gapConnectedEvent_t.peerAddressType (*C var*), 477, 671
gapConnectedEvent_t.peerRpa (*C var*), 477, 671
gapConnectedEvent_t.peerRpaResolved (*C var*), 477, 671
gapConnectedEvent_t.syncHandle (*C var*), 477, 671
gapConnectionCallback_t (*C type*), 389
gapConnectionCteReceiveParams_t (*C struct*), 466, 662
gapConnectionCteReceiveParams_t.aAntennaIds (*C var*), 466, 662
gapConnectionCteReceiveParams_t.iqSamplingEnable (*C var*), 466, 662
gapConnectionCteReceiveParams_t.slotDurations (*C var*), 466, 662

gapConnectionCteReceiveParams_t.switchingPatternLength (*C var*), 466, 662
gapConnectionCteReqEnableParams_t (*C struct*), 466, 662
gapConnectionCteReqEnableParams_t.cteReqEnable (*C var*), 467, 662
gapConnectionCteReqEnableParams_t.cteReqInterval (*C var*), 467, 662
gapConnectionCteReqEnableParams_t.requestedCteLength (*C var*), 467, 662
gapConnectionCteReqEnableParams_t.requestedCteType (*C var*), 467, 662
gapConnectionCteTransmitParams_t (*C struct*), 466, 662
gapConnectionCteTransmitParams_t.aAntennaIds (*C var*), 466, 662
gapConnectionCteTransmitParams_t.cteTypes (*C var*), 466, 662
gapConnectionCteTransmitParams_t.switchingPatternLength (*C var*), 466, 662
gapConnectionEvent_t (*C struct*), 483, 676
gapConnectionEvent_t.eventData (*C union*), 486, 679
gapConnectionEvent_t.eventData (*C var*), 483, 676
gapConnectionEvent_t.eventData.authenticationRejectedEvent (*C var*), 486, 679
gapConnectionEvent_t.eventData.channelMap (*C var*), 487, 680
gapConnectionEvent_t.eventData.connectedEvent (*C var*), 486, 679
gapConnectionEvent_t.eventData.connectionUpdateComplete (*C var*), 487, 680
gapConnectionEvent_t.eventData.connectionUpdateRequest (*C var*), 487, 680
gapConnectionEvent_t.eventData.connIqReport (*C var*), 487, 680
gapConnectionEvent_t.eventData.cteRequestFailedEvent (*C var*), 487, 680
gapConnectionEvent_t.eventData.disconnectedEvent (*C var*), 487, 679
gapConnectionEvent_t.eventData.eattBearerStatusNotification (*C var*), 488, 680
gapConnectionEvent_t.eventData.eattConnectionComplete (*C var*), 488, 680
gapConnectionEvent_t.eventData.eattConnectionRequest (*C var*), 488, 680
gapConnectionEvent_t.eventData.eattReconfigureResponse (*C var*), 488, 680
gapConnectionEvent_t.eventData.encryptionChangedEvent (*C var*), 487, 679
gapConnectionEvent_t.eventData.failReason (*C var*), 487, 680
gapConnectionEvent_t.eventData.handoverConnectedEvent (*C var*), 488, 680
gapConnectionEvent_t.eventData.handoverDisconnectedEvent (*C var*), 488, 680
gapConnectionEvent_t.eventData.incomingKeypressNotification (*C var*), 487, 680
gapConnectionEvent_t.eventData.keyExchangeRequestEvent (*C var*), 486, 679
gapConnectionEvent_t.eventData.keysReceivedEvent (*C var*), 487, 679
gapConnectionEvent_t.eventData.leDataLengthChanged (*C var*), 487, 680
gapConnectionEvent_t.eventData.longTermKeyRequestEvent (*C var*), 487, 679
gapConnectionEvent_t.eventData.numericValueForDisplay (*C var*), 487, 680
gapConnectionEvent_t.eventData.pairingCompleteEvent (*C var*), 487, 679
gapConnectionEvent_t.eventData.pairingEvent (*C var*), 486, 679
gapConnectionEvent_t.eventData.passkeyForDisplay (*C var*), 487, 680
gapConnectionEvent_t.eventData.pathLossThreshold (*C var*), 488, 680
gapConnectionEvent_t.eventData.perAdvSyncTransferStatus (*C var*), 488, 680
gapConnectionEvent_t.eventData.peripheralSecurityRequestEvent (*C var*), 486, 679
gapConnectionEvent_t.eventData.rssi_dBm (*C var*), 487, 680
gapConnectionEvent_t.eventData.smError (*C var*), 488, 680
gapConnectionEvent_t.eventData.transmitPowerInfo (*C var*), 488, 680
gapConnectionEvent_t.eventData.transmitPowerReporting (*C var*), 488, 680
gapConnectionEvent_t.eventData.txPowerLevel_dBm (*C var*), 487, 680
gapConnectionEvent_t.eventType (*C var*), 483, 676
gapConnectionEventType_t (*C enum*), 381
gapConnectionEventType_t.gConnEvtAuthenticationRejected_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtChannelMapRead_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtChannelMapReadFailure_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtChanSelectionAlgorithm2_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtConnected_c (*C enumerator*), 381
gapConnectionEventType_t.gConnEvtCteReceiveParamsSetupComplete_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtCteReqStateChanged_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtCteRequestFailed_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtCteRspStateChanged_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtCteTransmitParamsSetupComplete_c (*C enumerator*), 384

gapConnectionEventType_t.gConnEvtDisconnected_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtEattBearerStatusNotification_c (*C enumerator*), 385
gapConnectionEventType_t.gConnEvtEattChannelReconfigureResponse_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtEattConnectionComplete_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtEattConnectionRequest_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtEncryptionChanged_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtEnhancedReadTransmitPowerLevel_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtHandoverConnected_c (*C enumerator*), 385
gapConnectionEventType_t.gConnEvtIqReportReceived_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtKeyExchangeRequest_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtKeysReceived_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtLeDataLengthChanged_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtLeScDisplayNumericValue_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtLeScKeypressNotification_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtLeScOobDataRequest_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtLeSetDataLengthFailure_c (*C enumerator*), 385
gapConnectionEventType_t.gConnEvtLongTermKeyRequest_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtOobRequest_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtPairingAlreadyStarted_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtPairingComplete_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtPairingNoLtk_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtPairingRequest_c (*C enumerator*), 381
gapConnectionEventType_t.gConnEvtPairingResponse_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtParameterUpdateComplete_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtParameterUpdateRequest_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtPasskeyDisplay_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtPasskeyRequest_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtPathLossReportingParamsSetupComplete_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtPathLossReportingStateChanged_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtPathLossThreshold_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtPeripheralSecurityRequest_c (*C enumerator*), 382
gapConnectionEventType_t.gConnEvtPowerReadFailure_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtRssiRead_c (*C enumerator*), 383
gapConnectionEventType_t.gConnEvtSmError_c (*C enumerator*), 385
gapConnectionEventType_t.gConnEvtTransmitPowerReporting_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtTransmitPowerReportingStateChanged_c (*C enumerator*), 384
gapConnectionEventType_t.gConnEvtTxPowerLevelRead_c (*C enumerator*), 383
gapConnectionEventType_t.gHandoverDisconnected_c (*C enumerator*), 385
gapConnectionFromPawrParameters_t (*C struct*), 463, 660
gapConnectionFromPawrParameters_t.advHandle (*C var*), 464, 661
gapConnectionFromPawrParameters_t.connEventLengthMax (*C var*), 464, 660
gapConnectionFromPawrParameters_t.connEventLengthMin (*C var*), 464, 660
gapConnectionFromPawrParameters_t.connIntervalMax (*C var*), 463, 660
gapConnectionFromPawrParameters_t.connIntervalMin (*C var*), 463, 660
gapConnectionFromPawrParameters_t.connLatency (*C var*), 464, 660
gapConnectionFromPawrParameters_t.filterPolicy (*C var*), 463, 660
gapConnectionFromPawrParameters_t.initiatingPHYs (*C var*), 464, 660
gapConnectionFromPawrParameters_t.ownAddressType (*C var*), 463, 660
gapConnectionFromPawrParameters_t.peerAddress (*C var*), 463, 660
gapConnectionFromPawrParameters_t.peerAddressType (*C var*), 463, 660
gapConnectionFromPawrParameters_t.scanInterval (*C var*), 463, 660
gapConnectionFromPawrParameters_t.scanWindow (*C var*), 463, 660
gapConnectionFromPawrParameters_t.subevent (*C var*), 464, 661
gapConnectionFromPawrParameters_t.supervisionTimeout (*C var*), 464, 660
gapConnectionFromPawrParameters_t.usePeerIdentityAddress (*C var*), 464, 660
gapConnectionlessCteTransmitParams_t (*C struct*), 465, 661
gapConnectionlessCteTransmitParams_t.aAntennaIds (*C var*), 465, 661

gapConnectionlessCteTransmitParams_t.cteCount (*C var*), 465, 661
gapConnectionlessCteTransmitParams_t.cteLength (*C var*), 465, 661
gapConnectionlessCteTransmitParams_t.cteType (*C var*), 465, 661
gapConnectionlessCteTransmitParams_t.handle (*C var*), 465, 661
gapConnectionlessCteTransmitParams_t.switchingPatternLength (*C var*), 465, 661
gapConnectionlessIqReport_t (*C struct*), 475, 669
gapConnectionlessIqReport_t.aI_samples (*C var*), 476, 670
gapConnectionlessIqReport_t.aQ_samples (*C var*), 476, 670
gapConnectionlessIqReport_t.channelIndex (*C var*), 476, 669
gapConnectionlessIqReport_t.cteType (*C var*), 476, 670
gapConnectionlessIqReport_t.packetStatus (*C var*), 476, 670
gapConnectionlessIqReport_t.periodicEventCounter (*C var*), 476, 670
gapConnectionlessIqReport_t.rssi (*C var*), 476, 670
gapConnectionlessIqReport_t.rssiAntennaId (*C var*), 476, 670
gapConnectionlessIqReport_t.sampleCount (*C var*), 476, 670
gapConnectionlessIqReport_t.slotDurations (*C var*), 476, 670
gapConnectionlessIqReport_t.syncHandle (*C var*), 475, 669
gapConnectionlessIqSamplingParams_t (*C struct*), 465, 661
gapConnectionlessIqSamplingParams_t.aAntennaIds (*C var*), 466, 662
gapConnectionlessIqSamplingParams_t.iqSamplingEnable (*C var*), 465, 662
gapConnectionlessIqSamplingParams_t.maxSampledCtes (*C var*), 466, 662
gapConnectionlessIqSamplingParams_t.slotDurations (*C var*), 465, 662
gapConnectionlessIqSamplingParams_t.switchingPatternLength (*C var*), 466, 662
gapConnectionParameters_t (*C struct*), 464, 661
gapConnectionParameters_t.centralClockAccuracy (*C var*), 464, 661
gapConnectionParameters_t.connInterval (*C var*), 464, 661
gapConnectionParameters_t.connLatency (*C var*), 464, 661
gapConnectionParameters_t.supervisionTimeout (*C var*), 464, 661
gapConnectionRequestParameters_t (*C struct*), 462, 659
gapConnectionRequestParameters_t.connEventLengthMax (*C var*), 463, 660
gapConnectionRequestParameters_t.connEventLengthMin (*C var*), 463, 660
gapConnectionRequestParameters_t.connIntervalMax (*C var*), 462, 659
gapConnectionRequestParameters_t.connIntervalMin (*C var*), 462, 659
gapConnectionRequestParameters_t.connLatency (*C var*), 462, 660
gapConnectionRequestParameters_t.filterPolicy (*C var*), 462, 659
gapConnectionRequestParameters_t.initiatingPHYs (*C var*), 463, 660
gapConnectionRequestParameters_t.ownAddressType (*C var*), 462, 659
gapConnectionRequestParameters_t.peerAddress (*C var*), 462, 659
gapConnectionRequestParameters_t.peerAddressType (*C var*), 462, 659
gapConnectionRequestParameters_t.scanInterval (*C var*), 462, 659
gapConnectionRequestParameters_t.scanWindow (*C var*), 462, 659
gapConnectionRequestParameters_t.supervisionTimeout (*C var*), 462, 660
gapConnectionRequestParameters_t.usePeerIdentityAddress (*C var*), 463, 660
gapConnectionSecurityInformation_t (*C struct*), 455, 655
gapConnectionSecurityInformation_t.authenticated (*C var*), 455, 655
gapConnectionSecurityInformation_t.authorizedToRead (*C var*), 455, 655
gapConnectionSecurityInformation_t.authorizedToWrite (*C var*), 455, 655
gapConnIqReport_t (*C struct*), 480, 673
gapConnIqReport_t.aI_samples (*C var*), 481, 673
gapConnIqReport_t.aQ_samples (*C var*), 481, 674
gapConnIqReport_t.connEventCounter (*C var*), 481, 673
gapConnIqReport_t.cteType (*C var*), 481, 673
gapConnIqReport_t.dataChannelIndex (*C var*), 480, 673
gapConnIqReport_t.packetStatus (*C var*), 481, 673
gapConnIqReport_t.rssi (*C var*), 481, 673
gapConnIqReport_t.rssiAntennaId (*C var*), 481, 673
gapConnIqReport_t.rxPhy (*C var*), 480, 673
gapConnIqReport_t.sampleCount (*C var*), 481, 673

gapConnIqReport_t.slotDurations (*C var*), 481, 673
gapConnLeDataLengthChanged_t (*C struct*), 480, 673
gapConnLeDataLengthChanged_t.maxRxOctets (*C var*), 480, 673
gapConnLeDataLengthChanged_t.maxRxTime (*C var*), 480, 673
gapConnLeDataLengthChanged_t.maxTxOctets (*C var*), 480, 673
gapConnLeDataLengthChanged_t.maxTxTime (*C var*), 480, 673
gapConnParamsUpdateComplete_t (*C struct*), 479, 672
gapConnParamsUpdateComplete_t.connInterval (*C var*), 480, 673
gapConnParamsUpdateComplete_t.connLatency (*C var*), 480, 673
gapConnParamsUpdateComplete_t.status (*C var*), 673
gapConnParamsUpdateComplete_t.supervisionTimeout (*C var*), 480, 673
gapConnParamsUpdateReq_t (*C struct*), 479, 672
gapConnParamsUpdateReq_t.intervalMax (*C var*), 479, 672
gapConnParamsUpdateReq_t.intervalMin (*C var*), 479, 672
gapConnParamsUpdateReq_t.peripheralLatency (*C var*), 479, 672
gapConnParamsUpdateReq_t.timeoutMultiplier (*C var*), 479, 672
gapControllerTestCmd_t (*C enum*), 379
gapControllerTestCmd_t.gControllerTestCmdEnd_c (*C enumerator*), 379
gapControllerTestCmd_t.gControllerTestCmdStartRx_c (*C enumerator*), 379
gapControllerTestCmd_t.gControllerTestCmdStartTx_c (*C enumerator*), 379
gapControllerTestEvent_t (*C struct*), 602, 636
gapControllerTestEvent_t.receivedPackets (*C var*), 636
gapControllerTestEvent_t.testEventType (*C var*), 636
gapControllerTestEventType_t (*C enum*), 575
gapControllerTestEventType_t.gControllerReceiverTestStarted_c (*C enumerator*), 575
gapControllerTestEventType_t.gControllerTestEnded_c (*C enumerator*), 576
gapControllerTestEventType_t.gControllerTransmitterTestStarted_c (*C enumerator*), 576
gapControllerTestTxType_t (*C type*), 389
gapControllerTestTxType_tag (*C enum*), 379
gapControllerTestTxType_tag.gControllerTestTx00_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTx0F_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTx55_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTxAA_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTxF0_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTxFF_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTxPrbs9_c (*C enumerator*), 379
gapControllerTestTxType_tag.gControllerTestTxPrbs15_c (*C enumerator*), 379
gapCreateSyncReqFilterPolicy_t (*C type*), 389
gapCreateSyncReqFilterPolicy_tag (*C enum*), 373
gapCreateSyncReqFilterPolicy_tag.gUseCommandParameters_c (*C enumerator*), 374
gapCreateSyncReqFilterPolicy_tag.gUsePeriodicAdvList_c (*C enumerator*), 374
gapCreateSyncReqOptions_t (*C type*), 389
gapCreateSyncReqOptions_tag (*C struct*), 461, 658
gapCreateSyncReqOptions_tag.duplicateFilteringEnabled (*C var*), 659
gapCreateSyncReqOptions_tag.filterPolicy (*C var*), 659
gapCreateSyncReqOptions_tag.reportingEnabled (*C var*), 659
gapDecisionInstructionsAdvAChecks_t (*C enum*), 378
gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAinFilterAcceptList_c (*C enumerator*), 378
gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAmatchAddress1_c (*C enumerator*), 378
gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAmatchAddress1orAddress2_c (*C enumerator*), 378
gapDecisionInstructionsAdvMode_t (*C type*), 389
gapDecisionInstructionsData_t (*C type*), 389
gapDecisionInstructionsData_tag (*C struct*), 468, 664
gapDecisionInstructionsData_tag.passCriteria (*C var*), 664
gapDecisionInstructionsData_tag.relevantField (*C var*), 664
gapDecisionInstructionsData_tag.testGroup (*C var*), 664
gapDecisionInstructionsData_tag.testParameters (*C union*), 484, 677

gapDecisionInstructionsData_tag.testParameters (*C var*), 664
gapDecisionInstructionsData_tag.testParameters.advA (*C struct*), 485, 678
gapDecisionInstructionsData_tag.testParameters.advA (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.advA.address1 (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.advA.address1Type (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.advA.address2 (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.advA.address2Type (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.advA.check (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.advMode (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.arbitraryData (*C struct*), 484, 677
gapDecisionInstructionsData_tag.testParameters.arbitraryData (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.arbitraryData.mask (*C var*), 677
gapDecisionInstructionsData_tag.testParameters.arbitraryData.target (*C var*), 677
gapDecisionInstructionsData_tag.testParameters.pathLoss (*C struct*), 485, 677
gapDecisionInstructionsData_tag.testParameters.pathLoss (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.pathLoss.max (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.pathLoss.min (*C var*), 678
gapDecisionInstructionsData_tag.testParameters.resolvableTagKey (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.rssi (*C struct*), 485, 677
gapDecisionInstructionsData_tag.testParameters.rssi (*C var*), 484, 677
gapDecisionInstructionsData_tag.testParameters.rssi.max (*C var*), 677
gapDecisionInstructionsData_tag.testParameters.rssi.min (*C var*), 677
gapDecisionInstructionsRelevantField_t (*C enum*), 376
gapDecisionInstructionsRelevantField_t.gDIRF_AdvAddress_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_AdvMode_c (*C enumerator*), 376
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_1Byte_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_2Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_3Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_4Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_5Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_6Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_7Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_8Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_1Byte_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_2Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_3Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_4Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_5Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_6Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_7Bytes_c (*C enumerator*), 378
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_8Bytes_c (*C enumerator*), 378
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_1Byte_c (*C enumerator*),

377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_2Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_3Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_4Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_5Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_6Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_7Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_8Bytes_c (*C enumerator*), 377
gapDecisionInstructionsRelevantField_t.gDIRF_PathLoss_c (*C enumerator*), 376
gapDecisionInstructionsRelevantField_t.gDIRF_ResolvableTag_c (*C enumerator*), 376
gapDecisionInstructionsRelevantField_t.gDIRF_RSSI_c (*C enumerator*), 376
gapDecisionInstructionsTestGroup_t (*C enum*), 378
gapDecisionInstructionsTestGroup_t.gDITG_NewTestGroup_c (*C enumerator*), 378
gapDecisionInstructionsTestGroup_t.gDITG_SameTestGroup_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t (*C enum*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_Always_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckFails_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckFailsOrRelevantFieldNotPresent_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckPasses_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckPassesOrRelevantFieldNotPresent_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_Never_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_RelevantFieldNotPresent_c (*C enumerator*), 378
gapDecisionInstructionsTestPassCriteria_t.gDITPC_RelevantFieldPresent_c (*C enumerator*), 378
gapDeviceSecurityRequirements_t (*C struct*), 454, 654
gapDeviceSecurityRequirements_t.aServiceSecurityRequirements (*C var*), 455, 654
gapDeviceSecurityRequirements_t.cNumServices (*C var*), 455, 654
gapDeviceSecurityRequirements_t.pSecurityRequirements (*C var*), 455, 654
gapDisconnectedEvent_t (*C struct*), 479, 672
gapDisconnectedEvent_t.reason (*C var*), 479, 672
gapDisconnectionReason_t (*C type*), 389
gapEattBearerStatus_t (*C enum*), 385
gapEattBearerStatus_t.gEnhancedBearerActive_c (*C enumerator*), 385
gapEattBearerStatus_t.gEnhancedBearerDisconnected_c (*C enumerator*), 385
gapEattBearerStatus_t.gEnhancedBearerNoPeerCredits_c (*C enumerator*), 385
gapEattBearerStatus_t.gEnhancedBearerStatusEnd_c (*C enumerator*), 385
gapEattBearerStatus_t.gEnhancedBearerSuspendedNoLocalCredits_c (*C enumerator*), 385
gapEattBearerStatusNotification_t (*C struct*), 483, 675
gapEattBearerStatusNotification_t.bearerId (*C var*), 675
gapEattBearerStatusNotification_t.status (*C var*), 675
gapEattConnectionComplete_t (*C struct*), 482, 675
gapEattConnectionComplete_t.aBearerIds (*C var*), 675
gapEattConnectionComplete_t.cBearers (*C var*), 675
gapEattConnectionComplete_t.mtu (*C var*), 675
gapEattConnectionComplete_t.status (*C var*), 675
gapEattConnectionRequest_t (*C struct*), 482, 674
gapEattConnectionRequest_t.cBearers (*C var*), 675
gapEattConnectionRequest_t.initialCredits (*C var*), 675
gapEattConnectionRequest_t.mtu (*C var*), 675
gapEattReconfigureResponse_t (*C struct*), 483, 675

gapEattReconfigureResponse_t.aBearerIds (*C var*), 675
gapEattReconfigureResponse_t.cBearers (*C var*), 675
gapEattReconfigureResponse_t.localMtu (*C var*), 675
gapEattReconfigureResponse_t.status (*C var*), 675
gapEncryptionChangedEvent_t (*C struct*), 479, 672
gapEncryptionChangedEvent_t.newEncryptionState (*C var*), 479, 672
gapExtAdvertisingParameters_t (*C type*), 388
gapExtAdvertisingParameters_tag (*C struct*), 457, 656
gapExtAdvertisingParameters_tag.channelMap (*C var*), 458, 656
gapExtAdvertisingParameters_tag.enableScanReqNotification (*C var*), 458, 657
gapExtAdvertisingParameters_tag.extAdvProperties (*C var*), 458, 656
gapExtAdvertisingParameters_tag.filterPolicy (*C var*), 458, 656
gapExtAdvertisingParameters_tag.handle (*C var*), 457, 656
gapExtAdvertisingParameters_tag.maxInterval (*C var*), 457, 656
gapExtAdvertisingParameters_tag.minInterval (*C var*), 457, 656
gapExtAdvertisingParameters_tag.ownAddressType (*C var*), 457, 656
gapExtAdvertisingParameters_tag.ownRandomAddr (*C var*), 458, 656
gapExtAdvertisingParameters_tag.peerAddress (*C var*), 458, 656
gapExtAdvertisingParameters_tag.peerAddressType (*C var*), 458, 656
gapExtAdvertisingParameters_tag.primaryAdvPhyOptions (*C var*), 458, 657
gapExtAdvertisingParameters_tag.primaryPHY (*C var*), 458, 657
gapExtAdvertisingParameters_tag.secondaryAdvMaxSkip (*C var*), 458, 657
gapExtAdvertisingParameters_tag.secondaryAdvPhyOptions (*C var*), 458, 657
gapExtAdvertisingParameters_tag.secondaryPHY (*C var*), 458, 657
gapExtAdvertisingParameters_tag.SID (*C var*), 457, 656
gapExtAdvertisingParameters_tag.txPower (*C var*), 458, 656
gapExtScannedDevice_t (*C struct*), 472, 667
gapExtScannedDevice_t.aAddress (*C var*), 472, 667
gapExtScannedDevice_t.addressType (*C var*), 472, 667
gapExtScannedDevice_t.advertisingAddressResolved (*C var*), 472, 667
gapExtScannedDevice_t.advEventProperties (*C var*), 472, 667
gapExtScannedDevice_t.dataLength (*C var*), 473, 668
gapExtScannedDevice_t.directRpa (*C var*), 473, 668
gapExtScannedDevice_t.directRpaType (*C var*), 473, 668
gapExtScannedDevice_t.directRpaUsed (*C var*), 473, 667
gapExtScannedDevice_t.pData (*C var*), 473, 668
gapExtScannedDevice_t.periodicAdvInterval (*C var*), 473, 667
gapExtScannedDevice_t.primaryPHY (*C var*), 473, 667
gapExtScannedDevice_t.rssi (*C var*), 472, 667
gapExtScannedDevice_t.secondaryPHY (*C var*), 473, 667
gapExtScannedDevice_t.SID (*C var*), 472, 667
gapExtScannedDevice_t.txPower (*C var*), 473, 667
gapExtScanNotification_t (*C struct*), 470, 665
gapExtScanNotification_t.aScannerAddr (*C var*), 470, 665
gapExtScanNotification_t.handle (*C var*), 470, 665
gapExtScanNotification_t.scannerAddrResolved (*C var*), 470, 665
gapExtScanNotification_t.scannerAddrType (*C var*), 470, 665
gapFilterDuplicates_t (*C enum*), 373
gapFilterDuplicates_t.gGapDuplicateFilteringDisable_c (*C enumerator*), 373
gapFilterDuplicates_t.gGapDuplicateFilteringEnable_c (*C enumerator*), 373
gapFilterDuplicates_t.gGapDuplicateFilteringPeriodicEnable_c (*C enumerator*), 373
gapGenerateDHKeyV2Params_t (*C struct*), 465, 661
gapGenerateDHKeyV2Params_t.keyType (*C var*), 465, 661
gapGenerateDHKeyV2Params_t.remoteP256PublicKey (*C var*), 465, 661
gapGenericCallback_t (*C type*), 584
gapGenericEvent_t (*C struct*), 615, 648
gapGenericEvent_t.eventData (*C union*), 617, 650
gapGenericEvent_t.eventData (*C var*), 615, 648

gapGenericEvent_t.eventData.aAddress (*C var*), 617, 650
gapGenericEvent_t.eventData.aControllerLocalRPA (*C var*), 619, 651
gapGenericEvent_t.eventData.addrReady (*C var*), 617, 651
gapGenericEvent_t.eventData.advHandle (*C var*), 618, 651
gapGenericEvent_t.eventData.advTxPowerLevel_dBm (*C var*), 618, 651
gapGenericEvent_t.eventData.antennaInformation (*C var*), 619, 651
gapGenericEvent_t.eventData.bondCreatedEvent (*C var*), 618, 651
gapGenericEvent_t.eventData.deviceId (*C var*), 618, 651
gapGenericEvent_t.eventData.filterAcceptListSize (*C var*), 617, 650
gapGenericEvent_t.eventData.gapRemoteVersionInfoRead (*C var*), 620, 652
gapGenericEvent_t.eventData.gapSkdReport (*C var*), 620, 652
gapGenericEvent_t.eventData.getConnParams (*C var*), 619, 651
gapGenericEvent_t.eventData.handoverAnchorMonitor (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverAnchorMonitorPacket (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverAnchorMonitorPacketContinue (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverAnchorNotificationStateChanged (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverAnchorSearchStart (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverAnchorSearchStop (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverApplyConnectionUpdateProcedure (*C var*), 621, 653
gapGenericEvent_t.eventData.handoverConnect (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverConnectionUpdateProcedure (*C var*), 621, 653
gapGenericEvent_t.eventData.handoverConnParamUpdate (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverGetCsLlContext (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverGetData (*C var*), 619, 652
gapGenericEvent_t.eventData.handoverGetTime (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverLlPendingDataIndication (*C var*), 621, 653
gapGenericEvent_t.eventData.handoverResumeTransmitComplete (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverSetData (*C var*), 619, 652
gapGenericEvent_t.eventData.handoverSuspendTransmitComplete (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverTimeSync (*C var*), 620, 652
gapGenericEvent_t.eventData.handoverUpdateConnParams (*C var*), 620, 652
gapGenericEvent_t.eventData.initCompleteData (*C var*), 618, 651
gapGenericEvent_t.eventData.internalError (*C var*), 617, 650
gapGenericEvent_t.eventData.leGenerateDhKeyCompleteEvent (*C var*), 619, 652
gapGenericEvent_t.eventData.localOobData (*C var*), 618, 651
gapGenericEvent_t.eventData.monAdvListSize (*C var*), 621, 653
gapGenericEvent_t.eventData.newControllerPrivacyState (*C var*), 618, 651
gapGenericEvent_t.eventData.newHostPrivacyState (*C var*), 618, 651
gapGenericEvent_t.eventData.notifEvent (*C var*), 618, 651
gapGenericEvent_t.eventData.pawrAdvHandle (*C var*), 619, 652
gapGenericEvent_t.eventData.pawrSyncHandle (*C var*), 619, 652
gapGenericEvent_t.eventData.perAdvSetDefaultPerAdvSyncTransferParams (*C var*), 619, 652
gapGenericEvent_t.eventData.perAdvSetInfoTransfer (*C var*), 619, 651
gapGenericEvent_t.eventData.perAdvSetSyncTransferParams (*C var*), 619, 651
gapGenericEvent_t.eventData.perAdvSyncTransfer (*C var*), 619, 651
gapGenericEvent_t.eventData.perAdvSyncTransferEnable (*C var*), 619, 651
gapGenericEvent_t.eventData.perAdvSyncTransferReceived (*C var*), 619, 652
gapGenericEvent_t.eventData.phyEvent (*C var*), 618, 651
gapGenericEvent_t.eventData.setupFailError (*C var*), 618, 651
gapGenericEvent_t.eventData.syncHandle (*C var*), 619, 651
gapGenericEvent_t.eventData.testEvent (*C var*), 618, 651
gapGenericEvent_t.eventData.txPowerLevelSetStatus (*C var*), 618, 651
gapGenericEvent_t.eventData.unitaryTestData (*C var*), 621, 653
gapGenericEvent_t.eventData.verified (*C var*), 618, 651
gapGenericEvent_t.eventType (*C var*), 615, 648
gapGenericEventType_t (*C enum*), 562
gapGenericEventType_t.gAdvertisingDataSetupComplete_c (*C enumerator*), 562
gapGenericEventType_t.gAdvertisingParametersSetupComplete_c (*C enumerator*), 562

gapGenericEventType_t.gAdvertisingSetupFailed_c (*C enumerator*), 562
gapGenericEventType_t.gAdvTxPowerLevelRead_c (*C enumerator*), 563
gapGenericEventType_t.gAntennaInformationRead_c (*C enumerator*), 565
gapGenericEventType_t.gBondCreatedEvent_c (*C enumerator*), 564
gapGenericEventType_t.gChannelMapSet_c (*C enumerator*), 564
gapGenericEventType_t.gConnectionlessCteTransmitParamsSetupComplete_c (*C enumerator*), 565
gapGenericEventType_t.gConnectionlessCteTransmitStateChanged_c (*C enumerator*), 565
gapGenericEventType_t.gConnectionlessIqSamplingStateChanged_c (*C enumerator*), 565
gapGenericEventType_t.gConnEvtLeGenerateDhKeyComplete_c (*C enumerator*), 565
gapGenericEventType_t.gControllerLocalRPARead_c (*C enumerator*), 565
gapGenericEventType_t.gControllerNotificationEvent_c (*C enumerator*), 564
gapGenericEventType_t.gControllerPrivacyStateChanged_c (*C enumerator*), 563
gapGenericEventType_t.gControllerTestEvent_c (*C enumerator*), 564
gapGenericEventType_t.gCreateConnectionCanceled_c (*C enumerator*), 563
gapGenericEventType_t.gDecisionInstructionsSetupComplete_c (*C enumerator*), 567
gapGenericEventType_t.gDeInitializationComplete_c (*C enumerator*), 567
gapGenericEventType_t.gDeviceAddedToFilterAcceptList_c (*C enumerator*), 563
gapGenericEventType_t.gDeviceAddedToMonAdvList_c (*C enumerator*), 568
gapGenericEventType_t.gDeviceRemovedFromFilterAcceptList_c (*C enumerator*), 563
gapGenericEventType_t.gDeviceRemovedFromMonAdvList_c (*C enumerator*), 568
gapGenericEventType_t.gExtAdvertisingDataSetupComplete_c (*C enumerator*), 564
gapGenericEventType_t.gExtAdvertisingDecisionDataSetupComplete_c (*C enumerator*), 567
gapGenericEventType_t.gExtAdvertisingParametersSetupComplete_c (*C enumerator*), 564
gapGenericEventType_t.gExtAdvertisingSetRemoveComplete_c (*C enumerator*), 564
gapGenericEventType_t.gFilterAcceptListCleared_c (*C enumerator*), 563
gapGenericEventType_t.gFilterAcceptListSizeRead_c (*C enumerator*), 563
gapGenericEventType_t.gGetConnParamsComplete_c (*C enumerator*), 564
gapGenericEventType_t.gHandoverAnchorMonitorEvent_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverAnchorMonitorPacketContinueEvent_c (*C enumerator*), 567
gapGenericEventType_t.gHandoverAnchorMonitorPacketEvent_c (*C enumerator*), 567
gapGenericEventType_t.gHandoverAnchorNotificationStateChanged_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverAnchorSearchStarted_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverAnchorSearchStopped_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverApplyConnectionUpdateProcedureComplete_c (*C enumerator*), 568
gapGenericEventType_t.gHandoverConnectionUpdateProcedureEvent_c (*C enumerator*), 567
gapGenericEventType_t.gHandoverConnParamUpdateEvent_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverFreeComplete_c (*C enumerator*), 567
gapGenericEventType_t.gHandoverGetComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverGetCsLlContextComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverGetTime_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverLlPendingData_c (*C enumerator*), 567
gapGenericEventType_t.gHandoverResumeTransmitComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverSetComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverSetCsLlContextComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverSuspendTransmitComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverTimeSyncEvent_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverTimeSyncReceiveComplete_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverTimeSyncTransmitStateChanged_c (*C enumerator*), 566
gapGenericEventType_t.gHandoverUpdateConnParamsComplete_c (*C enumerator*), 567
gapGenericEventType_t.gHostPrivacyStateChanged_c (*C enumerator*), 563
gapGenericEventType_t.gInitializationComplete_c (*C enumerator*), 562
gapGenericEventType_t.gInternalError_c (*C enumerator*), 562
gapGenericEventType_t.gLeChannelOverrideComplete_c (*C enumerator*), 567
gapGenericEventType_t.gLePeriodicAdvUpdateSyncComplete_c (*C enumerator*), 568
gapGenericEventType_t.gLePhyEvent_c (*C enumerator*), 564
gapGenericEventType_t.gLeScLocalOobData_c (*C enumerator*), 563
gapGenericEventType_t.gLeScPublicKeyRegenerated_c (*C enumerator*), 563

gapGenericEventType_t.gLeSetDataRelatedAddressChangesComplete_c (*C enumerator*), 568
gapGenericEventType_t.gLeSetSchedulerPriorityComplete_c (*C enumerator*), 567
gapGenericEventType_t.gLlSkdReportEvent_c (*C enumerator*), 567
gapGenericEventType_t.gModifiedSleepClockAccuracy_c (*C enumerator*), 565
gapGenericEventType_t.gMonAdvEnabled_c (*C enumerator*), 568
gapGenericEventType_t.gMonAdvListCleared_c (*C enumerator*), 568
gapGenericEventType_t.gMonAdvListSizeRead_c (*C enumerator*), 568
gapGenericEventType_t.gPeriodicAdvCreateSyncCancelled_c (*C enumerator*), 564
gapGenericEventType_t.gPeriodicAdvDataSetupComplete_c (*C enumerator*), 564
gapGenericEventType_t.gPeriodicAdvertisingStateChanged_c (*C enumerator*), 564
gapGenericEventType_t.gPeriodicAdvListUpdateComplete_c (*C enumerator*), 564
gapGenericEventType_t.gPeriodicAdvParamSetupComplete_c (*C enumerator*), 564
gapGenericEventType_t.gPeriodicAdvRecvEnableComplete_c (*C enumerator*), 565
gapGenericEventType_t.gPeriodicAdvSetInfoTransferComplete_c (*C enumerator*), 565
gapGenericEventType_t.gPeriodicAdvSetResponseDataComplete_c (*C enumerator*), 567
gapGenericEventType_t.gPeriodicAdvSetSubeventDataComplete_c (*C enumerator*), 567
gapGenericEventType_t.gPeriodicAdvSyncTransferComplete_c (*C enumerator*), 565
gapGenericEventType_t.gPeriodicAdvSyncTransferFailed_c (*C enumerator*), 565
gapGenericEventType_t.gPeriodicAdvSyncTransferSucceeded_c (*C enumerator*), 565
gapGenericEventType_t.gPeriodicSyncSubeventComplete_c (*C enumerator*), 567
gapGenericEventType_t.gPrivateResolvableAddressVerified_c (*C enumerator*), 563
gapGenericEventType_t.gPublicAddressRead_c (*C enumerator*), 563
gapGenericEventType_t.gRandomAddressReady_c (*C enumerator*), 563
gapGenericEventType_t.gRandomAddressSet_c (*C enumerator*), 563
gapGenericEventType_t.gRemoteVersionInformationRead_c (*C enumerator*), 566
gapGenericEventType_t.gSetDefaultPeriodicAdvSyncTransferParamsComplete_c (*C enumerator*), 565
gapGenericEventType_t.gSetPeriodicAdvSyncTransferParamsComplete_c (*C enumerator*), 565
gapGenericEventType_t.gTxEntryAvailable_c (*C enumerator*), 564
gapGenericEventType_t.gTxPowerLevelSetComplete_c (*C enumerator*), 564
gapGenericEventType_t.gVendorUnitaryTestComplete_c (*C enumerator*), 568
gapHandleList_t (*C struct*), 455, 654
gapHandleList_t.aHandles (*C var*), 455, 654
gapHandleList_t.cNumHandles (*C var*), 455, 654
gapHandoverConnectedEvent_t (*C struct*), 483, 675
gapHandoverConnectedEvent_t.connectionRole (*C var*), 483, 676
gapHandoverConnectedEvent_t.peerAddress (*C var*), 483, 676
gapHandoverConnectedEvent_t.peerAddressType (*C var*), 483, 676
gapHandoverDisconnectedEvent_t (*C struct*), 483, 676
gapHandoverDisconnectedEvent_t.status (*C var*), 483, 676
gapHostVersion_t (*C type*), 390
gapHostVersion_tag (*C struct*), 484, 676
gapHostVersion_tag.bleHostVerMajor (*C var*), 677
gapHostVersion_tag.bleHostVerMinor (*C var*), 677
gapHostVersion_tag.bleHostVerPatch (*C var*), 677
gapIdentityInformation_t (*C struct*), 483, 676
gapIdentityInformation_t.identityAddress (*C var*), 484, 676
gapIdentityInformation_t.irk (*C var*), 484, 676
gapIdentityInformation_t.privacyMode (*C var*), 484, 676
gapInitComplete_t (*C struct*), 603, 637
gapInitComplete_t.leExtendedFeatures (*C var*), 637
gapInitComplete_t.maxAdvDataSize (*C var*), 637
gapInitComplete_t.numOfSupportedAdvSets (*C var*), 637
gapInitComplete_t.periodicAdvListSize (*C var*), 637
gapInitComplete_t.supportedFeatures (*C var*), 637
gapInternalError_t (*C struct*), 602, 636
gapInternalError_t.errorCode (*C var*), 602, 636
gapInternalError_t.errorSource (*C var*), 602, 636
gapInternalError_t.hciCommandOpcode (*C var*), 602, 636

gapInternalErrorSource_t (*C enum*), 568
gapInternalErrorSource_t.gAddDeviceToFilterAcceptList_c (*C enumerator*), 569
gapInternalErrorSource_t.gAddDeviceToMonAdvList_c (*C enumerator*), 575
gapInternalErrorSource_t.gAddNewConnection_c (*C enumerator*), 568
gapInternalErrorSource_t.gCancelCreateConnection_c (*C enumerator*), 569
gapInternalErrorSource_t.gCheckPrivateResolvableAddress_c (*C enumerator*), 568
gapInternalErrorSource_t.gClearFilterAcceptList_c (*C enumerator*), 569
gapInternalErrorSource_t.gClearMonAdvList_c (*C enumerator*), 575
gapInternalErrorSource_t.gConnect_c (*C enumerator*), 569
gapInternalErrorSource_t.gConnectionCteReqEnable_c (*C enumerator*), 571
gapInternalErrorSource_t.gConnectionCteRspEnable_c (*C enumerator*), 571
gapInternalErrorSource_t.gCreateRandomAddress_c (*C enumerator*), 569
gapInternalErrorSource_t.gDefaultPairingProcedure_c (*C enumerator*), 570
gapInternalErrorSource_t.gDenyLongTermKey_c (*C enumerator*), 569
gapInternalErrorSource_t.gDisconnect_c (*C enumerator*), 569
gapInternalErrorSource_t.gEattConnectionAccept_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattConnectionRequest_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattDisconnectRequest_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caCancelConnection_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caDisconnectLePsm_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caEnhancedCancelConnection_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caEnhancedReconfigureReq_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caHandleRecvLeCbData_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caHandleSendLeCbData_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattL2caSendLeFlowControlCredit_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattReconfigureRequest_c (*C enumerator*), 572
gapInternalErrorSource_t.gEattSendCreditsRequest_c (*C enumerator*), 572
gapInternalErrorSource_t.gEnableControllerPrivacy_c (*C enumerator*), 570
gapInternalErrorSource_t.gEnableHostPrivacy_c (*C enumerator*), 570
gapInternalErrorSource_t.gEnableLdmTimer_c (*C enumerator*), 570
gapInternalErrorSource_t.gEnableMonAdv_c (*C enumerator*), 575
gapInternalErrorSource_t.gEncryptLink_c (*C enumerator*), 569
gapInternalErrorSource_t.gEnhancedReadTransmitPowerLevel_c (*C enumerator*), 572
gapInternalErrorSource_t.gEnterPasskey_c (*C enumerator*), 569
gapInternalErrorSource_t.gExtAdvReportProcess_c (*C enumerator*), 571
gapInternalErrorSource_t.gGenerateDHKeyV2_c (*C enumerator*), 571
gapInternalErrorSource_t.gGetConnParams_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandover_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverAnchorNotifHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverAnchorNotifLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverAnchorSearchStartHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverAnchorSearchStartLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverAnchorSearchStopHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverAnchorSearchStopLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverApplyConnectionUpdateProcedure_c (*C enumerator*), 575
gapInternalErrorSource_t.gHandoverConnect_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverDisconnect_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverGetCsLlContext_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverGetLlContext_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverGetTimeLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverResumeTransmitHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverResumeTransmitLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverSetCsLlContext_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverSetLlPendingData_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverSetSkd_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverSuspendTransmitHciTx_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverSuspendTransmitHostTxInProgress_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverSuspendTransmitLl_c (*C enumerator*), 574

gapInternalErrorSource_t.gHandoverSuspendTransmitLlProcInProgress_c (*C enumerator*), 573
gapInternalErrorSource_t.gHandoverTimeSyncRxHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverTimeSyncRxLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverTimeSyncTxHciTx_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverTimeSyncTxLl_c (*C enumerator*), 574
gapInternalErrorSource_t.gHandoverUpdateConnParams_c (*C enumerator*), 573
gapInternalErrorSource_t.gHciCommandStatus_c (*C enumerator*), 568
gapInternalErrorSource_t.gHciDataDiscardedAlloc_c (*C enumerator*), 572
gapInternalErrorSource_t.gHciDataDiscardedInvalidStateParam_c (*C enumerator*), 572
gapInternalErrorSource_t.gHciEventReceiveHandler_c (*C enumerator*), 571
gapInternalErrorSource_t.gHciRecvFragmentOfPacket_c (*C enumerator*), 572
gapInternalErrorSource_t.gL2capRxPacket_c (*C enumerator*), 571
gapInternalErrorSource_t.gLeAddDeviceToPeriodicAdvList_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeChannelOverride_c (*C enumerator*), 574
gapInternalErrorSource_t.gLeClearPeriodicAdvList_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeControllerTest_c (*C enumerator*), 570
gapInternalErrorSource_t.gLePeriodicAdvResponseReport_c (*C enumerator*), 575
gapInternalErrorSource_t.gLePeriodicAdvSyncEstb_c (*C enumerator*), 570
gapInternalErrorSource_t.gLePeriodicAdvSyncLost_c (*C enumerator*), 570
gapInternalErrorSource_t.gLePeriodicAdvUpdateSync_c (*C enumerator*), 575
gapInternalErrorSource_t.gLeReadMaxAdvDataLen_c (*C enumerator*), 571
gapInternalErrorSource_t.gLeReadNumOfSupportedAdvSets_c (*C enumerator*), 571
gapInternalErrorSource_t.gLeReadPeriodicAdvListSize_c (*C enumerator*), 571
gapInternalErrorSource_t.gLeReadPhy_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeRemoveDeviceFromPeriodicAdvList_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeScGetLocalOobData_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeScRegeneratePublicKey_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeScSendKeypressNotification_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeScSetPeerOobData_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeScValidateNumericValue_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeSetHostFeature_c (*C enumerator*), 573
gapInternalErrorSource_t.gLeSetPeriodicAdvParamsV2_c (*C enumerator*), 575
gapInternalErrorSource_t.gLeSetPeriodicAdvResponseData_c (*C enumerator*), 575
gapInternalErrorSource_t.gLeSetPeriodicAdvSubeventData_c (*C enumerator*), 575
gapInternalErrorSource_t.gLeSetPeriodicSyncSubevent_c (*C enumerator*), 575
gapInternalErrorSource_t.gLeSetPhy_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeSetResolvablePrivateAddressTimeout_c (*C enumerator*), 570
gapInternalErrorSource_t.gLeSetSchedulerPriority_c (*C enumerator*), 573
gapInternalErrorSource_t.gModifySleepClockAccuracy_c (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvCancelSync (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvCreateSync (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvRcvEnable_c (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvSetInfoTransfer_c (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvSyncTransfer_c (*C enumerator*), 571
gapInternalErrorSource_t.gPeriodicAdvTerminateSync (*C enumerator*), 571
gapInternalErrorSource_t.gProvideLongTermKey_c (*C enumerator*), 569
gapInternalErrorSource_t.gProvideOob_c (*C enumerator*), 569
gapInternalErrorSource_t.gReadAllLocalSupportedFeatures_c (*C enumerator*), 575
gapInternalErrorSource_t.gReadAntennaInformation_c (*C enumerator*), 571
gapInternalErrorSource_t.gReadControllerLocalRPA_c (*C enumerator*), 571
gapInternalErrorSource_t.gReadDeviceAddress_c (*C enumerator*), 569
gapInternalErrorSource_t.gReadFilterAcceptListSize_c (*C enumerator*), 569
gapInternalErrorSource_t.gReadLeBufferSize_c (*C enumerator*), 568
gapInternalErrorSource_t.gReadLocalSupportedCommands_c (*C enumerator*), 570
gapInternalErrorSource_t.gReadLocalSupportedFeatures_c (*C enumerator*), 569
gapInternalErrorSource_t.gReadMonAdvListSize_c (*C enumerator*), 575
gapInternalErrorSource_t.gReadRadioPower_c (*C enumerator*), 569
gapInternalErrorSource_t.gReadRemoteTransmitPowerLevel_c (*C enumerator*), 572

gapInternalErrorSource_t.gReadRemoteVersionInfo_c (*C enumerator*), 573
gapInternalErrorSource_t.gReadSuggestedDefaultDataLength_c (*C enumerator*), 569
gapInternalErrorSource_t.gRemoveAdvertisingSet_c (*C enumerator*), 570
gapInternalErrorSource_t.gRemoveDeviceFromFilterAcceptList_c (*C enumerator*), 569
gapInternalErrorSource_t.gRemoveDeviceFromMonAdvList_c (*C enumerator*), 575
gapInternalErrorSource_t.gResetController_c (*C enumerator*), 568
gapInternalErrorSource_t.gSaveKeys_c (*C enumerator*), 570
gapInternalErrorSource_t.gSendPeripheralSecurityRequest_c (*C enumerator*), 569
gapInternalErrorSource_t.gSendSmpKeys_c (*C enumerator*), 569
gapInternalErrorSource_t.gSetChannelMap_c (*C enumerator*), 570
gapInternalErrorSource_t.gSetConnectionCteReceiveParams_c (*C enumerator*), 571
gapInternalErrorSource_t.gSetConnectionCteTransmitParams_c (*C enumerator*), 571
gapInternalErrorSource_t.gSetConnectionlessCteTransmitEnable_c (*C enumerator*), 571
gapInternalErrorSource_t.gSetConnectionlessCteTransmitParams_c (*C enumerator*), 571
gapInternalErrorSource_t.gSetConnectionlessIqSamplingEnable_c (*C enumerator*), 571
gapInternalErrorSource_t.gSetDataRelatedAddressChanges_c (*C enumerator*), 575
gapInternalErrorSource_t.gSetDecisionInstructions_c (*C enumerator*), 573
gapInternalErrorSource_t.gSetDefaultPeriodicAdvSyncTransferParams_c (*C enumerator*), 572
gapInternalErrorSource_t.gSetEventMask_c (*C enumerator*), 568
gapInternalErrorSource_t.gSetExpmSupportedFeatures_c (*C enumerator*), 573
gapInternalErrorSource_t.gSetExtAdvDecisionData_c (*C enumerator*), 573
gapInternalErrorSource_t.gSetLeEventMask_c (*C enumerator*), 569
gapInternalErrorSource_t.gSetPathLossReportingEnable_c (*C enumerator*), 572
gapInternalErrorSource_t.gSetPathLossReportingParams_c (*C enumerator*), 572
gapInternalErrorSource_t.gSetPeriodicAdvSyncTransferParams_c (*C enumerator*), 572
gapInternalErrorSource_t.gSetRandomAddress_c (*C enumerator*), 569
gapInternalErrorSource_t.gSetTransmitPowerReportingEnable_c (*C enumerator*), 572
gapInternalErrorSource_t.gTerminatePairing_c (*C enumerator*), 569
gapInternalErrorSource_t.gUpdateLeDataLength_c (*C enumerator*), 570
gapInternalErrorSource_t.gVendorUnitaryTest_c (*C enumerator*), 575
gapInternalErrorSource_t.gVerifySignature_c (*C enumerator*), 568
gapInternalErrorSource_t.gWriteSuggestedDefaultDataLength_c (*C enumerator*), 569
gapIoCapabilities_t (*C type*), 388
gapKeyExchangeRequestEvent_t (*C struct*), 477, 671
gapKeyExchangeRequestEvent_t.requestedKeys (*C var*), 478, 671
gapKeyExchangeRequestEvent_t.requestedLtkSize (*C var*), 478, 671
gapKeypressNotification_t (*C type*), 388
gapKeypressNotification_tag (*C enum*), 371
gapKeypressNotification_tag (*C type*), 388
gapKeypressNotification_tag.gKnPasskeyCleared_c (*C enumerator*), 372
gapKeypressNotification_tag.gKnPasskeyDigitErased_c (*C enumerator*), 372
gapKeypressNotification_tag.gKnPasskeyDigitStarted_c (*C enumerator*), 372
gapKeypressNotification_tag.gKnPasskeyEntryCompleted_c (*C enumerator*), 372
gapKeypressNotification_tag.gKnPasskeyEntryStarted_c (*C enumerator*), 372
gapKeysReceivedEvent_t (*C struct*), 478, 671
gapKeysReceivedEvent_t.pKeys (*C var*), 478, 671
gapLeAllPhyFlags_t (*C enum*), 576
gapLeAllPhyFlags_t.gLeRxPhyNoPreference_c (*C enumerator*), 576
gapLeAllPhyFlags_t.gLeTxPhyNoPreference_c (*C enumerator*), 576
gapLEGenerateDhKeyCompleteEvent_t (*C struct*), 615, 647
gapLEGenerateDhKeyCompleteEvent_t.aDHKey (*C var*), 647
gapLePhyFlags_t (*C type*), 583
gapLePhyMode_t (*C type*), 583
gapLePhyMode_tag (*C enum*), 576
gapLePhyMode_tag.gLePhy1M_c (*C enumerator*), 576
gapLePhyMode_tag.gLePhy2M_c (*C enumerator*), 576
gapLePhyMode_tag.gLePhyCoded_c (*C enumerator*), 576
gapLePhyOptionsFlags_t (*C enum*), 576

gapLePhyOptionsFlags_t.gLeCodingNoPreference_c (*C enumerator*), 576
gapLePhyOptionsFlags_t.gLeCodingS2_c (*C enumerator*), 576
gapLePhyOptionsFlags_t.gLeCodingS2Req_c (*C enumerator*), 576
gapLePhyOptionsFlags_t.gLeCodingS8_c (*C enumerator*), 576
gapLePhyOptionsFlags_t.gLeCodingS8Req_c (*C enumerator*), 576
gapLeScOobData_t (*C struct*), 602, 636
gapLeScOobData_t.confirmValue (*C var*), 602, 636
gapLeScOobData_t.randomValue (*C var*), 602, 636
gapLlSkdReport_t (*C type*), 584
gapLlSkdReport_tag (*C struct*), 615, 647
gapLlSkdReport_tag.aSKD (*C var*), 615, 647
gapLlSkdReport_tag.deviceId (*C var*), 615, 647
gapLongTermKeyRequestEvent_t (*C struct*), 478, 672
gapLongTermKeyRequestEvent_t.aRand (*C var*), 479, 672
gapLongTermKeyRequestEvent_t.ediv (*C var*), 479, 672
gapLongTermKeyRequestEvent_t.randSize (*C var*), 479, 672
gapMonAdvReportReport_t (*C struct*), 476, 670
gapMonAdvReportReport_t.condition (*C var*), 476, 670
gapMonAdvReportReport_t.peerAddress (*C var*), 476, 670
gapMonAdvReportReport_t.peerAddressType (*C var*), 476, 670
gApp2Host_TaskQueue (*C var*), 585
gapPairingCompleteEvent_t (*C struct*), 478, 671
gapPairingCompleteEvent_t.pairingCompleteData (*C union*), 486, 679
gapPairingCompleteEvent_t.pairingCompleteData (*C var*), 478, 672
gapPairingCompleteEvent_t.pairingCompleteData.failReason (*C var*), 486, 679
gapPairingCompleteEvent_t.pairingCompleteData.withBonding (*C var*), 486, 679
gapPairingCompleteEvent_t.pairingSuccessful (*C var*), 478, 672
gapPairingParameters_t (*C struct*), 455, 655
gapPairingParameters_t.centralKeys (*C var*), 456, 655
gapPairingParameters_t.leSecureConnectionSupported (*C var*), 456, 655
gapPairingParameters_t.localIoCapabilities (*C var*), 456, 655
gapPairingParameters_t.maxEncryptionKeySize (*C var*), 456, 655
gapPairingParameters_t.oobAvailable (*C var*), 456, 655
gapPairingParameters_t.peripheralKeys (*C var*), 456, 655
gapPairingParameters_t.securityModeAndLevel (*C var*), 455, 655
gapPairingParameters_t.useKeypressNotifications (*C var*), 456, 655
gapPairingParameters_t.withBonding (*C var*), 455, 655
gapPathLossReportingParams_t (*C struct*), 467, 663
gapPathLossReportingParams_t.highHysteresis (*C var*), 467, 663
gapPathLossReportingParams_t.highThreshold (*C var*), 467, 663
gapPathLossReportingParams_t.lowHysteresis (*C var*), 467, 663
gapPathLossReportingParams_t.lowThreshold (*C var*), 467, 663
gapPathLossReportingParams_t.minTimeSpent (*C var*), 467, 663
gapPathLossThresholdEvent_t (*C struct*), 481, 674
gapPathLossThresholdEvent_t.currentPathLoss (*C var*), 481, 674
gapPathLossThresholdEvent_t.zoneEntered (*C var*), 481, 674
gapPerAdvResponse_t (*C struct*), 471, 666
gapPerAdvResponse_t.aData (*C var*), 666
gapPerAdvResponse_t.advHandle (*C var*), 666
gapPerAdvResponse_t.cteType (*C var*), 666
gapPerAdvResponse_t.dataLength (*C var*), 666
gapPerAdvResponse_t.responseSlot (*C var*), 666
gapPerAdvResponse_t.rssi (*C var*), 666
gapPerAdvResponse_t.subevent (*C var*), 666
gapPerAdvResponse_t.txPower (*C var*), 666
gapPerAdvSubeventDataRequest_t (*C struct*), 471, 666
gapPerAdvSubeventDataRequest_t.handle (*C var*), 471, 666
gapPerAdvSubeventDataRequest_t.subeventDataCount (*C var*), 471, 666

gapPerAdvSubeventDataRequest_t.subeventStart (*C var*), 471, 666
gapPeriodicAdvertisingResponseData_t (*C struct*), 469, 664
gapPeriodicAdvertisingResponseData_t.pResponseData (*C var*), 469, 665
gapPeriodicAdvertisingResponseData_t.requestEvent (*C var*), 469, 665
gapPeriodicAdvertisingResponseData_t.requestSubevent (*C var*), 469, 665
gapPeriodicAdvertisingResponseData_t.responseSlot (*C var*), 469, 665
gapPeriodicAdvertisingResponseData_t.responseSubevent (*C var*), 469, 665
gapPeriodicAdvertisingSubeventData_t (*C struct*), 469, 664
gapPeriodicAdvertisingSubeventData_t.aSubeventDataStructures (*C var*), 469, 664
gapPeriodicAdvertisingSubeventData_t.cNumSubevents (*C var*), 469, 664
gapPeriodicAdvListOperation_t (*C enum*), 386
gapPeriodicAdvListOperation_t.gAddDevice_c (*C enumerator*), 386
gapPeriodicAdvListOperation_t.gRemoveAllDevices_c (*C enumerator*), 386
gapPeriodicAdvListOperation_t.gRemoveDevice_c (*C enumerator*), 386
gapPeriodicAdvParameters_t (*C type*), 389
gapPeriodicAdvParameters_tag (*C struct*), 459, 657
gapPeriodicAdvParameters_tag.addTxPowerInAdv (*C var*), 459, 657
gapPeriodicAdvParameters_tag.handle (*C var*), 459, 657
gapPeriodicAdvParameters_tag.maxInterval (*C var*), 459, 657
gapPeriodicAdvParameters_tag.minInterval (*C var*), 459, 657
gapPeriodicAdvParameters_tag.numResponseSlots (*C var*), 459, 657
gapPeriodicAdvParameters_tag.numSubevents (*C var*), 459, 657
gapPeriodicAdvParameters_tag.responseSlotDelay (*C var*), 459, 657
gapPeriodicAdvParameters_tag.responseSlotSpacing (*C var*), 459, 657
gapPeriodicAdvParameters_tag.subeventInterval (*C var*), 459, 657
gapPeriodicAdvSetInfoTransfer_t (*C type*), 389
gapPeriodicAdvSetInfoTransfer_tag (*C struct*), 460, 658
gapPeriodicAdvSetInfoTransfer_tag.advHandle (*C var*), 460, 658
gapPeriodicAdvSetInfoTransfer_tag.deviceId (*C var*), 460, 658
gapPeriodicAdvSetInfoTransfer_tag.serviceData (*C var*), 460, 658
gapPeriodicAdvSyncMode_t (*C type*), 389
gapPeriodicAdvSyncMode_tag (*C enum*), 373
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncNoReports_c (*C enumerator*), 373
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncNoSyncMode_c (*C enumerator*), 373
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncReportsEnabled_c (*C enumerator*), 373
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncReportsEnabledWithDF_c (*C enumerator*), 373
gapPeriodicAdvSyncReq_t (*C type*), 389
gapPeriodicAdvSyncReq_tag (*C struct*), 461, 659
gapPeriodicAdvSyncReq_tag.cteType (*C var*), 462, 659
gapPeriodicAdvSyncReq_tag.options (*C var*), 461, 659
gapPeriodicAdvSyncReq_tag.peerAddress (*C var*), 461, 659
gapPeriodicAdvSyncReq_tag.peerAddressType (*C var*), 461, 659
gapPeriodicAdvSyncReq_tag.SID (*C var*), 461, 659
gapPeriodicAdvSyncReq_tag.skipCount (*C var*), 461, 659
gapPeriodicAdvSyncReq_tag.timeout (*C var*), 461, 659
gapPeriodicAdvSyncTransfer_t (*C type*), 389
gapPeriodicAdvSyncTransfer_tag (*C struct*), 459, 657
gapPeriodicAdvSyncTransfer_tag.deviceId (*C var*), 459, 657
gapPeriodicAdvSyncTransfer_tag.serviceData (*C var*), 459, 657
gapPeriodicAdvSyncTransfer_tag.syncHandle (*C var*), 459, 657
gapPeriodicScannedDevice_t (*C struct*), 473, 668
gapPeriodicScannedDevice_t.cteType (*C var*), 473, 668
gapPeriodicScannedDevice_t.dataLength (*C var*), 474, 668
gapPeriodicScannedDevice_t.pData (*C var*), 474, 668
gapPeriodicScannedDevice_t.rssi (*C var*), 473, 668
gapPeriodicScannedDevice_t.syncHandle (*C var*), 473, 668
gapPeriodicScannedDevice_t.txPower (*C var*), 473, 668
gapPeriodicScannedDeviceV2_t (*C struct*), 474, 668

gapPeriodicScannedDeviceV2_t.cteType (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.dataLength (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.pData (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.periodicEventCounter (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.rssi (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.subevent (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.syncHandle (*C var*), 474, 668
gapPeriodicScannedDeviceV2_t.txPower (*C var*), 474, 668
gapPeriodicSyncSubeventParameters_t (*C struct*), 469, 665
gapPeriodicSyncSubeventParameters_t.aSubevents (*C var*), 470, 665
gapPeriodicSyncSubeventParameters_t.numSubevents (*C var*), 470, 665
gapPeriodicSyncSubeventParameters_t.perAdvProperties (*C var*), 470, 665
gapPeripheralSecurityRequestParameters_t (*C struct*), 456, 655
gapPeripheralSecurityRequestParameters_t.authenticationRequired (*C var*), 456, 655
gapPeripheralSecurityRequestParameters_t.bondAfterPairing (*C var*), 456, 655
gapPhyEvent_t (*C struct*), 602, 636
gapPhyEvent_t.deviceId (*C var*), 636
gapPhyEvent_t.phyEventType (*C var*), 636
gapPhyEvent_t.rxPhy (*C var*), 637
gapPhyEvent_t.txPhy (*C var*), 637
gapPhyEventType_t (*C enum*), 576
gapPhyEventType_t.gPhyRead_c (*C enumerator*), 577
gapPhyEventType_t.gPhySetDefaultComplete_c (*C enumerator*), 577
gapPhyEventType_t.gPhyUpdateComplete_c (*C enumerator*), 577
gapPrivateKeyType_t (*C enum*), 374
gapPrivateKeyType_t.gUseDebugKey_c (*C enumerator*), 374
gapPrivateKeyType_t.gUseGeneratedKey_c (*C enumerator*), 374
gapRadioPowerLevelReadType_t (*C enum*), 378
gapRadioPowerLevelReadType_t.gRssi_c (*C enumerator*), 379
gapRadioPowerLevelReadType_t.gTxPowerCurrentLevelInConnection_c (*C enumerator*), 378
gapRadioPowerLevelReadType_t.gTxPowerLevelForAdvertising_c (*C enumerator*), 379
gapRadioPowerLevelReadType_t.gTxPowerMaximumLevelInConnection_c (*C enumerator*), 379
gapRemoteVersionInfoRead_t (*C type*), 584
gapRemoteVersionInfoRead_tag (*C struct*), 615, 647
gapRemoteVersionInfoRead_tag.deviceId (*C var*), 647
gapRemoteVersionInfoRead_tag.manufacturerName (*C var*), 647
gapRemoteVersionInfoRead_tag.status (*C var*), 647
gapRemoteVersionInfoRead_tag.subversion (*C var*), 647
gapRemoteVersionInfoRead_tag.version (*C var*), 647
gapRole_t (*C enum*), 371
gapRole_t.gGapBroadcaster_c (*C enumerator*), 371
gapRole_t.gGapCentral_c (*C enumerator*), 371
gapRole_t.gGapObserver_c (*C enumerator*), 371
gapRole_t.gGapPeripheral_c (*C enumerator*), 371
gapScanMode_t (*C enum*), 372
gapScanMode_t.gAutoConnect_c (*C enumerator*), 372
gapScanMode_t.gDefaultScan_c (*C enumerator*), 372
gapScanMode_t.gGeneralDiscovery_c (*C enumerator*), 372
gapScanMode_t.gLimitedDiscovery_c (*C enumerator*), 372
gapScannedDevice_t (*C struct*), 471, 666
gapScannedDevice_t.aAddress (*C var*), 471, 667
gapScannedDevice_t.addressType (*C var*), 471, 667
gapScannedDevice_t.advertisingAddressResolved (*C var*), 472, 667
gapScannedDevice_t.advEventType (*C var*), 472, 667
gapScannedDevice_t.data (*C var*), 472, 667
gapScannedDevice_t.dataLength (*C var*), 471, 667
gapScannedDevice_t.directRpa (*C var*), 472, 667
gapScannedDevice_t.directRpaUsed (*C var*), 472, 667

gapScannedDevice_t.rssi (*C var*), 471, 667
gapScanningCallback_t (*C type*), 389
gapScanningEvent_t (*C struct*), 476, 670
gapScanningEvent_t.eventData (*C union*), 485, 678
gapScanningEvent_t.eventData (*C var*), 477, 670
gapScanningEvent_t.eventData.extScannedDevice (*C var*), 485, 678
gapScanningEvent_t.eventData.failReason (*C var*), 485, 678
gapScanningEvent_t.eventData.iqReport (*C var*), 486, 679
gapScanningEvent_t.eventData.monAdvReport (*C var*), 486, 679
gapScanningEvent_t.eventData.periodicScannedDevice (*C var*), 485, 679
gapScanningEvent_t.eventData.periodicScannedDeviceV2 (*C var*), 486, 679
gapScanningEvent_t.eventData.scannedDevice (*C var*), 485, 678
gapScanningEvent_t.eventData.syncEstb (*C var*), 486, 679
gapScanningEvent_t.eventData.syncLost (*C var*), 486, 679
gapScanningEvent_t.eventType (*C var*), 477, 670
gapScanningEventType_t (*C enum*), 380
gapScanningEventType_t.gConnectionlessIqReportReceived_c (*C enumerator*), 381
gapScanningEventType_t.gDeviceScanned_c (*C enumerator*), 381
gapScanningEventType_t.gExtDeviceScanned_c (*C enumerator*), 381
gapScanningEventType_t.gMonAdvReportEventReceived_c (*C enumerator*), 381
gapScanningEventType_t.gPeriodicAdvSyncEstablished_c (*C enumerator*), 381
gapScanningEventType_t.gPeriodicAdvSyncLost_c (*C enumerator*), 381
gapScanningEventType_t.gPeriodicAdvSyncTerminated_c (*C enumerator*), 381
gapScanningEventType_t.gPeriodicDeviceScanned_c (*C enumerator*), 381
gapScanningEventType_t.gPeriodicDeviceScannedV2_c (*C enumerator*), 381
gapScanningEventType_t.gScanCommandFailed_c (*C enumerator*), 380
gapScanningEventType_t.gScanStateChanged_c (*C enumerator*), 380
gapScanningParameters_t (*C struct*), 460, 658
gapScanningParameters_t.filterPolicy (*C var*), 461, 658
gapScanningParameters_t.interval (*C var*), 461, 658
gapScanningParameters_t.ownAddressType (*C var*), 461, 658
gapScanningParameters_t.scanningPHYs (*C var*), 461, 658
gapScanningParameters_t.type (*C var*), 461, 658
gapScanningParameters_t.window (*C var*), 461, 658
gapScanResponseData_t (*C type*), 389
gapSecurityLevel_t (*C type*), 388
gapSecurityMode_t (*C type*), 388
gapSecurityModeAndLevel_t (*C type*), 388
gapSecurityRequirements_t (*C struct*), 454, 654
gapSecurityRequirements_t.authorization (*C var*), 454, 654
gapSecurityRequirements_t.minimumEncryptionKeySize (*C var*), 454, 654
gapSecurityRequirements_t.securityModeLevel (*C var*), 454, 654
gapServiceSecurityRequirements_t (*C struct*), 454, 654
gapServiceSecurityRequirements_t.requirements (*C var*), 454, 654
gapServiceSecurityRequirements_t.serviceHandle (*C var*), 454, 654
gapSetPeriodicAdvSyncTransferParams_t (*C type*), 389
gapSetPeriodicAdvSyncTransferParams_tag (*C struct*), 460, 658
gapSetPeriodicAdvSyncTransferParams_tag.CTEType (*C var*), 460, 658
gapSetPeriodicAdvSyncTransferParams_tag.deviceId (*C var*), 460, 658
gapSetPeriodicAdvSyncTransferParams_tag.mode (*C var*), 460, 658
gapSetPeriodicAdvSyncTransferParams_tag.skip (*C var*), 460, 658
gapSetPeriodicAdvSyncTransferParams_tag.syncTimeout (*C var*), 460, 658
gapSleepClockAccuracy_t (*C enum*), 380
gapSleepClockAccuracy_t.gSwitchToLessAccurateClock_c (*C enumerator*), 380
gapSleepClockAccuracy_t.gSwitchToMoreAccurateClock_c (*C enumerator*), 380
gapSmpKeyFlags_t (*C type*), 388
gapSmpKeys_t (*C struct*), 453, 653
gapSmpKeys_t.aAddress (*C var*), 454, 654

gapSmpKeys_t.aCsrk (*C var*), 453, 653
gapSmpKeys_t.addressType (*C var*), 454, 654
gapSmpKeys_t.aIrk (*C var*), 453, 653
gapSmpKeys_t.aLtk (*C var*), 453, 653
gapSmpKeys_t.aRand (*C var*), 454, 653
gapSmpKeys_t.cLtkSize (*C var*), 453, 653
gapSmpKeys_t.cRandSize (*C var*), 453, 653
gapSmpKeys_t.ediv (*C var*), 454, 654
gapSubeventDataStructure_t (*C struct*), 468, 664
gapSubeventDataStructure_t.pAdvertisingData (*C var*), 469, 664
gapSubeventDataStructure_t.responseSlotCount (*C var*), 469, 664
gapSubeventDataStructure_t.responseSlotStart (*C var*), 469, 664
gapSubeventDataStructure_t.subevent (*C var*), 469, 664
gapSyncEstbEventData_t (*C struct*), 474, 669
gapSyncEstbEventData_t.advertiserClockAccuracy (*C var*), 475, 669
gapSyncEstbEventData_t.numSubevents (*C var*), 475, 669
gapSyncEstbEventData_t.peerAddress (*C var*), 475, 669
gapSyncEstbEventData_t.peerAddressType (*C var*), 475, 669
gapSyncEstbEventData_t.periodicAdvInterval (*C var*), 475, 669
gapSyncEstbEventData_t.PHY (*C var*), 475, 669
gapSyncEstbEventData_t.responseSlotDelay (*C var*), 475, 669
gapSyncEstbEventData_t.responseSlotSpacing (*C var*), 475, 669
gapSyncEstbEventData_t.SID (*C var*), 475, 669
gapSyncEstbEventData_t.status (*C var*), 474, 669
gapSyncEstbEventData_t.subeventInterval (*C var*), 475, 669
gapSyncEstbEventData_t.syncHandle (*C var*), 474, 669
gapSyncLostEventData_t (*C struct*), 475, 669
gapSyncLostEventData_t.syncHandle (*C var*), 475, 669
gapSyncTransferReceivedEventData_t (*C type*), 583
gapSyncTransferReceivedEventData_tag (*C struct*), 604, 639
gapSyncTransferReceivedEventData_tag.advAddress (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.advAddressType (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.advClockAccuracy (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.advPhy (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.advSID (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.deviceId (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.numSubevents (*C var*), 605, 640
gapSyncTransferReceivedEventData_tag.periodicAdvInt (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.responseSlotDelay (*C var*), 605, 640
gapSyncTransferReceivedEventData_tag.responseSlotSpacing (*C var*), 605, 640
gapSyncTransferReceivedEventData_tag.serviceData (*C var*), 605, 639
gapSyncTransferReceivedEventData_tag.status (*C var*), 639
gapSyncTransferReceivedEventData_tag.subeventInterval (*C var*), 605, 640
gapSyncTransferReceivedEventData_tag.syncHandle (*C var*), 605, 639
gapTransmitPowerInfo_t (*C struct*), 482, 674
gapTransmitPowerInfo_t.currTxPowerLevel (*C var*), 482, 674
gapTransmitPowerInfo_t.maxTxPowerLevel (*C var*), 482, 674
gapTransmitPowerInfo_t.phy (*C var*), 482, 674
gapTransmitPowerReporting_t (*C struct*), 482, 674
gapTransmitPowerReporting_t.delta (*C var*), 482, 674
gapTransmitPowerReporting_t.flags (*C var*), 482, 674
gapTransmitPowerReporting_t.phy (*C var*), 482, 674
gapTransmitPowerReporting_t.reason (*C var*), 482, 674
gapTransmitPowerReporting_t.txPowerLevel (*C var*), 482, 674
Gatt_GetMtu (*C function*), 492
Gatt_Init (*C function*), 492
gattAttribute_t (*C struct*), 493, 681
gattAttribute_t.handle (*C var*), 493, 681

gattAttribute_t.maxValueLength (*C var*), 493, 681
gattAttribute_t.paValue (*C var*), 493, 681
gattAttribute_t.uuid (*C var*), 493, 681
gattAttribute_t.uuidType (*C var*), 493, 681
gattAttribute_t.valueLength (*C var*), 493, 681
gattAttributePermissionsBitFields_t (*C type*), 533
gattCachingClientState_c (*C enum*), 490
gattCachingClientState_c.gGattClientChangeAware_c (*C enumerator*), 490
gattCachingClientState_c.gGattClientChangeUnaware_c (*C enumerator*), 490
gattCachingClientState_c.gGattClientStateChangePending_c (*C enumerator*), 490
gattCccdFlags_t (*C type*), 492
gattCharacteristic_t (*C struct*), 493, 681
gattCharacteristic_t.aDescriptors (*C var*), 494, 681
gattCharacteristic_t.cNumDescriptors (*C var*), 493, 681
gattCharacteristic_t.properties (*C var*), 493, 681
gattCharacteristic_t.value (*C var*), 493, 681
gattCharacteristicPropertiesBitFields_t (*C type*), 533
gattCharacteristicPropertiesBitFields_tag (*C enum*), 532
gattCharacteristicPropertiesBitFields_tag.gGattCharPropAuthSignedWrites_c (*C enumerator*), 533
gattCharacteristicPropertiesBitFields_tag.gGattCharPropBroadcast_c (*C enumerator*), 532
gattCharacteristicPropertiesBitFields_tag.gGattCharPropExtendedProperties_c (*C enumerator*), 533
gattCharacteristicPropertiesBitFields_tag.gGattCharPropIndicate_c (*C enumerator*), 533
gattCharacteristicPropertiesBitFields_tag.gGattCharPropNone_c (*C enumerator*), 532
gattCharacteristicPropertiesBitFields_tag.gGattCharPropNotify_c (*C enumerator*), 533
gattCharacteristicPropertiesBitFields_tag.gGattCharPropRead_c (*C enumerator*), 532
gattCharacteristicPropertiesBitFields_tag.gGattCharPropWrite_c (*C enumerator*), 533
gattCharacteristicPropertiesBitFields_tag.gGattCharPropWriteWithoutRsp_c (*C enumerator*), 532
GattClient_CharacteristicSignedWrite (*C macro*), 518
GattClient_CharacteristicWriteWithoutResponse (*C macro*), 518
GattClient_DiscoverAllCharacteristicDescriptors (*C function*), 503
GattClient_DiscoverAllCharacteristicsOfService (*C function*), 502
GattClient_DiscoverAllPrimaryServices (*C function*), 501
GattClient_DiscoverCharacteristicOfServiceByUuid (*C function*), 503
GattClient_DiscoverPrimaryServicesByUuid (*C function*), 501
GattClient_EnhancedDiscoverAllCharacteristicDescriptors (*C function*), 513
GattClient_EnhancedDiscoverAllCharacteristicsOfService (*C function*), 511
GattClient_EnhancedDiscoverAllPrimaryServices (*C function*), 510
GattClient_EnhancedDiscoverCharacteristicOfServiceByUuid (*C function*), 512
GattClient_EnhancedDiscoverPrimaryServicesByUuid (*C function*), 510
GattClient_EnhancedFindIncludedServices (*C function*), 511
GattClient_EnhancedReadCharacteristicDescriptor (*C function*), 516
GattClient_EnhancedReadCharacteristicValue (*C function*), 513
GattClient_EnhancedReadMultipleCharacteristicValues (*C function*), 515
GattClient_EnhancedReadMultipleVariableCharacteristicValues (*C function*), 517
GattClient_EnhancedReadUsingCharacteristicUuid (*C function*), 514
GattClient_EnhancedWriteCharacteristicDescriptor (*C function*), 517
GattClient_EnhancedWriteCharacteristicValue (*C function*), 515
GattClient_ExchangeMtu (*C function*), 500
GattClient_FindIncludedServices (*C function*), 502
GattClient_Init (*C function*), 498
GattClient_ReadCharacteristicDescriptor (*C function*), 507
GattClient_ReadCharacteristicValue (*C function*), 504
GattClient_ReadMultipleCharacteristicValues (*C function*), 505
GattClient_ReadMultipleVariableCharacteristicValues (*C function*), 508
GattClient_ReadUsingCharacteristicUuid (*C function*), 505
GattClient_RegisterEnhancedIndicationCallback (*C function*), 509
GattClient_RegisterEnhancedMultipleValueNotificationCallback (*C function*), 509
GattClient_RegisterEnhancedNotificationCallback (*C function*), 508

GattClient_RegisterEnhancedProcedureCallback (*C function*), 508
GattClient_RegisterIndicationCallback (*C function*), 499
GattClient_RegisterMultipleValueNotificationCallback (*C function*), 500
GattClient_RegisterNotificationCallback (*C function*), 499
GattClient_RegisterProcedureCallback (*C function*), 499
GattClient_ResetProcedure (*C function*), 499
GattClient_SimpleCharacteristicWrite (*C macro*), 518
GattClient_WriteCharacteristicDescriptor (*C function*), 507
GattClient_WriteCharacteristicValue (*C function*), 506
gattClientEnhancedIndicationCallback_t (*C type*), 498
gattClientEnhancedMultipleValueNotificationCallback_t (*C type*), 498
gattClientEnhancedNotificationCallback_t (*C type*), 498
gattClientEnhancedProcedureCallback_t (*C type*), 498
gattClientHashUpdateType_t (*C enum*), 490
gattClientHashUpdateType_t.gattClientActiveConnectionUpdate (*C enumerator*), 490
gattClientHashUpdateType_t.gattClientFirstConnection_c (*C enumerator*), 490
gattClientHashUpdateType_t.gattClientNoChange (*C enumerator*), 490
gattClientHashUpdateType_t.gattClientReconnectBondedPeer_c (*C enumerator*), 490
gattClientIndicationCallback_t (*C type*), 498
gattClientMultipleValueNotificationCallback_t (*C type*), 498
gattClientNotificationCallback_t (*C type*), 498
gattClientProcedureCallback_t (*C type*), 498
gattDatabase (*C var*), 533
GattDb_ComputeDatabaseHash (*C function*), 534
GattDb_Deinit (*C function*), 535
GattDb_FindCccdHandleForCharValueHandle (*C function*), 538
GattDb_FindCharValueHandleInService (*C function*), 537
GattDb_FindDescriptorHandleForCharValueHandle (*C function*), 538
GattDb_FindServiceHandle (*C function*), 536
GattDb_FindServiceRange (*C function*), 534
GattDb_GetAttributeValueSize (*C function*), 534
GattDb_GetIndexOfHandle (*C function*), 534
GattDb_Init (*C function*), 535
GattDb_ReadAttribute (*C function*), 536
GattDb_ServiceStartHandle (*C function*), 534
GattDb_WriteAttribute (*C function*), 535
gattDbAccessType_t (*C enum*), 533
gattDbAccessType_t.gAccessNotify_c (*C enumerator*), 533
gattDbAccessType_t.gAccessRead_c (*C enumerator*), 533
gattDbAccessType_t.gAccessWrite_c (*C enumerator*), 533
gattDbAttribute_t (*C struct*), 539, 686
gattDbAttribute_t.handle (*C var*), 540, 686
gattDbAttribute_t.maxVariableValueLength (*C var*), 540, 687
gattDbAttribute_t.permissions (*C var*), 540, 686
gattDbAttribute_t.pValue (*C var*), 540, 686
gattDbAttribute_t.uuid (*C var*), 540, 686
gattDbAttribute_t.uuidType (*C var*), 540, 687
gattDbAttribute_t.valueLength (*C var*), 540, 686
gattDbCharPresFormat_t (*C struct*), 494, 682
gattDbCharPresFormat_t.description (*C var*), 495, 682
gattDbCharPresFormat_t.exponent (*C var*), 494, 682
gattDbCharPresFormat_t.format (*C var*), 494, 682
gattDbCharPresFormat_t.ns (*C var*), 495, 682
gattDbCharPresFormat_t.unitUuid16 (*C var*), 494, 682
gAttDefaultMtu_c (*C macro*), 622
gattHandleRange_t (*C struct*), 495, 682
gattHandleRange_t.endHandle (*C var*), 495, 682
gattHandleRange_t.startHandle (*C var*), 495, 682

gAttMaxMtu_c (*C macro*), 622
gAttMaxValueLength_c (*C macro*), 623
gattProcedurePhase_t (*C enum*), 492
gattProcedurePhase_t.gattProcPhaseInitiated (*C enumerator*), 492
gattProcedurePhase_t.gattProcPhaseRunning (*C enumerator*), 492
gattProcedureResult_t (*C enum*), 497
gattProcedureResult_t.gGattProcError_c (*C enumerator*), 498
gattProcedureResult_t.gGattProcSuccess_c (*C enumerator*), 497
gattProcedureType_t (*C enum*), 491
gattProcedureType_t.gGattProcDiscoverAllCharacteristicDescriptors_c (*C enumerator*), 491
gattProcedureType_t.gGattProcDiscoverAllCharacteristics_c (*C enumerator*), 491
gattProcedureType_t.gGattProcDiscoverAllPrimaryServices_c (*C enumerator*), 491
gattProcedureType_t.gGattProcDiscoverCharacteristicByUuid_c (*C enumerator*), 491
gattProcedureType_t.gGattProcDiscoverPrimaryServicesByUuid_c (*C enumerator*), 491
gattProcedureType_t.gGattProcExchangeMtu_c (*C enumerator*), 491
gattProcedureType_t.gGattProcFindIncludedServices_c (*C enumerator*), 491
gattProcedureType_t.gGattProcReadCharacteristicDescriptor_c (*C enumerator*), 491
gattProcedureType_t.gGattProcReadCharacteristicValue_c (*C enumerator*), 491
gattProcedureType_t.gGattProcReadMultipleCharacteristicValues_c (*C enumerator*), 491
gattProcedureType_t.gGattProcReadMultipleVarLengthCharValues_c (*C enumerator*), 492
gattProcedureType_t.gGattProcReadUsingCharacteristicUuid_c (*C enumerator*), 491
gattProcedureType_t.gGattProcSignalServiceDiscoveryComplete_c (*C enumerator*), 491
gattProcedureType_t.gGattProcUpdateDatabaseCopy_c (*C enumerator*), 491
gattProcedureType_t.gGattProcWriteCharacteristicDescriptor_c (*C enumerator*), 491
gattProcedureType_t.gGattProcWriteCharacteristicValue_c (*C enumerator*), 491
GattServer_EnhancedSendAttributeReadStatus (*C function*), 527
GattServer_EnhancedSendAttributeWrittenStatus (*C function*), 527
GattServer_EnhancedSendIndication (*C function*), 528
GattServer_EnhancedSendInstantValueIndication (*C function*), 529
GattServer_EnhancedSendInstantValueNotification (*C function*), 528
GattServer_EnhancedSendMultipleHandleValueNotification (*C function*), 529
GattServer_EnhancedSendNotification (*C function*), 528
GattServer_Init (*C function*), 520
GattServer_RegisterCallback (*C function*), 521
GattServer_RegisterEnhancedCallback (*C function*), 526
GattServer_RegisterHandlesForReadNotifications (*C function*), 522
GattServer_RegisterHandlesForWriteNotifications (*C function*), 521
GattServer_RegisterUniqueHandlesForNotifications (*C function*), 526
GattServer_SendAttributeReadStatus (*C function*), 524
GattServer_SendAttributeWrittenStatus (*C function*), 522
GattServer_SendIndication (*C function*), 524
GattServer_SendInstantValueIndication (*C function*), 525
GattServer_SendInstantValueNotification (*C function*), 525
GattServer_SendMultipleHandleValueNotification (*C function*), 525
GattServer_SendNotification (*C function*), 524
GattServer_UnregisterHandlesForReadNotifications (*C function*), 523
GattServer_UnregisterHandlesForWriteNotifications (*C function*), 521
gattServerAttributeReadEvent_t (*C struct*), 531, 685
gattServerAttributeReadEvent_t.handle (*C var*), 531, 685
gattServerAttributeWrittenEvent_t (*C struct*), 530, 684
gattServerAttributeWrittenEvent_t.aValue (*C var*), 530, 685
gattServerAttributeWrittenEvent_t.bearerId (*C var*), 530, 685
gattServerAttributeWrittenEvent_t.cValueLength (*C var*), 530, 685
gattServerAttributeWrittenEvent_t.handle (*C var*), 530, 685
gattServerCallback_t (*C type*), 520
gattServerCccdWrittenEvent_t (*C struct*), 531, 685
gattServerCccdWrittenEvent_t.handle (*C var*), 531, 685
gattServerCccdWrittenEvent_t.newCccd (*C var*), 531, 685

gattServerEnhancedCallback_t (*C type*), 520
gattServerEvent_t (*C struct*), 531, 686
gattServerEvent_t.eventData (*C union*), 532, 686
gattServerEvent_t.eventData (*C var*), 532, 686
gattServerEvent_t.eventData.attributeOpCode (*C var*), 532, 686
gattServerEvent_t.eventData.attributeReadEvent (*C var*), 532, 686
gattServerEvent_t.eventData.attributeWrittenEvent (*C var*), 532, 686
gattServerEvent_t.eventData.charCccdWrittenEvent (*C var*), 532, 686
gattServerEvent_t.eventData.longCharWrittenEvent (*C var*), 532, 686
gattServerEvent_t.eventData.mtuChangedEvent (*C var*), 532, 686
gattServerEvent_t.eventData.procedureError (*C var*), 532, 686
gattServerEvent_t.eventType (*C var*), 532, 686
gattServerEventType_t (*C enum*), 519
gattServerEventType_t.gEvtAttributeRead_c (*C enumerator*), 520
gattServerEventType_t.gEvtAttributeWritten_c (*C enumerator*), 519
gattServerEventType_t.gEvtAttributeWrittenWithoutResponse_c (*C enumerator*), 519
gattServerEventType_t.gEvtCharacteristicCccdWritten_c (*C enumerator*), 519
gattServerEventType_t.gEvtError_c (*C enumerator*), 519
gattServerEventType_t.gEvtHandleValueConfirmation_c (*C enumerator*), 519
gattServerEventType_t.gEvtInvalidPduReceived_c (*C enumerator*), 520
gattServerEventType_t.gEvtLongCharacteristicWritten_c (*C enumerator*), 519
gattServerEventType_t.gEvtMtuChanged_c (*C enumerator*), 519
gattServerInvalidPdu_t (*C struct*), 531, 685
gattServerInvalidPdu_t.attOpCode (*C var*), 531, 686
gattServerLongCharacteristicWrittenEvent_t (*C struct*), 530, 685
gattServerLongCharacteristicWrittenEvent_t.aValue (*C var*), 530, 685
gattServerLongCharacteristicWrittenEvent_t.cValueLength (*C var*), 530, 685
gattServerLongCharacteristicWrittenEvent_t.handle (*C var*), 530, 685
gattServerMtuChangedEvent_t (*C struct*), 530, 684
gattServerMtuChangedEvent_t.newMtu (*C var*), 530, 684
gattServerProcedureError_t (*C struct*), 531, 685
gattServerProcedureError_t.error (*C var*), 531, 685
gattServerProcedureError_t.procedureType (*C var*), 531, 685
gattServerProcedureType_t (*C enum*), 520
gattServerProcedureType_t.gSendAttributeReadStatus_c (*C enumerator*), 520
gattServerProcedureType_t.gSendAttributeWrittenStatus_c (*C enumerator*), 520
gattServerProcedureType_t.gSendIndication_c (*C enumerator*), 520
gattServerProcedureType_t.gSendMultipleValNotification_c (*C enumerator*), 520
gattServerProcedureType_t.gSendNotification_c (*C enumerator*), 520
gattService_t (*C type*), 492
gattService_tag (*C struct*), 494, 681
gattService_tag.aCharacteristics (*C var*), 494, 682
gattService_tag.aIncludedServices (*C var*), 494, 682
gattService_tag.cNumCharacteristics (*C var*), 494, 682
gattService_tag.cNumIncludedServices (*C var*), 494, 682
gattService_tag.endHandle (*C var*), 494, 681
gattService_tag.startHandle (*C var*), 494, 681
gattService_tag.uuid (*C var*), 494, 682
gattService_tag.uuidType (*C var*), 494, 681
gBasicFilteredScan_c (*C macro*), 589
gBasicUnfilteredScan_c (*C macro*), 589
gBleAddrTypePublic_c (*C macro*), 588
gBleAddrTypeRandom_c (*C macro*), 588
gBleAdvTxPowerNoPreference_c (*C macro*), 624
gBleBondDataDescriptorSize_c (*C macro*), 621
gBleBondDataDeviceInfoSize_c (*C macro*), 621
gBleBondDataDynamicSize_c (*C macro*), 621
gBleBondDataLegacySize_c (*C macro*), 621

gBleBondDataSize_c (*C macro*), 622
gBleBondDataStaticSize_c (*C macro*), 621
gBleCteMaxTxOctets_c (*C macro*), 623
gBleCteMaxTxTime_c (*C macro*), 623
gBleCteMinTxOctets_c (*C macro*), 623
gBleCteMinTxTime_c (*C macro*), 623
gBleEattMaxConnectionChannels (*C var*), 585
gBleEattPsmMtu (*C var*), 585
gBleExtAdvDefaultSetHandle_c (*C macro*), 624
gBleExtAdvDefaultSetId_c (*C macro*), 623
gBleExtAdvLegacySetHandle_c (*C macro*), 623
gBleExtAdvLegacySetId_c (*C macro*), 623
gBleExtAdvMaxAuxOffsetUsec_c (*C macro*), 624
gBleExtAdvMaxSetId_c (*C macro*), 623
gBleExtAdvNoDuration_c (*C macro*), 624
gBleExtAdvNoMaxEvents_c (*C macro*), 624
gBleHighDutyDirectedAdvDuration (*C macro*), 624
gBleMaxActiveConnections (*C var*), 585
gBleMaxADStructureLength_c (*C macro*), 624
gBleMaxExtAdvDataLength_c (*C macro*), 624
gBleMaxTxOctets_c (*C macro*), 623
gBleMaxTxTime_c (*C macro*), 623
gBleMaxTxTimeCodedPhy_c (*C macro*), 623
gBleMinTxOctets_c (*C macro*), 623
gBleMinTxTime_c (*C macro*), 623
gBleMonAdvRssiThresholdMax_c (*C macro*), 625
gBleMonAdvRssiThresholdMin_c (*C macro*), 625
gBleP256KeyLength_c (*C macro*), 624
gBlePeriodicAdvDefaultHandle_c (*C macro*), 624
gBlePeriodicAdvMaxSyncHandle_c (*C macro*), 623
gBlePeriodicAdvOngoingSyncCancelHandle (*C macro*), 450
gBlePeriodicAdvSkipMax_c (*C macro*), 624
gBlePeriodicAdvSyncTimeoutMax_c (*C macro*), 624
gBlePeriodicAdvSyncTimeoutMin_c (*C macro*), 624
gBleSig_AlertLevel_d (*C macro*), 594
gBleSig_AlertNotifControlPoint_d (*C macro*), 596
gBleSig_AlertNotificationService_d (*C macro*), 593
gBleSig_AlertStatus_d (*C macro*), 596
gBleSig_BatteryLevel_d (*C macro*), 594
gBleSig_BatteryService_d (*C macro*), 592
gBleSig_BloodPressureFeature_d (*C macro*), 596
gBleSig_BloodPressureService_d (*C macro*), 592
gBleSig_BodySensorLocation_d (*C macro*), 596
gBleSig_BootKeyboardInputReport_d (*C macro*), 595
gBleSig_BootKeyboardOutputReport_d (*C macro*), 595
gBleSig_BootMouseInputReport_d (*C macro*), 595
gBleSig_BpEnhancedMeasurement_d (*C macro*), 595
gBleSig_BpMeasurement_d (*C macro*), 595
gBleSig_BpRecord_d (*C macro*), 595
gBleSig_BtpService_d (*C macro*), 593
gBleSig_CAR_NotSupported_d (*C macro*), 599
gBleSig_CAR_Supported_d (*C macro*), 599
gBleSig_CCCD_d (*C macro*), 591
gBleSig_CentralAddressResolution_d (*C macro*), 598
gBleSig_Characteristic_d (*C macro*), 591
gBleSig_CharAggregateFormat_d (*C macro*), 591
gBleSig_CharExtendedProperties_d (*C macro*), 591
gBleSig_CharPresFormatDescriptor_d (*C macro*), 591

gBleSig_CharUserDescription_d (*C macro*), 591
gBleSig_CpControlPoint_d (*C macro*), 598
gBleSig_CpFeature_d (*C macro*), 598
gBleSig_CpMeasurement_d (*C macro*), 597
gBleSig_CpVector_d (*C macro*), 597
gBleSig_CscFeature_d (*C macro*), 597
gBleSig_CscMeasurement_d (*C macro*), 597
gBleSig_CurrentTime_d (*C macro*), 595
gBleSig_CurrentTimeService_d (*C macro*), 592
gBleSig_CyclingPowerService_d (*C macro*), 593
gBleSig_CyclingSpeedAndCadenceService_d (*C macro*), 593
gBleSig_DeviceInformationService_d (*C macro*), 592
gBleSig_EncryptedDataKeyMaterial_d (*C macro*), 599
gBleSig_FirmwareRevisionString_d (*C macro*), 595
gBleSig_GapAppearance_d (*C macro*), 593
gBleSig_GapDeviceName_d (*C macro*), 593
gBleSig_GapPpcp_d (*C macro*), 593
gBleSig_GattClientSupportedFeatures_d (*C macro*), 593
gBleSig_GattDatabaseHash_d (*C macro*), 594
gBleSig_GattSecurityLevels_d (*C macro*), 599
gBleSig_GattServerSupportedFeatures_d (*C macro*), 594
gBleSig_GattServiceChanged_d (*C macro*), 593
gBleSig_GenericAccessProfile_d (*C macro*), 592
gBleSig_GenericAttributeProfile_d (*C macro*), 592
gBleSig_GlucoseFeature_d (*C macro*), 597
gBleSig_GlucoseMeasurement_d (*C macro*), 594
gBleSig_GlucoseMeasurementContext_d (*C macro*), 595
gBleSig_GlucoseService_d (*C macro*), 592
gBleSig_HardwareRevisionString_d (*C macro*), 595
gBleSig_HealthThermometerService_d (*C macro*), 592
gBleSig_HeartRateService_d (*C macro*), 592
gBleSig_HidBootMouseInputReport_d (*C macro*), 596
gBleSig_HidCtrlPoint_d (*C macro*), 596
gBleSig_HidInformation_d (*C macro*), 596
gBleSig_HidService_d (*C macro*), 593
gBleSig_HrControlPoint_d (*C macro*), 596
gBleSig_HrMeasurement_d (*C macro*), 596
gBleSig_HTTP_ControlPoint_d (*C macro*), 598
gBleSig_HTTP_EntityBody_d (*C macro*), 598
gBleSig_HTTP_Headers_d (*C macro*), 598
gBleSig_HTTP_StatusCode_d (*C macro*), 598
gBleSig_HTTPProxyService_d (*C macro*), 593
gBleSig_HTTPS_Security_d (*C macro*), 598
gBleSig_IeeeRcdl_d (*C macro*), 595
gBleSig_ImmediateAlertService_d (*C macro*), 592
gBleSig_Include_d (*C macro*), 591
gBleSig_IntermediateCuffPressure_d (*C macro*), 595
gBleSig_IntermediateTemperature_d (*C macro*), 594
gBleSig_IpsService_d (*C macro*), 593
gBleSig_LinkLossService_d (*C macro*), 592
gBleSig_LnControlPoint_d (*C macro*), 598
gBleSig_LnFeature_d (*C macro*), 598
gBleSig_LocalTimeInformation_d (*C macro*), 594
gBleSig_LocationAndNavigationService_d (*C macro*), 593
gBleSig_LocationAndSpeed_d (*C macro*), 598
gBleSig_ManufacturerNameString_d (*C macro*), 595
gBleSig_MeasurementInterval_d (*C macro*), 594
gBleSig_MeshProvDataIn_d (*C macro*), 599

gBleSig_MeshProvDataOut_d (*C macro*), 599
gBleSig_MeshProvisioningService_d (*C macro*), 599
gBleSig_MeshProxyDataIn_d (*C macro*), 599
gBleSig_MeshProxyDataOut_d (*C macro*), 599
gBleSig_MeshProxyService_d (*C macro*), 599
gBleSig_ModelNumberString_d (*C macro*), 595
gBleSig_Navigation_d (*C macro*), 598
gBleSig_NewAlert_d (*C macro*), 596
gBleSig_NextDSTChangeService_d (*C macro*), 592
gBleSig_PhoneAlertStatusService_d (*C macro*), 592
gBleSig_PlxContMeasurement_d (*C macro*), 597
gBleSig_PlxSCMeasurement_d (*C macro*), 597
gBleSig_PnpId_d (*C macro*), 597
gBleSig_PositionQuality_d (*C macro*), 598
gBleSig_PrimaryService_d (*C macro*), 591
gBleSig_ProtocolMode_d (*C macro*), 597
gBleSig_PulseOximeterFeature_d (*C macro*), 597
gBleSig_PulseOximeterService_d (*C macro*), 593
gBleSig_RaCtrlPoint_d (*C macro*), 597
gBleSig_RangingService_d (*C macro*), 599
gBleSig_RasControlPoint_d (*C macro*), 599
gBleSig_RasFeature_d (*C macro*), 599
gBleSig_RasOnDemandProcData_d (*C macro*), 599
gBleSig_RasprocDataOverwritten_d (*C macro*), 600
gBleSig_RasProcDataReady_d (*C macro*), 599
gBleSig_RasRealTimeProcData_d (*C macro*), 599
gBleSig_ReferenceTimeInformation_d (*C macro*), 594
gBleSig_ReferenceTimeUpdateService_d (*C macro*), 592
gBleSig_Report_d (*C macro*), 596
gBleSig_ResolvablePrivateAddressOnly_d (*C macro*), 598
gBleSig_RingerControlPoint_d (*C macro*), 596
gBleSig_RingerSetting_d (*C macro*), 596
gBleSig_RPAO_Used_d (*C macro*), 599
gBleSig_RscFeature_d (*C macro*), 597
gBleSig_RscMeasurement_d (*C macro*), 597
gBleSig_RunningSpeedAndCadenceService_d (*C macro*), 593
gBleSig_ScanIntervalWindow_d (*C macro*), 597
gBleSig_SCCD_d (*C macro*), 591
gBleSig_ScControlPoint_d (*C macro*), 597
gBleSig_SecondaryService_d (*C macro*), 591
gBleSig_SensorLocation_d (*C macro*), 597
gBleSig_SerialNumberString_d (*C macro*), 595
gBleSig_SoftwareRevisionString_d (*C macro*), 595
gBleSig_SupportedNewAlertCategory_d (*C macro*), 596
gBleSig_SupportedUnreadAlertCategory_d (*C macro*), 596
gBleSig_SystemId_d (*C macro*), 594
gBleSig_Temperature_d (*C macro*), 598
gBleSig_TemperatureMeasurement_d (*C macro*), 594
gBleSig_TemperatureType_d (*C macro*), 594
gBleSig_TimeUpdateControlPoint_d (*C macro*), 594
gBleSig_TimeUpdateState_d (*C macro*), 594
gBleSig_TimeWithDST_d (*C macro*), 594
gBleSig_TxPower_d (*C macro*), 594
gBleSig_TxPowerService_d (*C macro*), 592
gBleSig_UnreadAlertStatus_d (*C macro*), 596
gBleSig_URI_d (*C macro*), 598
gBleSig_ValidRangeDescriptor_d (*C macro*), 592
gBleSig_WPTService_d (*C macro*), 593

gBleUuidType16_c (*C macro*), 588
gBleUuidType32_c (*C macro*), 588
gBleUuidType128_c (*C macro*), 588
gbmpDIAM_ConnectableUndirected_c (*C macro*), 453
gbmpDIAM_NonConnectableNonScannableUndirected_c (*C macro*), 453
gbmpDIAM_ScannableUndirected_c (*C macro*), 453
gBrEdrNotSupported_c (*C macro*), 450
gCancelOngoingInitiatingConnection_d (*C macro*), 447
gcBleChannelMapSize_c (*C macro*), 623
gcBleDeviceAddressSize_c (*C macro*), 621
gcBleLongUuidSize_c (*C macro*), 622
gCccdEmpty_c (*C macro*), 493
gCccdIndication_c (*C macro*), 493
gCccdNotification_c (*C macro*), 493
gcConnectionEventMaxDefault_c (*C var*), 585
gcConnectionEventMinDefault_c (*C var*), 585
gcConnectionIntervalMax_c (*C macro*), 587
gcConnectionIntervalMaxDefault_c (*C macro*), 587
gcConnectionIntervalMin_c (*C macro*), 587
gcConnectionIntervalMinDefault_c (*C macro*), 587
gcConnectionPeripheralLatencyDefault_c (*C macro*), 587
gcConnectionPeripheralLatencyMax_c (*C macro*), 587
gcConnectionSupervisionTimeoutDefault_c (*C macro*), 587
gcConnectionSupervisionTimeoutMax_c (*C macro*), 587
gcConnectionSupervisionTimeoutMin_c (*C macro*), 587
gcDecisionDataKeySize_c (*C macro*), 624
gcDecisionDataMaxSize_c (*C macro*), 625
gcDecisionDataPrandSize_c (*C macro*), 625
gcDecisionDataResolvableTagSize_c (*C macro*), 625
gcDecisionInstructionsArbitraryDataMaskSize_c (*C macro*), 624
gcDecisionInstructionsArbitraryDataTargetSize_c (*C macro*), 625
gcDecisionInstructionsParamSize_c (*C macro*), 624
gcEadAadValue_c (*C macro*), 625
gcEadIvSize_c (*C macro*), 625
gcEadKeySize_c (*C macro*), 625
gcEadMicSize_c (*C macro*), 625
gcEadRandomizerSize_c (*C macro*), 625
gcGapMaxAdvertisingDataLength_c (*C macro*), 622
gcGapMaxAuthorizationHandles_c (*C macro*), 621
gcGapMaxDeviceNameSize_c (*C macro*), 622
gcGapMaximumSavedCccds_c (*C macro*), 621
gcMaxNumTestsInDecisionInstruction_c (*C macro*), 625
gcReservedFlashSizeForCustomInformation_c (*C macro*), 623
gCsAccessAdressSize_c (*C macro*), 590
gcSecureModeSavedLocalKeysNo_c (*C macro*), 621
gCSMaxSubeventLen_c (*C macro*), 590
gcSmpCsrkSize_c (*C macro*), 622
gcSmpIrkSize_c (*C macro*), 622
gcSmpMaxBlobSize_c (*C macro*), 622
gcSmpMaxIrkBlobSize_c (*C macro*), 622
gcSmpMaxLtkSize_c (*C macro*), 622
gcSmpMaxRandSize_c (*C macro*), 622
gcSmpOobSize_c (*C macro*), 622
gCSNumPhysMax_c (*C macro*), 590
gCSNumToneAntennaIds_c (*C macro*), 590
gCSReflectorTableSize_c (*C macro*), 590
gCsrk_c (*C macro*), 451
gCSSyncRandomSize_c (*C macro*), 590

gCSTestMaxChannelLength_c (*C macro*), 590
gDefaultEncryptionKeySize_d (*C macro*), 448
gDevicePrivacy_c (*C macro*), 590
gDHKeySize_c (*C macro*), 590
gDRBGKeySize_c (*C macro*), 590
gEattMaxMtu_c (*C macro*), 622
gEattMinMtu_c (*C macro*), 623
gEnhancedL2capMinimumMps_c (*C macro*), 633
getConnParams_t (*C type*), 583
getConnParams_tag (*C struct*), 605, 640
getConnParams_tag.aChMapBm (*C var*), 606, 640
getConnParams_tag.aCrcInitVal (*C var*), 606, 640
getConnParams_tag.aucRemoteMasRxPHY (*C var*), 606, 640
getConnParams_tag.connectionHandle (*C var*), 606, 640
getConnParams_tag.seqNum (*C var*), 606, 640
getConnParams_tag.ucCentralSCA (*C var*), 606, 640
getConnParams_tag.ucChannelSelection (*C var*), 606, 640
getConnParams_tag.ucHop (*C var*), 606, 640
getConnParams_tag.ucRole (*C var*), 606, 640
getConnParams_tag.ucUnMapChIdx (*C var*), 606, 640
getConnParams_tag.uiAnchorDelay (*C var*), 607, 641
getConnParams_tag.uiConnEvent (*C var*), 606, 640
getConnParams_tag.uiConnInterval (*C var*), 606, 640
getConnParams_tag.uiConnLatency (*C var*), 606, 640
getConnParams_tag.uiSuperTO (*C var*), 606, 640
getConnParams_tag.ulAnchorClk (*C var*), 606, 640
getConnParams_tag.ulRxInstant (*C var*), 607, 641
getConnParams_tag.ulTxAccCode (*C var*), 606, 640
getLeExtendedFeatureBit (*C macro*), 591
getLeExtendedFeatureByte (*C macro*), 591
getSecurityLevel (*C macro*), 447
getSecurityMode (*C macro*), 447
gExpandAsEnum_m (*C macro*), 633
gExpandAsTable_m (*C macro*), 633
gExtendedFilteredScan_c (*C macro*), 589
gExtendedUnfilteredScan_c (*C macro*), 589
gGapAdvertisingChannelMapDefault_c (*C macro*), 448
gGapAdvertisingIntervalDefault_c (*C macro*), 448
gGapAdvertisingIntervalRangeMaximum_c (*C macro*), 448
gGapAdvertisingIntervalRangeMinimum_c (*C macro*), 448
gGapChSelAlgorithmNo2 (*C macro*), 450
gGapConnEventLengthMax_d (*C macro*), 450
gGapConnEventLengthMin_d (*C macro*), 450
gGapConnIntervalMax_d (*C macro*), 449
gGapConnIntervalMin_d (*C macro*), 449
gGapConnLatencyMax_d (*C macro*), 449
gGapConnLatencyMin_d (*C macro*), 449
gGapConnSuperTimeoutMax_d (*C macro*), 450
gGapConnSuperTimeoutMin_d (*C macro*), 450
gGapCteMaxCount_c (*C macro*), 453
gGapCteMaxLength_c (*C macro*), 453
gGapCteMinCount_c (*C macro*), 453
gGapCteMinLength_c (*C macro*), 453
gGapDefaultAdvertisingParameters_d (*C macro*), 448
gGapDefaultConnectionLatency_d (*C macro*), 450
gGapDefaultConnectionRequestParameters_d (*C macro*), 450
gGapDefaultDeviceSecurity_d (*C macro*), 448
gGapDefaultExtAdvertisingParameters_d (*C macro*), 448

gGapDefaultMaxConnectionInterval_d (*C macro*), 450
gGapDefaultMinConnectionInterval_d (*C macro*), 450
gGapDefaultPeriodicAdvParameters_d (*C macro*), 448
gGapDefaultScanningParameters_d (*C macro*), 449
gGapDefaultSecurityRequirements_d (*C macro*), 448
gGapDefaultSupervisionTimeout_d (*C macro*), 450
gGapEattMaxBearers (*C macro*), 453
gGapExtAdvertisingIntervalDefault_c (*C macro*), 448
gGapExtAdvertisingIntervalRangeMaximum_c (*C macro*), 448
gGapExtAdvertisingIntervalRangeMinimum_c (*C macro*), 448
gGapInvalidSyncHandle (*C macro*), 450
gGapMaxSwitchingPatternLength_c (*C macro*), 453
gGapMinSwitchingPatternLength_c (*C macro*), 453
gGapPeriodicAdvIntervalDefault_c (*C macro*), 448
gGapPeriodicAdvIntervalRangeMaximum_c (*C macro*), 448
gGapPeriodicAdvIntervalRangeMinimum_c (*C macro*), 448
gGapRssiMax_d (*C macro*), 449
gGapRssiMin_d (*C macro*), 449
gGapRssiNotAvailable_d (*C macro*), 449
gGapScanContinuously_d (*C macro*), 449
gGapScanIntervalDefault_d (*C macro*), 449
gGapScanIntervalMax_d (*C macro*), 449
gGapScanIntervalMin_d (*C macro*), 449
gGapScanPeriodicDisabled_d (*C macro*), 449
gGapScanWindowDefault_d (*C macro*), 449
gGapScanWindowMax_d (*C macro*), 449
gGapScanWindowMin_d (*C macro*), 449
gGattDatabaseHashSize_c (*C macro*), 539
gGattDbAttributeCount_c (*C var*), 533
gGattDbInvalidHandle_d (*C macro*), 539
gGattDbInvalidHandleIndex_d (*C macro*), 539
gGattDynamicAttrSize (*C var*), 533
gGattDynamicValSize (*C var*), 533
gHci2Host_TaskQueue (*C var*), 585
gHCICSChannelMapSize (*C macro*), 590
gHciTransportUartChannel_c (*C macro*), 623
gIncompatibleIoCapabilities_c (*C macro*), 452
gInvalidDeviceId_c (*C macro*), 587
gInvalidNvmIndex_c (*C macro*), 587
gIoDisplayOnly_c (*C macro*), 452
gIoDisplayYesNo_c (*C macro*), 452
gIoKeyboardDisplay_c (*C macro*), 452
gIoKeyboardOnly_c (*C macro*), 452
gIoNone_c (*C macro*), 452
gIrk_c (*C macro*), 451
gL2caLePsmDynamicFirst_c (*C macro*), 633
gL2caLePsmDynamicLast_c (*C macro*), 633
gL2caLePsmSigAssignedEatt_c (*C macro*), 633
gL2caLePsmSigAssignedFirst_c (*C macro*), 633
gL2caLePsmSigAssignedLast_c (*C macro*), 633
gL2capCidAtt_c (*C macro*), 632
gL2capCidLePsmDynamicFirst_c (*C macro*), 633
gL2capCidLePsmDynamicLast_c (*C macro*), 633
gL2capCidNotApplicable_c (*C macro*), 633
gL2capCidNull_c (*C macro*), 632
gL2capCidSigAssignedFirst_c (*C macro*), 632
gL2capCidSigAssignedLast_c (*C macro*), 633
gL2capCidSignaling_c (*C macro*), 632

gL2capCidSmp_c (*C macro*), 632
gL2capDefaultMps_c (*C macro*), 633
gL2capDefaultMtu_c (*C macro*), 633
gL2capEnhancedMaxChannels_c (*C macro*), 633
gL2capHeaderLength_c (*C macro*), 633
gL2capMaximumMps_c (*C macro*), 633
gLeExtendedFeaturesSize_c (*C macro*), 591
gLeGeneralDiscoverableMode_c (*C macro*), 450
gLeLimitedDiscoverableMode_c (*C macro*), 450
gLePhy1MFlag_c (*C macro*), 588
gLePhy2MFlag_c (*C macro*), 588
gLePhyCodedFlag_c (*C macro*), 588
gLePsmSigAssignedNumbersTable_m (*C macro*), 633
gLinkEncryptionFailed_c (*C macro*), 452
gLowEncryptionKeySize_c (*C macro*), 452
gLtk_c (*C macro*), 451
gMaxAdvReportQueueSize (*C var*), 585
gMaxAdvSets_c (*C macro*), 587
gMaxEncryptionKeySize_d (*C macro*), 448
gMode_2_Mask_d (*C macro*), 447
gNetworkPrivacy_c (*C macro*), 590
gNoKeys_c (*C macro*), 451
gNone_c (*C macro*), 450
gOobNotAvailable_c (*C macro*), 452
gPairingNotSupported_c (*C macro*), 452
gPermissionFlagReadable_c (*C macro*), 539
gPermissionFlagReadWithAuthentication_c (*C macro*), 539
gPermissionFlagReadWithAuthorization_c (*C macro*), 539
gPermissionFlagReadWithEncryption_c (*C macro*), 539
gPermissionFlagWritable_c (*C macro*), 539
gPermissionFlagWriteWithAuthentication_c (*C macro*), 539
gPermissionFlagWriteWithAuthorization_c (*C macro*), 539
gPermissionFlagWriteWithEncryption_c (*C macro*), 539
gPermissionNone_c (*C macro*), 539
gRepeatedAttempts_c (*C macro*), 452
gScanAll_c (*C macro*), 589
gScanAllPDUs_c (*C macro*), 590
gScanOnlyDecisionPDUs_c (*C macro*), 590
gScanOnlyNonDecisionPDUs_c (*C macro*), 589
gScanRspDataChange_c (*C macro*), 588
gScanWithFilterAcceptList_c (*C macro*), 589
gSecurityLevel_LeSecureConnections_c (*C macro*), 451
gSecurityLevel_NoMitmProtection_c (*C macro*), 451
gSecurityLevel_NoSecurity_c (*C macro*), 451
gSecurityLevel_WithMitmProtection_c (*C macro*), 451
gSecurityMode_1_c (*C macro*), 451
gSecurityMode_1_Level_1_c (*C macro*), 451
gSecurityMode_1_Level_2_c (*C macro*), 451
gSecurityMode_1_Level_3_c (*C macro*), 451
gSecurityMode_1_Level_4_c (*C macro*), 452
gSecurityMode_2_c (*C macro*), 451
gSecurityMode_2_Level_1_c (*C macro*), 452
gSecurityMode_2_Level_2_c (*C macro*), 452
gSimultaneousLeBrEdrCapableController_c (*C macro*), 451
gSimultaneousLeBrEdrCapableHost_c (*C macro*), 451
gSkdSize_c (*C macro*), 590
gSmpLeScRandomConfirmValueSize_c (*C macro*), 622
gSmpLeScRandomValueSize_c (*C macro*), 622

gUnenhancedBearerId_c (*C macro*), 590
gUnspecifiedReason_c (*C macro*), 452
gUseDecisionPDU_UseFilterAcceptListForOtherPDUs_c (*C macro*), 589
gUseDeviceAddress_c (*C macro*), 588
gUseDeviceAddressNoDecisionPDUs_c (*C macro*), 589
gUseFilterAcceptList_c (*C macro*), 589
gUseFilterAcceptListAllPDUs_c (*C macro*), 589
gUseFilterAcceptListNoDecisionPDUs_c (*C macro*), 589
gUseHciCommandFlowControl (*C var*), 585
gUseOnlyDecisionPDUs_c (*C macro*), 589
gVendorHandoverMaxCsLlContextSize_c (*C macro*), 591
gVendorUnitaryTestSize_c (*C macro*), 591

H

handoverAnchorMonitorEvent_t (*C type*), 584
handoverAnchorMonitorEvent_tag (*C struct*), 608, 642
handoverAnchorMonitorEvent_tag.anchorClock625Us (*C var*), 609, 643
handoverAnchorMonitorEvent_tag.anchorDelay (*C var*), 609, 643
handoverAnchorMonitorEvent_tag.chIdx (*C var*), 609, 643
handoverAnchorMonitorEvent_tag.connectionHandle (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.connEvent (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.lqiActive (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.lqiRemote (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.rssiActive (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.rssiRemote (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.statusActive (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.statusRemote (*C var*), 609, 642
handoverAnchorMonitorEvent_tag.ucNbReports (*C var*), 609, 643
handoverAnchorMonitorPacketContinueEvent_t (*C type*), 584
handoverAnchorMonitorPacketContinueEvent_tag (*C struct*), 610, 643
handoverAnchorMonitorPacketContinueEvent_tag.connectionHandle (*C var*), 611, 644
handoverAnchorMonitorPacketContinueEvent_tag.packetCounter (*C var*), 611, 644
handoverAnchorMonitorPacketContinueEvent_tag.pduSize (*C var*), 611, 644
handoverAnchorMonitorPacketContinueEvent_tag.pPdu (*C var*), 611, 644
handoverAnchorMonitorPacketEvent_t (*C type*), 584
handoverAnchorMonitorPacketEvent_tag (*C struct*), 609, 643
handoverAnchorMonitorPacketEvent_tag.anchorClock625Us (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.anchorDelay (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.chIdx (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.connectionHandle (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.connEvent (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.lqiPacket (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.packetCounter (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.pduSize (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.phy (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.pPdu (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.rssiPacket (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.statusPacket (*C var*), 610, 643
handoverAnchorMonitorPacketEvent_tag.ucNbConnIntervals (*C var*), 610, 643
handoverAnchorNotificationStateChanged_t (*C type*), 584
handoverAnchorNotificationStateChanged_tag (*C struct*), 614, 646
handoverAnchorNotificationStateChanged_tag.connectionHandle (*C var*), 615, 646
handoverAnchorSearchStart_t (*C type*), 583
handoverAnchorSearchStart_tag (*C struct*), 607, 641
handoverAnchorSearchStart_tag.connectionHandle (*C var*), 607, 641
handoverAnchorSearchStart_tag.status (*C var*), 607, 641
handoverAnchorSearchStop_t (*C type*), 583
handoverAnchorSearchStop_tag (*C struct*), 607, 641

handoverAnchorSearchStop_tag.connectionHandle (*C var*), 607, 641
handoverAnchorSearchStop_tag.status (*C var*), 607, 641
handoverApplyConnectionUpdateProcedure_t (*C type*), 584
handoverApplyConnectionUpdateProcedure_tag (*C struct*), 614, 646
handoverApplyConnectionUpdateProcedure_tag.connectionHandle (*C var*), 614, 646
handoverConnect_t (*C type*), 583
handoverConnect_tag (*C struct*), 607, 641
handoverConnect_tag.connectionHandle (*C var*), 608, 641
handoverConnect_tag.status (*C var*), 608, 641
handoverConnectionUpdateProcedureEvent_t (*C type*), 584
handoverConnectionUpdateProcedureEvent_tag (*C struct*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.connectionHandle (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.currentEventCounter (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.instant (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.interval (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.latency (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.timeout (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.winOffset (*C var*), 611, 644
handoverConnectionUpdateProcedureEvent_tag.winSize (*C var*), 611, 644
handoverConnParamUpdateEvent_t (*C type*), 584
handoverConnParamUpdateEvent_tag (*C struct*), 612, 644
handoverConnParamUpdateEvent_tag.aChMapBm (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.aCrcInitVal (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.aucRemoteMasRxPHY (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.connectionHandle (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.seqNum (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.status (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.ucCentralSCA (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.ucChannelSelection (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.ucHop (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.ucRole (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.ucUnMapChIdx (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.uiAnchorDelay (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.uiConnEvent (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.uiConnInterval (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.uiConnLatency (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.uiSuperTO (*C var*), 612, 645
handoverConnParamUpdateEvent_tag.ulAnchorClk (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.ulRxInstant (*C var*), 613, 645
handoverConnParamUpdateEvent_tag.ulTxAccCode (*C var*), 612, 645
handoverGetCsLlContext_t (*C type*), 584
handoverGetCsLlContext_tag (*C struct*), 608, 642
handoverGetCsLlContext_tag.llContext (*C var*), 608, 642
handoverGetCsLlContext_tag.llContextLength (*C var*), 608, 642
handoverGetCsLlContext_tag.responseMask (*C var*), 608, 642
handoverGetCsLlContext_tag.status (*C var*), 608, 642
handoverGetData_t (*C type*), 583
handoverGetData_tag (*C struct*), 608, 641
handoverGetData_tag.pData (*C var*), 608, 642
handoverGetData_tag.status (*C var*), 608, 642
handoverGetTime_t (*C type*), 583
handoverGetTime_tag (*C struct*), 607, 641
handoverGetTime_tag.slot (*C var*), 607, 641
handoverGetTime_tag.status (*C var*), 607, 641
handoverGetTime_tag.us_offset (*C var*), 607, 641
handoverLlPendingDataIndication_t (*C type*), 584
handoverLlPendingDataIndication_tag (*C struct*), 615, 646
handoverLlPendingDataIndication_tag.dataSize (*C var*), 615, 647

handoverLlPendingDataIndication_tag.pData (*C var*), 615, 647
handoverResumeTransmitCompleteEvent_t (*C type*), 584
handoverResumeTransmitCompleteEvent_tag (*C struct*), 614, 646
handoverResumeTransmitCompleteEvent_tag.connectionHandle (*C var*), 614, 646
handoverSetData_t (*C type*), 583
handoverSetData_tag (*C struct*), 608, 642
handoverSetData_tag.pData (*C var*), 608, 642
handoverSetData_tag.status (*C var*), 608, 642
handoverSuspendTransmitCompleteEvent_t (*C type*), 584
handoverSuspendTransmitCompleteEvent_tag (*C struct*), 613, 645
handoverSuspendTransmitCompleteEvent_tag.connectionHandle (*C var*), 613, 646
handoverSuspendTransmitCompleteEvent_tag.noOfPendingAclPackets (*C var*), 613, 646
handoverSuspendTransmitCompleteEvent_tag.sizeOfDataNAckInOldestPacket (*C var*), 614, 646
handoverSuspendTransmitCompleteEvent_tag.sizeOfDataTxInOldestPacket (*C var*), 614, 646
handoverSuspendTransmitCompleteEvent_tag.sizeOfPendingAclPackets (*C var*), 613, 646
handoverTimeSyncEvent_t (*C type*), 584
handoverTimeSyncEvent_tag (*C struct*), 611, 644
handoverTimeSyncEvent_tag.rssi (*C var*), 612, 644
handoverTimeSyncEvent_tag.rxClkSlot (*C var*), 612, 644
handoverTimeSyncEvent_tag.rxUs (*C var*), 612, 644
handoverTimeSyncEvent_tag.txClkSlot (*C var*), 612, 644
handoverTimeSyncEvent_tag.txUs (*C var*), 612, 644
handoverUpdateConnParams_t (*C type*), 584
handoverUpdateConnParams_tag (*C struct*), 614, 646
handoverUpdateConnParams_tag.connectionHandle (*C var*), 614, 646
handoverUpdateConnParams_tag.status (*C var*), 614, 646
hciHostToControllerInterface_t (*C type*), 584
hciPacketType_t (*C enum*), 557
hciPacketType_t.gHciCommandPacket_c (*C enumerator*), 557
hciPacketType_t.gHciDataPacket_c (*C enumerator*), 557
hciPacketType_t.gHciEventPacket_c (*C enumerator*), 557
hciPacketType_t.gHciIsoDataPacket_c (*C enumerator*), 558
hciPacketType_t.gHciSynchronousDataPacket_c (*C enumerator*), 557
Host_TaskHandler (*C function*), 587

I

isMode_1 (*C macro*), 447
isMode_2 (*C macro*), 447
isSameMode (*C macro*), 447
isSupportedLeExtendedFeature (*C macro*), 591

L

L2ca_CancelConnection (*C function*), 630
L2ca_ConnectLePsm (*C function*), 629
L2ca_DeregisterLePsm (*C function*), 629
L2ca_DisconnectLeCbChannel (*C function*), 630
L2ca_EnhancedCancelConnection (*C function*), 632
L2ca_EnhancedChannelReconfigure (*C function*), 631
L2ca_EnhancedConnectLePsm (*C function*), 631
L2ca_RegisterLeCbCallbacks (*C function*), 628
L2ca_RegisterLePsm (*C function*), 629
L2ca_SendLeCbData (*C function*), 630
L2ca_SendLeCredit (*C function*), 631
l2caChannelStatus_t (*C enum*), 627
l2caChannelStatus_t.gL2ca_ChannelStatusChannelBusy_c (*C enumerator*), 627
l2caChannelStatus_t.gL2ca_ChannelStatusChannelIdle_c (*C enumerator*), 627
l2caControlCallback_t (*C type*), 628
l2caEattCallback_t (*C type*), 628

l2caEnhancedConnectionComplete_t (*C type*), 628
l2caEnhancedConnectionComplete_tag (*C struct*), 634, 689
l2caEnhancedConnectionComplete_tag.aCids (*C var*), 690
l2caEnhancedConnectionComplete_tag.deviceId (*C var*), 690
l2caEnhancedConnectionComplete_tag.initialCredits (*C var*), 690
l2caEnhancedConnectionComplete_tag.noOfChannels (*C var*), 690
l2caEnhancedConnectionComplete_tag.peerMps (*C var*), 690
l2caEnhancedConnectionComplete_tag.peerMtu (*C var*), 690
l2caEnhancedConnectionComplete_tag.result (*C var*), 690
l2caEnhancedConnectionRequest_t (*C type*), 628
l2caEnhancedConnectionRequest_tag (*C struct*), 634, 689
l2caEnhancedConnectionRequest_tag.aCids (*C var*), 689
l2caEnhancedConnectionRequest_tag.deviceId (*C var*), 689
l2caEnhancedConnectionRequest_tag.initialCredits (*C var*), 689
l2caEnhancedConnectionRequest_tag.lePsm (*C var*), 689
l2caEnhancedConnectionRequest_tag.noOfChannels (*C var*), 689
l2caEnhancedConnectionRequest_tag.peerMps (*C var*), 689
l2caEnhancedConnectionRequest_tag.peerMtu (*C var*), 689
l2caEnhancedReconfigureRequest_t (*C type*), 628
l2caEnhancedReconfigureRequest_tag (*C struct*), 634, 690
l2caEnhancedReconfigureRequest_tag.aCids (*C var*), 690
l2caEnhancedReconfigureRequest_tag.deviceId (*C var*), 690
l2caEnhancedReconfigureRequest_tag.newMps (*C var*), 690
l2caEnhancedReconfigureRequest_tag.newMtu (*C var*), 690
l2caEnhancedReconfigureRequest_tag.noOfChannels (*C var*), 690
l2caEnhancedReconfigureRequest_tag.result (*C var*), 690
l2caEnhancedReconfigureResponse_t (*C type*), 628
l2caEnhancedReconfigureResponse_tag (*C struct*), 634, 690
l2caEnhancedReconfigureResponse_tag.deviceId (*C var*), 690
l2caEnhancedReconfigureResponse_tag.result (*C var*), 690
l2caErrorSource_t (*C enum*), 626
l2caErrorSource_t.gL2ca_CancelConnection_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_DisconnectLePsm_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_EnhancedCancelConnection_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_EnhancedReconfigureReq_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_HandleLeFlowControlCredit_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_HandleRecvLeCbData_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_HandleSendLeCbData_c (*C enumerator*), 626
l2caErrorSource_t.gL2ca_SendLeFlowControlCredit_c (*C enumerator*), 626
l2caGenericCallback_t (*C type*), 628
l2caHandoverConnectionComplete_t (*C type*), 628
l2caHandoverConnectionComplete_tag (*C struct*), 633, 687
l2caHandoverConnectionComplete_tag.cId (*C var*), 688
l2caHandoverConnectionComplete_tag.credits (*C var*), 688
l2caHandoverConnectionComplete_tag.deviceId (*C var*), 688
l2caHandoverConnectionComplete_tag.peerMps (*C var*), 688
l2caHandoverConnectionComplete_tag.peerMtu (*C var*), 688
l2caLeCbChannelStatusNotification_t (*C type*), 628
l2caLeCbChannelStatusNotification_tag (*C struct*), 634, 689
l2caLeCbChannelStatusNotification_tag.cId (*C var*), 689
l2caLeCbChannelStatusNotification_tag.deviceId (*C var*), 689
l2caLeCbChannelStatusNotification_tag.status (*C var*), 689
l2caLeCbConnectionComplete_t (*C type*), 628
l2caLeCbConnectionComplete_tag (*C struct*), 633, 687
l2caLeCbConnectionComplete_tag.cId (*C var*), 687
l2caLeCbConnectionComplete_tag.deviceId (*C var*), 687
l2caLeCbConnectionComplete_tag.initialCredits (*C var*), 687
l2caLeCbConnectionComplete_tag.peerMps (*C var*), 687

l2caLeCbConnectionComplete_tag.peerMtu (*C var*), 687
l2caLeCbConnectionComplete_tag.result (*C var*), 687
l2caLeCbConnectionRequest_t (*C type*), 628
l2caLeCbConnectionRequest_tag (*C struct*), 633, 687
l2caLeCbConnectionRequest_tag.deviceId (*C var*), 687
l2caLeCbConnectionRequest_tag.initialCredits (*C var*), 687
l2caLeCbConnectionRequest_tag.lePsm (*C var*), 687
l2caLeCbConnectionRequest_tag.peerMps (*C var*), 687
l2caLeCbConnectionRequest_tag.peerMtu (*C var*), 687
l2caLeCbConnectionRequestResult_t (*C enum*), 625
l2caLeCbConnectionRequestResult_t.gCommandRejected_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInsufficientAuthentication_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInsufficientAuthorization_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInsufficientEncryption_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInsufficientEncryptionKeySize_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInvalidParameters_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gInvalidSourceCid_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gLePsmNotSupported_c (*C enumerator*), 625
l2caLeCbConnectionRequestResult_t.gNoResourcesAvailable_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gResponseTimeout_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gSourceCidAreadyAllocated_c (*C enumerator*), 626
l2caLeCbConnectionRequestResult_t.gSuccessful_c (*C enumerator*), 625
l2caLeCbConnectionRequestResult_t.gUnacceptableParameters_c (*C enumerator*), 626
l2caLeCbControlCallback_t (*C type*), 628
l2caLeCbDataCallback_t (*C type*), 628
l2caLeCbDisconnection_t (*C type*), 628
l2caLeCbDisconnection_tag (*C struct*), 633, 688
l2caLeCbDisconnection_tag.cId (*C var*), 688
l2caLeCbDisconnection_tag.deviceId (*C var*), 688
l2caLeCbError_t (*C type*), 628
l2caLeCbError_tag (*C struct*), 634, 689
l2caLeCbError_tag.deviceId (*C var*), 689
l2caLeCbError_tag.errorSource (*C var*), 689
l2caLeCbError_tag.result (*C var*), 689
l2caLeCbLocalCreditsNotification_t (*C type*), 628
l2caLeCbLocalCreditsNotification_tag (*C struct*), 634, 688
l2caLeCbLocalCreditsNotification_tag.cId (*C var*), 688
l2caLeCbLocalCreditsNotification_tag.deviceId (*C var*), 688
l2caLeCbLocalCreditsNotification_tag.localCredits (*C var*), 689
l2caLeCbLowPeerCredits_t (*C type*), 628
l2caLeCbLowPeerCredits_tag (*C struct*), 634, 688
l2caLeCbLowPeerCredits_tag.cId (*C var*), 688
l2caLeCbLowPeerCredits_tag.deviceId (*C var*), 688
l2caLeCbNoPeerCredits_t (*C type*), 628
l2caLeCbNoPeerCredits_tag (*C struct*), 634, 688
l2caLeCbNoPeerCredits_tag.cId (*C var*), 688
l2caLeCbNoPeerCredits_tag.deviceId (*C var*), 688
l2capControlMessage_t (*C type*), 628
l2capControlMessage_tag (*C struct*), 634, 690
l2capControlMessage_tag.messageData (*C union*), 634, 691
l2capControlMessage_tag.messageData (*C var*), 691
l2capControlMessage_tag.messageData.channelStatusNotification (*C var*), 635, 691
l2capControlMessage_tag.messageData.connectionComplete (*C var*), 634, 691
l2capControlMessage_tag.messageData.connectionRequest (*C var*), 634, 691
l2capControlMessage_tag.messageData.disconnection (*C var*), 634, 691
l2capControlMessage_tag.messageData.enhancedConnComplete (*C var*), 635, 691
l2capControlMessage_tag.messageData.enhancedConnRequest (*C var*), 635, 691
l2capControlMessage_tag.messageData.error (*C var*), 635, 691

l2capControlMessage_tag.messageData.handoverConnectionComplete (*C var*), 635, 691
l2capControlMessage_tag.messageData.localCreditsNotification (*C var*), 634, 691
l2capControlMessage_tag.messageData.lowPeerCredits (*C var*), 634, 691
l2capControlMessage_tag.messageData.noPeerCredits (*C var*), 634, 691
l2capControlMessage_tag.messageData.reconfigureRequest (*C var*), 635, 691
l2capControlMessage_tag.messageData.reconfigureResponse (*C var*), 635, 691
l2capControlMessage_tag.messageType (*C var*), 691
l2capControlMessageType_t (*C enum*), 627
l2capControlMessageType_t.gL2ca_ChannelStatusNotification_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_EnhancedReconfigureRequest_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_EnhancedReconfigureResponse_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_Error_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_HandoverConnectionComplete_c (*C enumerator*), 628
l2capControlMessageType_t.gL2ca_LePsmConnectionComplete_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LePsmConnectRequest_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LePsmDisconnectNotification_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LePsmEnhancedConnectionComplete_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LePsmEnhancedConnectRequest_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LocalCreditsNotification_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_LowPeerCredits_c (*C enumerator*), 627
l2capControlMessageType_t.gL2ca_NoPeerCredits_c (*C enumerator*), 627
l2capReconfigureResponse_t (*C enum*), 626
l2capReconfigureResponse_t.gDestinationCidInvalid_c (*C enumerator*), 627
l2capReconfigureResponse_t.gMtuReductionNotAllowed_c (*C enumerator*), 627
l2capReconfigureResponse_t.gMultipleChannelMpsReductionNotAllowed_c (*C enumerator*), 627
l2capReconfigureResponse_t.gReconfigurationSuccessful_c (*C enumerator*), 626
l2capReconfigureResponse_t.gReconfigurationTimeout_c (*C enumerator*), 627
l2capReconfigureResponse_t.gReconfReserved_c (*C enumerator*), 627
l2capReconfigureResponse_t.gUnacceptableReconfParameters_c (*C enumerator*), 627
leExtendedSupportedFeatures_tag (*C enum*), 562
leExtendedSupportedFeatures_tag.gLeMonitoringAdvertisers_c (*C enumerator*), 562
leSupportedFeatures_t (*C type*), 583
leSupportedFeatures_tag (*C enum*), 561
leSupportedFeatures_tag.gAdvertisingCodingSelection_c (*C enumerator*), 562
leSupportedFeatures_tag.gAdvertisingCodingSelectionHostSupport_c (*C enumerator*), 562
leSupportedFeatures_tag.gLe2MbPhy_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeChannelSelAlg2_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeCodedPhy_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeConnectionParametersRequestProcedure_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeDataPacketLengthExtension_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeDecisionBasedAdvertisingFiltering_c (*C enumerator*), 562
leSupportedFeatures_tag.gLeEncryption_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeExtendedAdv_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeExtendedRejectIndication_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeExtendedScannerFilterPolicies_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeLlPrivacy_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeMinNumOfUsedChanProcedure_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePathLossMonitoring_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePawrAdvertiser_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePawrScanner_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePeriodicAdv_c (*C enumerator*), 561
leSupportedFeatures_tag.gLePeriodicAdvSyncTransferReceiver_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePeriodicAdvSyncTransferSender_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePeripheralInitiatedFeaturesExchange_c (*C enumerator*), 561
leSupportedFeatures_tag.gLePing_c (*C enumerator*), 561
leSupportedFeatures_tag.gLePowerClass1_c (*C enumerator*), 561
leSupportedFeatures_tag.gLePowerControlRequest1_c (*C enumerator*), 562
leSupportedFeatures_tag.gLePowerControlRequest2_c (*C enumerator*), 562

leSupportedFeatures_tag.gLeStableModulationIdxRx_c (*C enumerator*), 561
leSupportedFeatures_tag.gLeStableModulationIdxTx_c (*C enumerator*), 561
LL_PHY_1M (*C macro*), 590
LL_PHY_2M (*C macro*), 591
LL_PHY_S2 (*C macro*), 591
LL_PHY_S8 (*C macro*), 591

M

mServerServiceChangedCCCDHandle (*C var*), 533
mServerServiceChangedCharHandle (*C var*), 533

P

PACKED_STRUCT (*C macro*), 600
periodicAdvSetInfoTransferEvent_t (*C type*), 583
periodicAdvSetInfoTransferEvent_tag (*C struct*), 604, 639
periodicAdvSetInfoTransferEvent_tag.deviceId (*C var*), 639
periodicAdvSetInfoTransferEvent_tag.status (*C var*), 639
periodicAdvSetSyncTransferParamsEvent_t (*C type*), 583
periodicAdvSetSyncTransferParamsEvent_tag (*C struct*), 604, 639
periodicAdvSetSyncTransferParamsEvent_tag.deviceId (*C var*), 639
periodicAdvSetSyncTransferParamsEvent_tag.status (*C var*), 639
periodicAdvSyncTransferEvent_t (*C type*), 583
periodicAdvSyncTransferEvent_tag (*C struct*), 604, 638
periodicAdvSyncTransferEvent_tag.deviceId (*C var*), 639
periodicAdvSyncTransferEvent_tag.status (*C var*), 639
pProcedureData (*C var*), 498
procDataStruct_t (*C struct*), 495, 683
procDataStruct_t.array (*C union*), 496, 683
procDataStruct_t.array (*C var*), 496, 683
procDataStruct_t.array.aBytes (*C var*), 496, 683
procDataStruct_t.array.aChars (*C var*), 496, 683
procDataStruct_t.array.aDescriptors (*C var*), 496, 683
procDataStruct_t.array.aHandles (*C var*), 496, 684
procDataStruct_t.array.aServices (*C var*), 496, 683
procDataStruct_t.bAllocatedArray (*C var*), 496, 683
procDataStruct_t.charUuid (*C var*), 495, 683
procDataStruct_t.charUuidType (*C var*), 496, 683
procDataStruct_t.index (*C var*), 495, 683
procDataStruct_t.max (*C var*), 495, 683
procDataStruct_t.pOutActualCount (*C union*), 496, 683
procDataStruct_t.pOutActualCount (*C var*), 496, 683
procDataStruct_t.pOutActualCount.pCount8b (*C var*), 496, 683
procDataStruct_t.pOutActualCount.pCount16b (*C var*), 496, 683
procDataStruct_t.reliableLongWrite (*C var*), 496, 683
procDataStruct_t.reqParams (*C union*), 497, 684
procDataStruct_t.reqParams (*C var*), 496, 683
procDataStruct_t.reqParams.ewParams (*C var*), 497, 684
procDataStruct_t.reqParams.fbtvParams (*C var*), 497, 684
procDataStruct_t.reqParams.fiParams (*C var*), 497, 684
procDataStruct_t.reqParams.pwParams (*C var*), 497, 684
procDataStruct_t.reqParams.rbgtParams (*C var*), 497, 684
procDataStruct_t.reqParams.rbParams (*C var*), 497, 684
procDataStruct_t.reqParams.rbtParams (*C var*), 497, 684
procDataStruct_t.reqParams.rmParams (*C var*), 497, 684
procDataStruct_t.reqParams.rParams (*C var*), 497, 684
procDataStruct_t.reqParams.swParams (*C var*), 497, 684
procDataStruct_t.reqParams.wParams (*C var*), 497, 684
procStatus_t (*C struct*), 495, 682

procStatus_t.isOngoing (*C var*), 495, 682
procStatus_t.ongoingProcedurePhase (*C var*), 495, 682
procStatus_t.ongoingProcedureType (*C var*), 495, 682

S

STATIC (*C macro*), 588

U

Utils_BeExtractFourByteValue (*C macro*), 600
Utils_BeExtractThreeByteValue (*C macro*), 600
Utils_BeExtractTwoByteValue (*C macro*), 600
Utils_BePackFourByteValue (*C macro*), 601
Utils_BePackThreeByteValue (*C macro*), 601
Utils_BePackTwoByteValue (*C macro*), 601
Utils_Copy8 (*C macro*), 601
Utils_Copy16 (*C macro*), 601
Utils_Copy32 (*C macro*), 601
Utils_Copy64 (*C macro*), 601
Utils_ExtractEightByteValue (*C macro*), 600
Utils_ExtractFourByteValue (*C macro*), 600
Utils_ExtractThreeByteValue (*C macro*), 600
Utils_ExtractTwoByteValue (*C macro*), 600
Utils_PackEightByteValue (*C macro*), 600
Utils_PackFourByteValue (*C macro*), 600
Utils_PackThreeByteValue (*C macro*), 600
Utils_PackTwoByteValue (*C macro*), 600
Utils_RevertByteArray (*C macro*), 601
Uuid16 (*C macro*), 600
Uuid32 (*C macro*), 600
UuidArray (*C macro*), 600

V

vendorUnitaryTestEvent_t (*C type*), 584
vendorUnitaryTestEvent_tag (*C struct*), 615, 647
vendorUnitaryTestEvent_tag.aParam (*C var*), 647
vendorUnitaryTestEvent_tag.paramLength (*C var*), 647
vendorUnitaryTestEvent_tag.status (*C var*), 647